



**PTSD: Pothole Tracking, Sizing and
Detection
Final Report
University of Central Florida
Dr. Chung Yong Chan
Group 2**

Authors:

Travis Grant - Computer Engineering
John Billeci - Photonic Science and Engineering
Samuel Welch - Electrical Engineering
Jose Kostyun - Computer Engineering
Brandon Skervin - Mechanical Engineering

Reviewer Committee

Dr. Peter Delfyett
Dr. Mark Maddox
Dr. Sudeshna Pal

CREOL
ECE
MAE

Committee
Committee
Committee

Mentor: Dr. Chung Yong Chan

Table of Contents

1 - Executive Summary.....	1
2. Project Description.....	2
2.1 Motivation and Background.....	2
2.2 Features and Functionalities.....	4
2.2.1 Optical Schematic.....	5
2.2.2 Customer Input:.....	5
2.2.3 Comparable Products/Projects:.....	6
2.3 Goals and Objectives.....	7
2.4 Specifications.....	13
2.5 Block Diagrams.....	16
2.5.1 Hardware Diagram.....	16
2.5.2 Software Flowchart.....	17
2.5.3 Flowchart Description.....	17
2.5.4 Mobile Application Software Flowchart.....	19
2.5.5 Application Flowchart Description.....	19
2.5.6 Mobile Application Design Diagram.....	21
2.6 House of Quality.....	23
3. Research.....	23
3.1 Electrical Hardware Technology Comparisons.....	23
3.1.1 Microcontroller vs FPGA vs SBC.....	23
3.1.2 Device Power Source.....	28
3.1.3 Pothole Data Access.....	33
3.1.4 Buzzer Selection.....	37
3.2 Optical Technology and Part Selection.....	38
3.2.1 Detection Technology Selection.....	38
3.2.2 Detector Part Selection.....	44
3.2.3 Illumination Technology and Part Selection.....	51
3.3 Software Technology Comparisons.....	59
3.3.1 Computer Vision Models.....	59
3.3.2 Modern Mobile Development IDEs and Frameworks.....	66
3.3.3 Mobile Application Architectures.....	70
3.3.4 Database Architectures for Mobile Applications.....	76
3.3.5 Evaluation of GPS Frameworks for Mobile Application Integration.....	81
3.3.6 Optimization of Parallel Threads.....	87
3.4 Mechanical Part Selection.....	90

3.4.1 Mechanical Mount and Housing Material Selection.....	91
3.4.2 Mechanical Properties and Behavior.....	93
3.4.3 Tensile Testing of Prospective Materials.....	94
3.4.4 Mechanical Design Calculations and Analysis.....	96
3.4.5 Geometry Optimization for Structural Stiffness.....	99
3.4.6 Mounting Configuration and Suction Cup Strategy.....	100
3.4.7 Failure Modes and Effect Analysis.....	101
4. Standards and Design Constraints.....	103
4.1 Industry Standards.....	103
4.1.1 Electrical Hardware Standards.....	103
4.1.2 Wired Communication Standards.....	104
4.1.3 Bluetooth Communication Standards.....	105
4.1.4 Laser Safety Standards.....	106
4.1.5 Mechanical Standards.....	107
4.2 Design Constraints.....	108
4.2.1 Time.....	108
4.2.2 Cost and Resources.....	110
4.2.3 Processing Power.....	112
4.3 Mechanical Design Constraints.....	114
4.3.1 Environmental Sealing / Waterproofing.....	115
4.3.2 Weight Limit.....	115
4.3.3 Operating Temperature Range.....	116
4.3.4 Serviceability.....	116
5. Pros and Cons of ChatGPT.....	116
5.1 Case Study 1.....	118
5.2 Case Study 2.....	119
5.3 Case Study 3.....	119
6. Hardware Design.....	120
6.1 Overall System Design.....	121
6.2 Electrical Design and Power Delivery.....	123
6.2.1 Overall Electrical Design.....	126
6.2.2 Main PCB Schematic.....	126
6.2.5 3.3V Regulator Schematic.....	131
6.2.6 BLE PCB Schematic.....	132
6.3 Optical Design.....	133
6.3.1 Illumination System.....	133
6.3.2 Detection System.....	134
6.3.3 Full System Operation.....	135
6.4 Mechanical Design.....	137

6.4.1 Mounting for Camera and Laser Housing.....	137
6.4.2 Main PCB Housing Unit for Internal Control System.....	139
6.4.3 IR/RGB Dual-Camera Housing Unit.....	141
6.4.4 GoPro Compatible Camera Housing Unit Mount.....	142
6.4.5 Laser Housing Unit.....	143
7. Software Design.....	145
7.1 Overall Software Design.....	145
7.2 MSP430 Software Design.....	147
7.3 Raspberry Pi 5 Software Design.....	149
7.3.1 High-Level Module Interaction Diagram.....	149
7.3.2 IR Camera Thread Flowchart.....	151
7.3.3 RGB Camera Thread Data Structure.....	153
7.3.4 Inference Thread Flowchart.....	154
7.3.5 Post-Processing Thread Flowchart.....	156
7.4 App Software Design.....	158
8. System Fabrication.....	161
8.1 PCB Design.....	161
8.1.1 Main PCB Design.....	162
8.1.2 Laser PCB Design.....	163
8.1.3 5.2V Regulator PCB Design.....	164
8.1.4 5V Laser Regulator PCB Design.....	165
8.1.5 3.3V Regulator PCB Design.....	166
8.1.6 Bluetooth PCB Design.....	166
8.2 Assembled PCB.....	167
8.2.1 Main PCB.....	167
8.2.2 Laser Driver PCB.....	168
8.3 Mechanical Systems Fabrication.....	169
8.3.1. Mounting System.....	169
8.3.2 Camera Housing.....	170
8.3.3. Laser Housing.....	170
8.3.4 Main PCB Housing.....	171
9. System Testing and Evaluation.....	171
9.1 Computer Vision Training and Testing.....	171
9.2 Optical Testing.....	175
9.3 SPI Interface Validation.....	177
9.4 Full System Testing.....	180
10. Administrative Content.....	181
10.1 Budget.....	181
10.2 Bill of Materials.....	182

10.3 Milestones and Schedule.....	183
10.3.1 Senior Design 1.....	183
10.3.2 Senior Design 2.....	184
10.4 Table of Work.....	185
11. Conclusion.....	185
Appendix A References.....	188
Appendix B Copyright Permission.....	198
Appendix C ChatGPT Prompts and Outcomes.....	199
Appendix D Codes.....	208

List of Figures

Figure 2.1: Optical Schematic.....	5
Figure 2.2: Mounting Device Prototype.....	15
Figure 2.3: Mounting Device and Housing Unit Secured on Vehicle.....	15
Figure 2.4: Hardware Block Diagram.....	16
Figure 2.5: Software Flowchart.....	17
Figure 2.6: Mobile Application Flowchart.....	19
Figure 2.7: Mobile application design.....	21
Figure 2.8: House of Quality.....	23
Figure 3.1: Architecture of the SSD300 model. Adapted with permission from [19].....	61
Figure 3.2: YOLOv5 Architecture (Nepal & Eslamiat, 2022)[14], with permission from [15].....	62
Figure 3.3: Stress-Strain Graph for 6061-T6 Aluminum and Polycarbonate.....	95
Figure 6.1: Overall System Design Illustration.....	121
Figure 6.2: Overall Electrical Design.....	126
Figure 6.3: Main PCB Schematic.....	128
Figure 6.4: Laser Driver PCB Schematic.....	128
Figure 6.5: 5.2V Regulator Schematic.....	130
Figure 6.6: 3.3V Regulator.....	131
Figure 6.7: BLE PCB Schematic.....	132
Figure 6.8: Optical Design Schematic.....	133
Figure 6.9: Laser diode illumination system.....	134
Figure 6.10: Camera detection system.....	135
Figure 6.11: Laser illuminator incident over the target area.....	136
Figure 6.12: View of the target area from the camera system.....	136
Figure 6.13: Pothole depth measurement using trigonometry.....	137
Figure 6.14 : Schematics of Camera and Laser Mount.....	137
Figure 6.15: Schematic of the Main PCB Housing Unit.....	139
Figure 6.16: Schematic of the Dual Camera Housing Unit.....	141
Figure 6.17: Schematic of the Camera Housing Unit Mount.....	142
Figure 6.18: Schematic of the Laser Housing Unit.....	143
Figure 7.1: Overall Software Flowchart.....	145
Figure 7.2: Overall Software Use Case Diagram.....	146
Figure 7.3: MSP430 MCU Software Flowchart.....	147
Figure 7.4: High-Level Computer Vision Block Diagram.....	149
Figure 7.5: IR Camera Thread Flowchart.....	151
Figure 7.6: Ring Buffer Workflow Diagram.....	153
Figure 7.7: Inference Thread Flowchart.....	154

Figure 7.8: Post-Processing Thread Flowchart.....	156
Figure 7.9 Example of Pothole Detection and Measurement.....	157
Figure 7.10: Mobile App Block Diagram.....	158
Figure 7.11: Mobile App Use Case Diagram.....	159
Figure 8.1: Main PCB Layout.....	162
Figure 8.2: Laser Driver PCB Layout.....	163
Figure 8.3: 5.2V Regulator PCB Layout.....	164
Figure 8.4: 5.2V Laser Regulator PCB Layout.....	165
Figure 8.5: 3.3V Regulator PCB Layout.....	166
Figure 8.6: Bluetooth PCB Layout.....	167
Figure 8.7: Assembled Main PCB.....	168
Figure 8.8: Assembled Laser PCB.....	169
Figure 8.9: Full Assembled Mounting Arm.....	169
Figure 8.10: Printed Camera Housing.....	170
Figure 8.11: Printed Laser Housing.....	170
Figure 8.12: Printed Main PCB Housing.....	171
Figure 9.1: Overall Model Training Results.....	172
Figure 9.2: Precision-Recall Curve.....	173
Figure 9.3: Bounding Box Visualizations.....	174
Figure 9.4: Example Detection Results.....	174
Figure 9.5: Test surface illuminated by IR laser line at an angle.....	175
Figure 9.6: Machine vision algorithm output and measurements.....	176
Figure 9.7: Laser operation on fresh asphalt in daytime, Camera output in top left.....	177
Figure 9.8: Working SPI connection between Raspberry PI 5 and MSP430FR6989.....	180
Figure 9.9: Device mounted to a vehicle.....	180
Figure 9.10: Data collection in app (Right) Pothole location and size data on map (Left).....	181

List of Tables

Table 2.4.1: System Specifications.....	13
Table 2.4.2: Component Specifications.....	13
Table 2.4.3: Housing Unit Specifications.....	14
Table 2.4.4: Mounting System Specifications.....	14
Table 3.1.1: MCU vs FPGA.....	24
Table 3.1.2: MCU Comparison.....	26
Table 3.1.3: Development Board Comparison.....	28
Table 3.1.4: Energy Source.....	29
Table 3.1.5: Voltage Regulator Comparison.....	32
Table 3.1.6: Pothole Data Access Comparison.....	34
Table 3.1.7: Bluetooth Module Comparison.....	35
Table 3.1.8: GPS Module Comparison.....	36
Table 3.1.9: Buzzer Comparison Table.....	37
Table 3.2.1: LiDAR vs Machine Vision.....	41
Table 3.2.2: Wavelength Comparison.....	44
Table 3.2.3: Camera Comparison.....	47
Table 3.2.4: RGB Camera Comparison Table.....	49
Table 3.2.5: Optical Filter Comparison.....	50
Table 3.2.6: Absorptive Filter Comparison.....	51
Table 3.2.7: Illumination Source Comparison.....	53
Table 3.2.8: Laser Diode Selection.....	54
Table 3.2.9: Beam Expansion Selection.....	56
Table 3.2.10: Powell Lens Selection.....	56

Table 3.2.11: Aspheric Collimating Lens Selection.....	58
Table 3.3.1: Single-Stage vs Two-Stage Object Detection Models.....	60
Table 3.3.2: YOLOv5 vs SSD300 Object Detection Models.....	63
Table 3.3.3: Object Detection Model Optimization Methods.....	64
Table 3.3.4: Dynamic vs Static Quantization.....	66
Table 3.3.5 Comparative Evaluation of Mobile Development IDEs.....	69
Table 3.3.6 Core Mobile Application Types.....	74
Table 3.3.7 Comparative Table of Databases.....	80
Table 3.3.8 GPS SDK Comparison.....	85
Table 3.3.9: Comparing Parallel Processing Methods for Python.....	89
Table 3.4.1: “6061-T6 Aluminum vs Carbon Fiber Composite vs ABS Plastic”.....	92
Table 3.4.2: “Polycarbonate vs Acrylic vs ABS Plastic ”.....	92
Table 3.4.3: “FMEA Table.....	102
Table 4.3.1: Mechanical Design Constraints.....	114
Table 6.2.1: First 5V Regulator: Laser System.....	124
Table 6.2.2: Second 5V Regulator: Raspberry Pi.....	124
Table 6.3.1: 3.3V Regulator.....	125
Table 10.2.1: Bill of Materials.....	182
Table 10.3.1: Senior Design Schedule.....	184
Table 10.3.2: Senior Design 2 Schedule.....	184
Table 10.4.1: Table of Work.....	185

1 - Executive Summary

Potholes and road surface damage continue to be a major concern for both driver safety and city budgets. Current inspection methods, which include manual surveys, user-reported complaints, and specialized scanning vehicles, are too slow, expensive, or inconsistent. To address these issues, our team has designed and built a Pothole Detection and Tracking System that can automatically identify and log potholes while a vehicle is in motion. This system was developed as part of a senior design project by undergraduate engineering students at the University of Central Florida.

The main goal of the project is to create a compact, low-cost, and automated platform that detects road surface defects in real time, confirms each detection with a machine learning model, and sends the results to a mobile app for display on a live map. The overall design is built around three major components: an infrared and RGB camera scanning system, an onboard processing platform using a Raspberry Pi and an MSP430FR2355 PCB, and a mobile application with a cloud-connected database for storage and visualization.

Camera and Infrared Scanning System

The system uses a structured-light method to detect potholes. An infrared laser projects a thin horizontal line of light across the road behind the vehicle. When the vehicle passes over a road defect, this line is visibly distorted. An infrared camera continuously records this projected line, allowing the system to detect changes in road height caused by dips or holes. When a distortion is found, the onboard processor immediately triggers the RGB camera to capture a clear image of that same section of the road. These RGB images are then analyzed by an AI model to verify that the irregularity is a pothole, helping to reduce false detections.

Both cameras and the IR laser module are mounted in a weather-resistant enclosure that attaches to the rear of a vehicle using an adjustable telescoping bracket. This mounting solution minimizes vibration, protects the hardware from environmental conditions, and allows easy installation on different vehicle types.

Processing Hardware

The detection system relies on a Raspberry Pi as its main processor. The Pi runs image processing software to analyze the distortion in the IR laser line in real time. Once a potential pothole is detected and an RGB image is captured, the Pi uses a YOLOv5n-based neural network to confirm whether the event is a true pothole.

To support this, a custom PCB built around an MSP430 handles control tasks. The MSP manages the laser driver, triggers the cameras, and communicates with the Raspberry Pi over SPI.

Mobile App and Cloud System

On pothole confirmation by the onboard AI running on the Raspberry Pi, the detection data is packaged by the MSP430 and transmitted to the custom mobile application through Bluetooth Low Energy. The app receives the specialized packet, converts the integer-formatted GPS coordinates supplied by the vehicle-mounted GPS module, and saves the full entry locally. When a network connection is available, the application synchronizes the detection with the cloud database, ensuring that all records remain up to date across devices and user roles. Through this interface, drivers, field technicians, and administrators can view potholes on an interactive map, verify or dismiss individual detections, and monitor the progress of repairs as they are completed.

The mobile application enables the system to operate reliably even without continuous internet access, since detections can be cached offline and uploaded automatically when connectivity is restored. It also provides a unified platform for managing pothole data across an entire fleet, supporting confirmation workflows, repair tracking, and centralized oversight within a single, purpose-built interface.

2. Project Description

2.1 Motivation and Background

Maintaining safe roads is a constant challenge for city governments, particularly when operating on limited budgets. One of the most widespread issues is potholes, which can cause vehicle damage, increase accident risk, and put a strain on municipal maintenance operations. Detecting and tracking these hazards quickly and accurately is essential to ensuring public safety. However, current methods rely on manual surveys, inconsistent user reports, or expensive forms of data collection.

Our project, the Pothole Detection and Tracking System, aims to automate this process with a cost-effective, vehicle-mounted solution that simplifies the gathering, verification, and reporting of pothole data. This system uses infrared light to cast a structured light pattern onto the road surface behind a moving vehicle. A filtered, wide-angle infrared camera captures reflections of this light, and deviations in the laser pattern indicate the presence of a potential pothole.

The captured IR data is processed in real time by an onboard Raspberry Pi, which runs a laser line deviation detection algorithm.

If a potential pothole is detected, the system triggers AI-based image confirmation using buffered RGB frames from a separate visible-light camera. A lightweight YOLOv5n model verifies whether the anomaly is a true pothole. The system also includes an MSP430 microcontroller on the custom PCB, which handles low-level control tasks such as laser driver management and UART communication with the Raspberry Pi. Confirmed detections are assembled on the MSP430 by combining Raspberry Pi-generated dimensional data with GPS coordinates from the onboard MTK3339 module, then transmitted via the bluetooth BM71 module to the paired smartphone. The mobile application receives the complete event packet and synchronizes it with the cloud-based database whenever network connectivity is available.

The system is designed with accessibility and real-world use in mind. A custom, rear-mounted bracket allows for both temporary and permanent attachment to most standard vehicles. Power can be drawn from a vehicle's battery system or an independent power supply if needed. We chose to keep the total cost of components below \$1,200 to ensure the device remains affordable for widespread development by local governments as well as by smaller contractors.

Unlike more expensive systems, such as vehicle-integrated road monitoring offered by manufacturers or LiDAR-based platforms like XenoTrack, our design provides a lightweight, standalone option that customers can adopt without overhauling existing infrastructure. Compared to user-driven apps like Waze or post-processed video analysis tools such as RoadBotics, our project offers consistent, autonomous detection with real-time reporting and built-in location services.

Initial development has focused on creating a functional prototype that includes a camera and IR laser setup, embedded software for pattern detection, iOS application for data centralization and management, and secure Bluetooth communication. Future plans for this project involve expanding the app to be compatible with Android mobile systems, and enabling more advanced features such as volume estimation for potholes. We also plan to improve the adjustability and durability of the mounting system to support a wider range of road and weather conditions.

The Pothole Detection and Tracking System uses practical innovation to help solve an issue that affects many people on a daily basis. By streamlining the pothole detection process, our system offers realistic, affordable tools that give

cities and customers the data they need to take quick action in repairing potholes and improving roadway safety.

2.2 Features and Functionalities

The Pothole Detection and Tracking System is designed to achieve the following:

Bluetooth Connection: The system should support wireless communication with the user's smartphone using a Bluetooth Low Energy link. It must be capable of transmitting pothole detection data, including measured dimensions and other event details, and may optionally transmit associated image data for human confirmation if required by future versions of the application.

iOS/Android App: The data will be accessible through a dedicated mobile application installed on the user's phone. The app presents a list of potholes detected within the user's region, along with detailed information for each entry. It also includes an interactive map that displays pinned locations corresponding to detections received from the MSP430 and onboard GPS module. When a detection is made, the app parses the transmitted \$PD packet and uses the MCU-provided coordinates to accurately place the pothole marker on the map, independent of the phone's internal GPS.

Custom Mounting: The Sensors will use a custom mounting device that will be compatible with all vehicles. The device will have multiple degrees of freedom, allowing it to get the right height, angle, and distance from the vehicle. The device will also have the ability to use magnets, suction cups, and screws as options for mounting.

IR Laser Sensor: Part of the sensor will include a visible-light camera with a near-infrared bandpass filter that runs at 50 frames per second at 480p. The Laser part of the sensor will cover at least a 6 feet horizontally behind the vehicle, with the ideal size being a 10 feet.

AI Image Detection: A separate RGB visible-light camera will be used to capture higher-resolution images, which are stored in a buffer. These buffered images are processed through a custom-trained YOLOv5n model to confirm potential potholes detected by the IR laser system before any data is sent over Bluetooth.

Cloud Synchronization and Data Export: The mobile app will feature cloud synchronization, allowing all verified pothole detections to be uploaded to a database in real time. This ensures that the data can be accessed remotely, is backed up, and is able to be shared among users.

Weather Resilience: This system uses water-resistant housing units, an IR laser, and a filter-based imaging system, meaning our device is less affected by rain and poor lighting. This offers greater reliability than other systems.

2.2.1 Optical Schematic

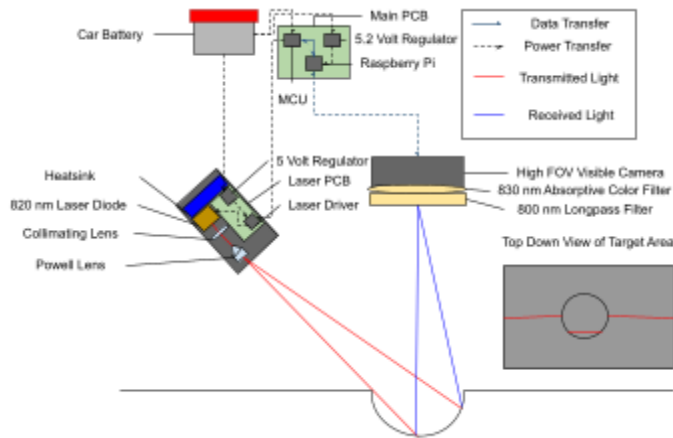


Figure 2.1: Optical Schematic

The Optical Schematic shown above better shows the relationship between the optical/laser systems and the relevant hardware systems of the project. The optical system uses an IR laser to project a line of light onto the road surface and a modified visible-light camera to locate any discrepancies on the road surface by monitoring the integrity of the line. The camera will need to operate at at least 41 FPS to capture a frame every foot while moving at 30 mph. If possible, operating at 60 FPS will allow us to capture a frame every foot while at a max speed of 40 mph.

2.2.2 Customer Input:

Cost efficiency: City planners and maintenance departments with limited budgets emphasized the need for a solution that delivers results without the need for a high investment. In response, we prioritized the use of low-cost components and open-source software, ensuring that our system cost was both functional and affordable.

Ease of Installation: Fleet operators and maintenance teams requested a solution that could easily be mounted and removed without the need for specialized tools or permanent vehicle modifications. This led to the development of a custom, non-invasive, mounting bracket compatible with a wide range of vehicle types.

Accurate & Real-time Measurements: Road maintenance teams emphasized the need for prompt and precise detection of potholes. This drove the integration of laser scan image processing, allowing potholes to be accurately detected, with minimal false positives.

2.2.3 Comparable Products/Projects:

AI Pothole Patrol Systems (Chicago, IL, San Jose, CA, Memphis, TN): Several cities have implemented AI and machine learning systems to analyze video footage from city vehicles to detect potholes with high accuracy. For instance, Memphis has equipped public works vehicles with AI-enabled cameras that process footage to identify issues like potholes and trash, achieving a 90% accuracy rate. These systems can automatically generate work tickets, streamlining maintenance operations. Some implementations also integrate LiDAR systems, allowing for precise measurements of pothole size and depth. These technologies demonstrate the viability of automated pothole detection and inspired our system's fusion of sensors and onboard analysis.

Pros: Enhances efficiency in infrastructure maintenance, reduces manual inspections, and enables proactive repairs.

Cons: Initial setup and training of AI models require significant resources, and dependence on cloud services may raise data privacy concerns.

Integrated OEM Solutions (Tesla, Mercedes, Ford): Some modern vehicles, like Teslas and newer models from Mercedes and Ford, have built-in systems to detect rough roads and potholes. Tesla uses cameras and sensors to find bad patches and adjust the suspension to reduce damage. Mercedes sends pothole alerts to nearby drivers using car-to-car messaging. Ford uses sensors to adjust the suspension in real time when a wheel hits a pothole. These tools are impressive, but they're locked into expensive vehicles and can't be added to other fleets. Our system, in contrast, is low-cost, works on any vehicle, and is made for city use.

Pros: Improves ride comfort and can help avoid damage to vehicles.

Cons: Only available in certain cars and can't be used for city-wide road scanning.

XenoTrack: XenoTrack is a LiDAR system that mounts to a car and creates a high-resolution 3D scan of the road. It's very accurate and great for spotting potholes, bumps, and cracks. It's mainly used in big infrastructure projects and has deep integration with mapping tools. However, Xenotrack is very expensive

and designed for large-scale road studies, rather than daily use by cities. Our system offers similar features using cheaper parts, making it better for local fleets.

Pros: Ideal for road surveys and getting detailed road condition maps.

Cons: Too expensive for small cities or wide deployment.

RoadBotics: RoadBotics uses a smartphone mounted on the dashboard to record the road and upload video for AI analysis. It can detect surface problems like cracks and potholes using that footage. The benefit is that it's cheap and uses everyday phones. The downside is that the system is not real-time. It has to be uploaded and analyzed later. Our system does all processing on the spot, which gives cities instant feedback.

Pros: Uses common devices without needing special hardware, keeping it affordable.

Cons: Can't detect in real-time and needs cloud-based processing.

Waze: Waze lets drivers report road problems like potholes through the app. Cities can partner with Waze to access this data to help plan repairs. While this is a free and useful system, it depends on people taking time to report issues, which leads to inconsistent results. Our design solves this by automating the detection process without risk of common human error.

Pros: Free to use, no hardware needed, and connects to city planning tools.

Cons: Data depends on user input, which can miss lots of issues.

Imagevision: ImageVision uses AI and forward-facing cameras to detect potholes from video footage. This works well in many cases, but can be sensitive to bad lighting, glare, and dirt on the lens. Our system uses infrared laser scanning, which isn't as sensitive to lighting conditions. That makes our detection more reliable, no matter the time of day or weather.

Pros: Uses standard cameras, good image analysis tools.

Cons: Needs clean, clear video and ideal lighting to work well.

2.3 Goals and Objectives

The primary aim of this project is to develop a reliable, cost-effective, and intelligent pothole detection system. This system must be capable of scanning

road surfaces from a vehicle, identifying and measuring potholes using laser-based sensing, and logging results with accurate location data. Additional emphasis is placed on real-time data access via Bluetooth mobile applications and robust environmental durability for deployment in various weather conditions. The goals have been divided into basic, advanced, and stretch categories, reflecting the increasing complexity and technical ambition of each.

Basic Goals

- Scan a 6 ft line from about 4 ft off the ground.
- Have a high SNR with a 200 mW laser diode.

The basic goal of this project is to develop a device that can analyze a road surface 6 feet horizontally from a mounting height of about 4 feet. This area ensures reasonable coverage behind a moving vehicle without requiring excessive mounting hardware. To ensure high-quality signal capture, the system must maintain a high signal-to-noise ratio using a 200 mW laser diode. This ensures the reflected laser signal can be clearly detected by the camera sensor despite the challenges posed by road surfaces and daylight conditions.

- Reliably detect >95% of potholes that are larger than 4 inches in diameter.
- Be able to differentiate between the laser light and noise.
- Block out visible light using a bandpass filter so that a visible-light camera can capture the NIR laser light.

To eliminate interference from ambient light, particularly sunlight and headlights, a narrow-band absorptive color bandpass filter will be used, allowing only the desired near-infrared laser wavelength to pass through. The system must reliably detect at least 95% of potholes larger than 4 inches in diameter.

- Keep the budget for the entire project under \$1,600.
- Draw power from the vehicle that the device is attached to. If this is not an option, create our own power supply that can provide enough power for the device to run for extended periods of time.

The design must be financially viable and remain under a \$1,600 development budget. This budget was selected to allow the system to remain inexpensive, while also taking into consideration that all funding is entirely from our own finances. The device will be powered by the vehicle it's attached to; however, if direct vehicle power is not an option, a custom battery pack or

onboard power solution will be developed to keep the system running for extended periods.

- Determine the GPS coordinates of detected potholes with >95% accuracy using the GPS on the user's phone.
- Make data easily accessible from a user's phone.
- Use AI object detection model to confirm pothole detections.

Pothole location data must be accurate, with GPS coordinates logged using our onboard GPS module and displayed within a user-friendly iOS app. Durability is essential, so the sensor housing must withstand varying weather conditions such as rain or heavy winds. A mounting device must also be designed that can universally be fitted onto any vehicle. This mounting device should also reduce road vibrations, ensuring smooth sensor readings. Finally, the system will implement a lightweight AI model, the YOLOv5n, to validate pothole detections, reducing false positives and improving reliability.

- Protect internal electronics from weather elements (water, dust, debris), meeting the IP65 standards.
- Prevent sensor misalignments caused by vibrations and bumps, using proper damping techniques.
- Properly secure the system to vehicle exterior without drilling or vehicle modification, using suction cups.

The mechanical system needs to ensure that all electronics are properly protected from environmental factors such as rain, dust, and road debris. This is accomplished by meeting IP65 enclosure specifications through a weatherproof housing design. Additionally, the mounting framework must minimize vibration frequency from the road to prevent sensor misalignment that would impact detection accuracy. The proper damping materials and structural rigidity are used to keep the correct orientation of the laser and cameras during driving. Finally, to ensure the system can be used universally, the entire system must mount securely to the exterior of the vehicle using suction cups. This facilitates easy installation and removal on a wide range of vehicles, while providing structural integrity in the real world driving environment.

Advanced Goals

- Scan an 8 ft line from about 4 ft off the ground.

- Reliably detect >95% of potholes that are larger than 2 inches in diameter.

Once basic goals are achieved, the system will be enhanced to increase detection area coverage to 8 feet, improving the rate at which road conditions are monitored. The detection capability will also be upgraded to identify smaller potholes, those at least 2 inches in diameter, with the same high reliability of >95%.

- Accurately estimate the area of potholes with less than 5% error.
- Be able to estimate the depth of potholes with less than 5% error.
- Detect if a pothole is filled with water and subsequently cancel depth estimations.

In addition to binary detection logic, the system will now attempt to calculate pothole size by estimating both area and depth, with a target accuracy of under 5% error for both. This will enable road maintenance teams to better prioritize repairs. However, if water is detected in the pothole, which could distort depth readings, the system will automatically cancel or flag depth data as invalid to maintain reporting accuracy.

- Design mounting system that avoids resonance within (5-30 Hz) range.
- Allow for adjustable mounting points for camera and laser alignment for accurate sensing.
- Ensure the system is adjustable in height and reach, allowing for universal use across a variety of vehicle types.
- Maintain internal operating temperature between 0°C and 45°C within the housing unit to ensure reliable performance for onboard electronics.

The mounting system must be designed so that it can avoid resonance within the 5–30 Hz range of vibrations that are normally encountered on the road, to prevent fatigue and sensor misalignment. To ensure ideal optical performance, the system must also include adjustable mounting points for the laser and camera modules for precise alignment and calibration. This flexibility ensures that both sensors are properly directed regardless of the mounting angle or vehicle configuration. The mount must also be height adjustable and extendable so it will accommodate a wide variety of vehicles without compromising structural rigidity. Internally, the enclosure must have a stable working temperature range of

0°C to 45°C to allow for the proper working of sensitive components like the IR camera, the RGB camera, and the laser emitter. Thermal management techniques, such as ventilation, are used to attain this level of temperature without using active cooling.

- Create an Android app in addition to the iOS app.
- Allow app users to review detections visually if necessary.
- Have pictures of potholes sent over Raspberry Pi Bluetooth.

For broader accessibility, an Android app will be developed alongside the iOS version. The app will allow users to manually delete false detections, or delete potholes that have been repaired, improving data quality. Additionally, the system will include functionality to transmit a visual image of the pothole to the user's phone via Bluetooth for further verification or logging.

Stretch Goals

- Be able to provide an estimate of the volume of the potholes with 90% accuracy.
- Be able to estimate the depth and volume of a pothole full of water with 80% accuracy.
- Develop a website in addition to the iOS and Android apps.

Stretch goals are ambitious and will be pursued as time and resources permit. Volume estimation represents a significant leap in data value for repair crews. The goal is to estimate pothole volume with 90% accuracy, providing direct insight into material requirements and urgency. Achieving reliable depth and volume estimation even for water-filled potholes will involve advanced image processing and depth estimation techniques. A target of 80% accuracy will be set for such challenging detections. Finally, a web-based dashboard will be developed to provide centralized access to pothole data. This will allow road maintenance authorities, researchers, or even community members to view and export condition reports from anywhere, expanding the system's reach beyond mobile platforms.

Objectives

- Determine the ideal method to create a line of laser light on the road, taking into account the cost and ability to cover enough of the road from a reasonable height.
- Expand a beam of laser light using a Powell lens.
- Take advantage of absorptive color bandpass filters to get a high FOV from the camera while minimizing blue shift.
- Determine the best method for identifying a disturbance in the laser light for identification purposes.
- Identify the ideal method of sending information from the device to a cloud server, even in somewhat remote areas.
- Design a shock-resistant and weatherproof housing for the device to protect the electronics from the elements and prevent any components from being dislodged.
- Create an iOS app that allows the user to access the pothole location and size data from their phone. This app should have the ability to show active potholes as pin markers on a map. This app uses the GPS coordinates given by the GPS module via bluetooth via the onboard BLE module.
- Develop a system that transmits the stored information when connected via Bluetooth.
- Develop a cloud server for storing pothole information data.
- Develop a system to draw power from the vehicle to which the device is attached. If this is not feasible, we will develop a power supply that can power the device for an extended period of time.
- Create a custom mount for the sensors. It should be able to mount onto any vehicle's rear, either temporarily (using suction cups or magnets) or permanently mounted to the vehicle as needed. The arm should have multiple degrees of freedom, allowing the sensors to be at the right height, angle, and distance from the vehicle.
- Use an object detection model to confirm detected potholes.

- Give app users with administrative access the ability to review, confirm or delete detections.
- Make use of a 200 mW laser diode to keep SNR high.

2.4 Specifications

Table 2.4.1: System Specifications

Parameters	Specifications	Priority
Operating Temperature	0°C to 48°C	High
Pothole Location Accuracy	Within 20 feet	High
Power Consumption	12 V	High
Cost	< \$1,000	Low
Length of Laser Line Coverage	6ft	High
Pothole Size Detection	Recognize potholes at least 4 inches in diameter	High
Laser System Weight	< 5 lbs	Medium

Table 2.4.2: Component Specifications

Component	Parameter	Specifications	Priority
IR Camera	Frame Rate	50 frames/second	High
IR Camera	FOV	100°	High
RGB Camera	Frame Rate	> 30 frames/second	High
RGB Camera	FOV	80°- 100°	High
RGB Camera	Resolution	1920 x 1080	Medium
Raspberry Pi	RAM	8 GB	High
Powell Lens	Fan Angle	≥90°	High

Table 2.4.3: Housing Unit Specifications

Parameters	Specification	Priority
Housing Material	Polycarbonate	High
Mounting Compatibility	GoPro Compatible attachment (Rear Mounted)	Medium
Camera Cutouts	2 front-facing cutouts for IR and RGB cameras	High
Weather Protection Rating	IP65 for protection against water and dust	High
Thermal Management	Vent slots to dissipate heat	Medium
Operating Temperature	0°C to 45°C	High

Table 2.4.4: Mounting System Specifications

Parameter	Specification	Priority
Material	6061-T6 Aluminum	High
Geometry (Cross Section)	Hollow rectangular tube	High
Length	Tescoping: 0.4m - 1m	Medium
Fastening Method	Clamp locking levers	High
Vehicle Attachment Method	5 Suction cups with vacuum locks	High
Vibration Tolerance	Designed to avoid resonance	High
Natural Frequency	>30 with damping	High
Environmental	Resistance to water and	Medium

Protection	mechanical shock	
------------	------------------	--

Prototype Illustration

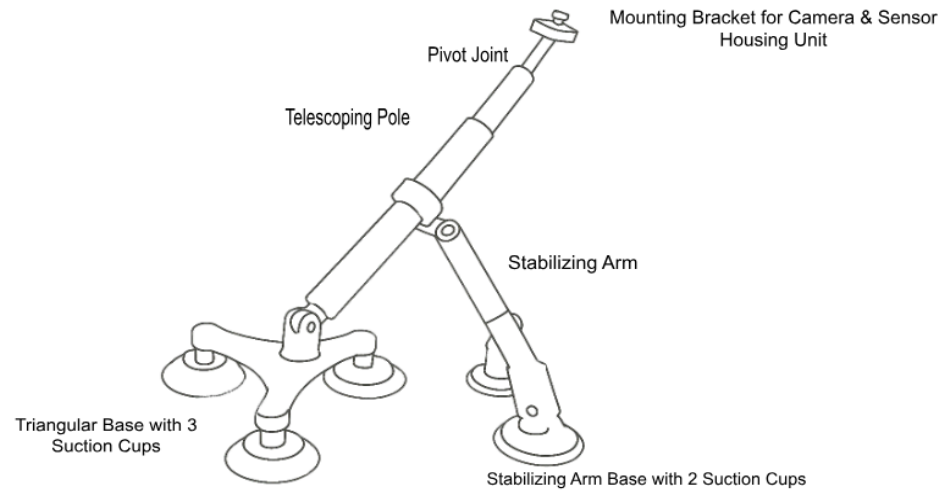


Figure 2.2: *Mounting Device Prototype*

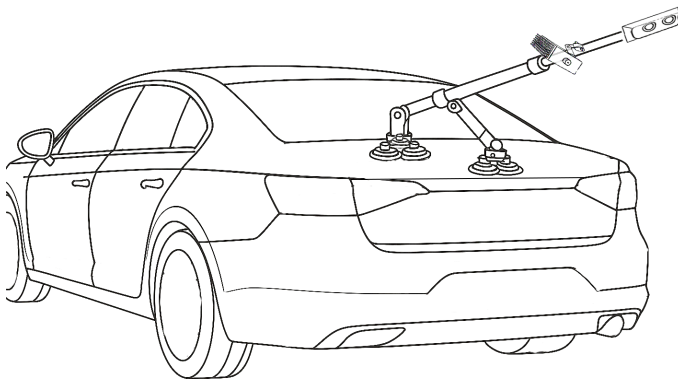


Figure 2.3: *Mounting Device and Housing Unit Secured on Vehicle*

The illustrated design for the mount system serves as a visual representation of the intended mounting assembly. The structure features a triangular base supported by three suction cups, each essential for providing a

stable anchor to the vehicle's surface. The three suction cups in the rear deliver primary load support, while the two positioned at the base of the stabilizing arm absorb torque and prevent tilt during sudden movements or turns. This secure mounting is critical to ensure accurate data collection while the vehicle is in motion. Extending from the base is a telescopic pole, designed to offer adjustable height and reach. The use of a modular, extendable pole and support arm ensures adaptability across different vehicle types and operating conditions.

The extendable pole allows for the mounted housing unit, containing the sensor and camera, to be positioned at an optimal vantage point. This ensures clear capturing of road imagery. The mounting bracket allows for quick mounting and includes a pivot joint, allowing for flexibility to determine the optimal and precise angle for surface scanning. Together, these components form a cohesive system that supports our project's goals: accurate pothole detection, reliable video capture and AI detection, and seamless integration with a mobile app. This prototype not only communicates mechanical function, but also reinforces the mount's role in ensuring accurate and high quality data acquisition while on the move.

2.5 Block Diagrams

2.5.1 Hardware Diagram

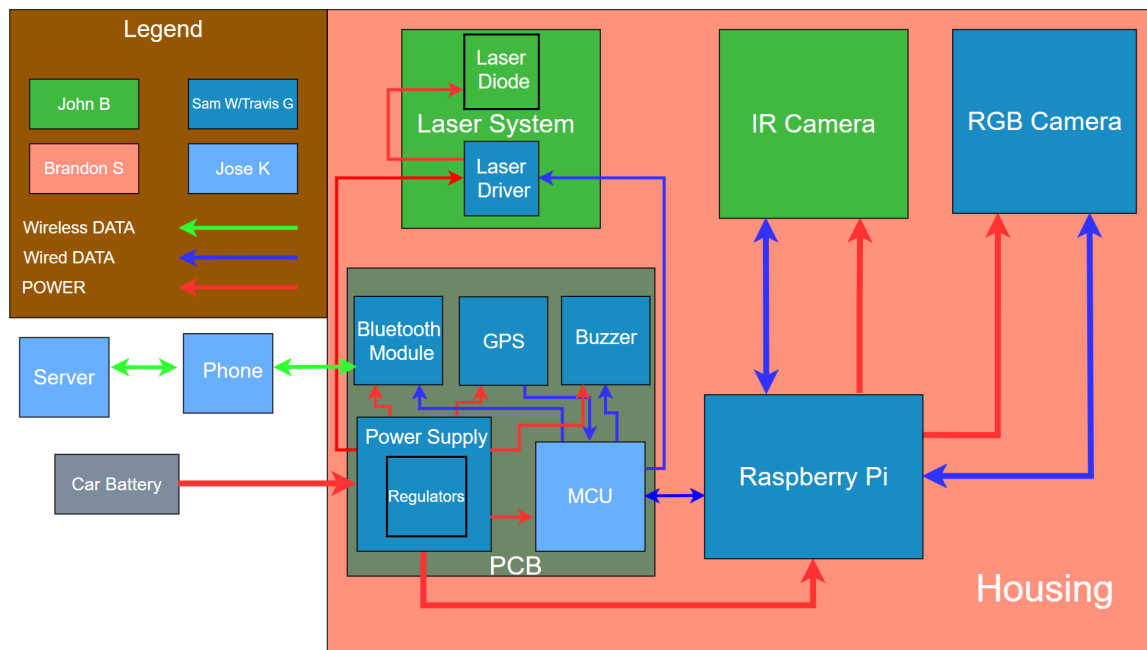


Figure 2.4: Hardware Block Diagram

The hardware block diagram outlines the system's hardware components and their interactions. Power from the car battery is regulated through a custom PCB that supplies the microcontroller, GPS module, laser driver, and Raspberry Pi. The laser system projects an IR laser line onto the road, and the laser driver, controlled by the MCU, powers the laser diode. An IR camera detects discrepancies in the IR sheet, after which the RGB camera captures an image of the suspected pothole, sending data to the Raspberry Pi for image processing. When a pothole is confirmed, the microcontroller reads the GPS coordinates from the onboard GPS module and sends the location and detection time over Bluetooth to the user's iOS application, where the data is displayed.

2.5.2 Software Flowchart

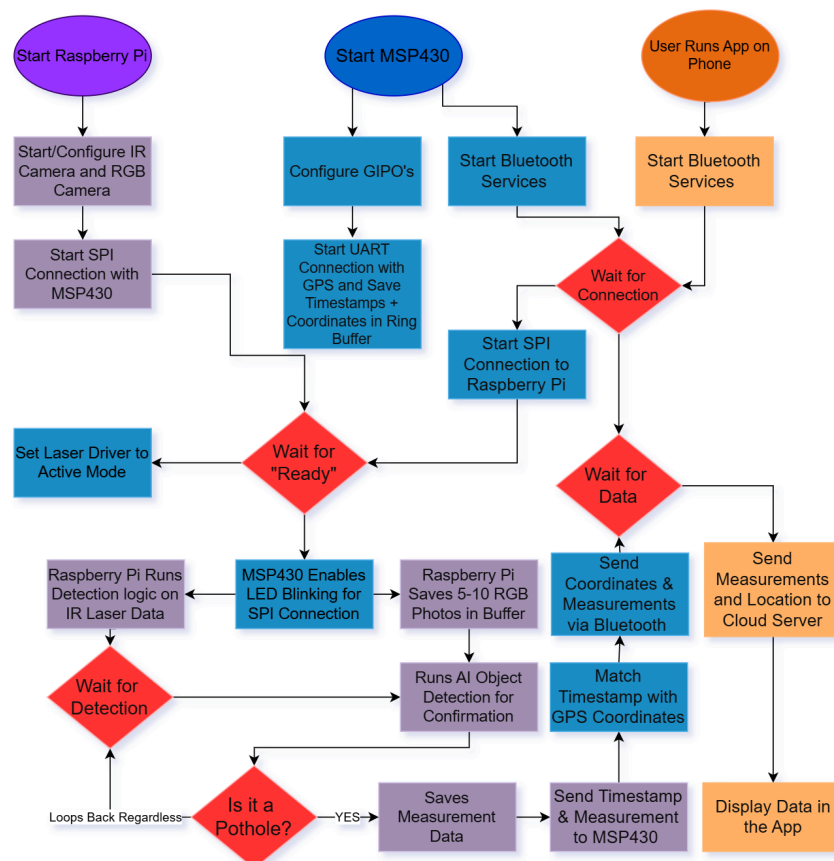


Figure 2.5: Software Flowchart

2.5.3 Flowchart Description

The software system begins with the MSP430 MCU and the Raspberry Pi starting up. The MSP430 initializes hardware-level controls, configures its

GPIOs, starts Bluetooth services, and begins reading GPS data through UART so it can continuously store timestamps and coordinates inside a ring buffer. At the same time, the Raspberry Pi initializes the infrared camera and RGB camera and begins setting up its SPI connection to the MSP430. Both devices wait for SPI communication to be acknowledged. Once the SPI link is established, the MSP430 enables LED blinking to indicate activity and the Raspberry Pi commands the laser driver to enter active mode.

With all systems ready, the Raspberry Pi starts running the infrared laser line deviation detection algorithm, while also saving the most recent 5–10 RGB frames in a rolling buffer. This ensures high-quality visual context is always available if a pothole is suspected. Because AI processing is only triggered after the IR laser system detects a possible pothole, the system avoids constant inference and reduces computational load, allowing the AI model to run efficiently when needed.

When the IR pipeline detects a potential pothole, it measures the deformation width and sends a signal to initiate AI-based confirmation. A separate thread runs the YOLOv5n model on the buffered RGB images to verify whether the detection is real. If the AI does not confirm a pothole, the Raspberry Pi clears the event and resumes monitoring. If confirmed, the Raspberry Pi retrieves the closest timestamp and matches it with the GPS coordinates stored in the MSP430's ring buffer, ensuring accurate location data.

Once the detection is finalized, the Raspberry Pi sends the measurement data and the matched timestamp to the MSP430 over SPI. The MSP430 then packages the measurements and GPS coordinates and transmits them over Bluetooth to the user's phone. The mobile app receives the data and displays the pothole's location, measurement results, and associated detection time.

2.5.4 Mobile Application Software Flowchart

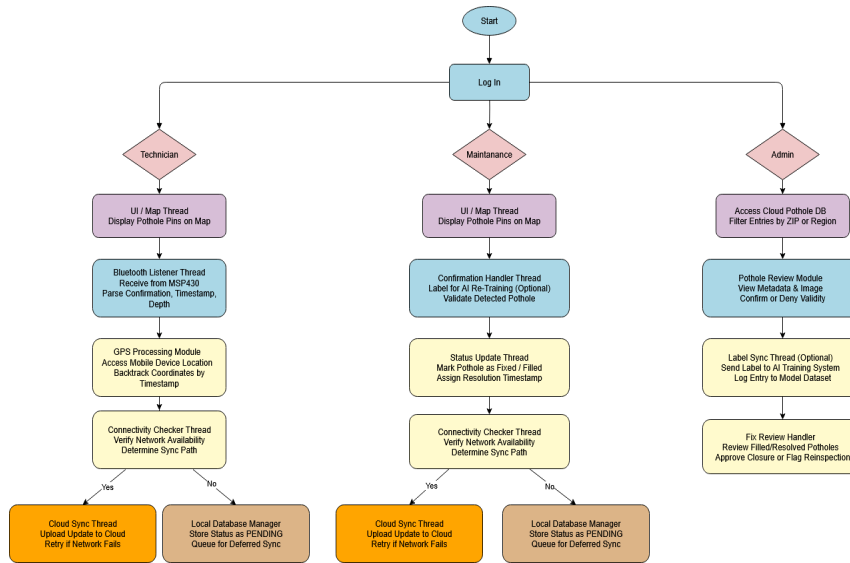


Figure 2.6: Mobile Application Flowchart

2.5.5 Application Flowchart Description

The mobile application system begins operation once the user launches the app and its BLE services are initialized. The application immediately starts scanning for nearby BM71-based MSP430 devices using the standard BLE protocol stack. Once the correct device is detected, the system establishes a secure connection and transitions into its live-data monitoring mode.

After a successful connection, the Bluetooth Listener Thread activates and continuously watches the incoming stream for \$PD packets sent from the MSP430. These packets contain pothole attributes generated by the Pi-based AI model, such as pothole depth, area, severity, and the detection timestamp. The Listener Thread parses each payload and forwards the structured data to the GPS Association Module.

The GPS Association Module is responsible for assigning accurate coordinates to each pothole event. Because the MSP430 relies on the dedicated vehicle-mounted GPS module rather than the phone's internal GPS, the incoming packet provides integer-formatted latitude and longitude values. When valid GPS data is received, the module converts these values into standard coordinate format and performs a brief de-duplication check to prevent stacking multiple markers in nearly the same location during rapid consecutive detections.

Next, the Cloud Uplink Handler evaluates mobile data or Wi-Fi availability and determines the appropriate storage path. If the device is online, the Cloud Sync Thread packages the full detection record, including timestamp, coordinates, depth, area, severity, and user ID, into a payload for insertion into the MongoDB cloud database. If connectivity is unavailable or the request fails, the entry is handed off to the Local Storage Manager, where it is cached and flagged as PENDING. This queue is monitored continuously so that once connectivity returns, all unsent records are uploaded automatically without requiring user input.

The application maintains a multithreaded structure to allow Bluetooth handling, GPS conversion, storage operations, and UI updates to run concurrently. The UI / Map Thread renders an Apple-Maps-based interface, dynamically placing new pothole pins as detections arrive. These pins display metadata and allow technicians or drivers to review severity, confirm events through in-app controls, and visualize historical detections along recently driven routes.

In parallel, the Administrator Thread offers a privileged interface for designated users to review pothole submissions across a wider geographical area. Administrators may filter entries by region and other criteria, such as severity or status, to quickly identify clusters of unresolved hazards. Through the Pothole Review Module, they can inspect the associated GPS coordinates, severity values, and timestamps, and can either confirm or deny detections, supporting quality control and improving the reliability of system evaluations.

Confirmed entries are stored in the cloud database along with their review status, allowing the backend to maintain a clear distinction between verified and unverified detections. While any machine-learning improvements for the Pi-based model are handled offline, the administrator-confirmed records serve as reliable ground-truth data that can be exported later if model adjustments are required. Administrators also have access to the Fix Review Handler, which allows them to audit potholes marked as “fixed” or “resolved” by maintenance technicians. This final review gate ensures that only fully verified repairs are removed from the active map and cloud database. This modular and thread-based design allows the application to serve multiple user types; drivers, maintenance personnel, and administrators while maintaining data integrity and responsiveness even during limited connectivity or offline conditions.

2.5.6 Mobile Application Design Diagram

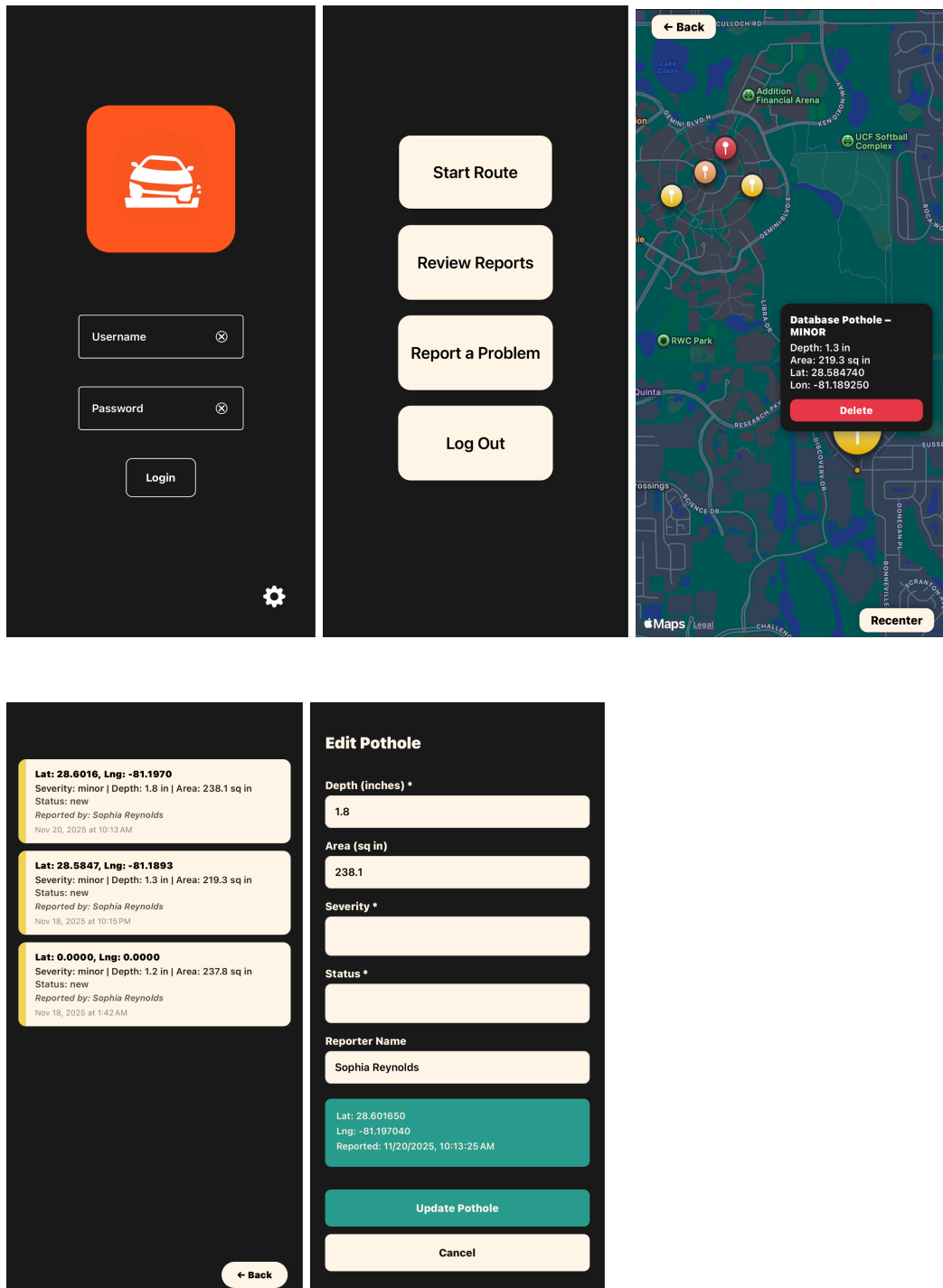


Figure 2.7: Mobile application design

The Mobile Application Design Diagram illustrates the conceptual layout of the interfaces that make up the mobile application. The Login Screen provides a simple entry point for authenticated access, allowing users to sign in under their designated role driver, maintenance technician, or administrator. After authentication, the Home or Landing Screen presents a streamlined overview of key functions, including access to the live map, list of potholes, and other role-specific actions. This layout ensures that users can immediately navigate to essential tools without unnecessary complexity.

The Map View uses an Apple-Maps-based visualization to display pothole locations using clear, color-coded markers. Confirmed potholes are shown with red pins, newly detected or unreviewed entries appear with yellow markers, and the user's current position is represented with a distinct blue indicator. This real-time layout allows users and field technicians to quickly assess hazard locations, track recent detections, and plan driving or inspection routes efficiently.

The List of Potholes screen provides a structured, scrollable view of all stored pothole entries. Each item includes metadata such as depth, area, severity, and GPS coordinates, along with visual cues identifying whether the detection has been confirmed, denied, or marked as fixed. Selecting any entry opens the Update Pothole Screen, where authorized users can modify the pothole's status, adjust severity or depth values, or confirm repairs made in the field. This update interface supports the administrative workflow by ensuring that changes are accurately logged and synchronized with the cloud database.

Overall, the interface prioritizes clarity, ease of navigation, and operational efficiency, allowing drivers, maintenance personnel, and administrators to make quick, informed decisions throughout the detection and verification process.

2.6 House of Quality

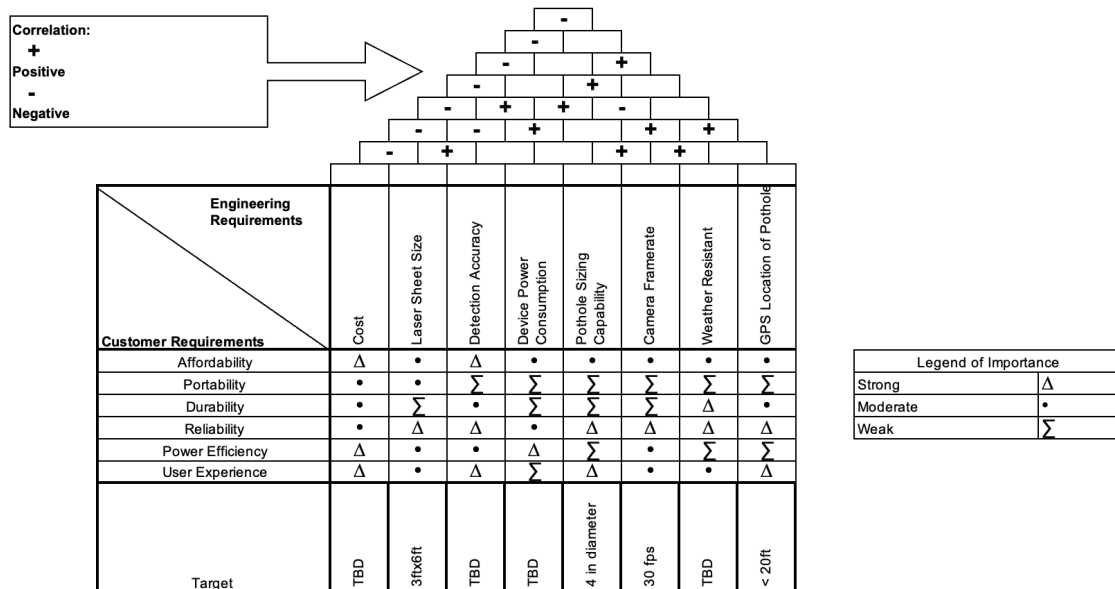


Figure 2.8: House of Quality

3. Research

3.1 Electrical Hardware Technology Comparisons

3.1.1 Microcontroller vs FPGA vs SBC

For this device, the design team is tasked with choosing between platforms that will be responsible for managing the other hardware components. These three technologies are the microcontroller (MCU), field programmable gate array (FPGA), and the single board computer (SBC). Each has their own set of benefits as well as their own set of setbacks.

There are many benefits to using an FPGA, this is because FPGAs are programmed using a hardware descriptive language, and they are programmed to perform one specific task. They implement true parallelism, which can run tasks simultaneously. They have a better processing power than the MCU and have a higher scalability.

One of the major setbacks of using an FPGA instead of a microcontroller is the complexity of programming the FPGA, there is a steep learning curve for

using hardware descriptive languages, which would add more time needed to program the device.

The benefits of using a microcontroller is to manage low-level hardware tasks that do not require a lot of processing power. The responsibility of the microcontroller is to manage the laser driver, establish a UART connection with the Raspberry Pi, toggling GPIO status LEDs, monitoring the TEC, and sending GPS information to the Bluetooth module. The microcontroller is there to ensure real-time responsiveness at the hardware level.

The microcontroller takes on a major role in the design because it is not only required to be a part of the design, but also because it allows the Raspberry Pi to handle the image processing while it makes sure certain peripherals have an established connection/power being supplied to them.

Microcontrollers usually have low-power characteristics, are reliable, and are good at interrupt handling. This makes them an ideal choice for what the device is trying to accomplish. Having a microcontroller handle the control logic and enabling consistent coordination in the system is essential.

Table 3.1.1: MCU vs FPGA

Criteria	MCU	FPGA	SBC
UART Communication	Excellent	Poor	Good
Programming Difficulty	Excellent	Fair	Fair
Power Efficiency	Excellent	Poor	Poor
Real-Time Control	Good	Excellent	Poor
Cost	Excellent	Poor	Fair
Processing Power	Good	Excellent	Excellent
Overall	Excellent	Fair	Good

Given this information and the priorities of the project, the design team decided to go with utilizing an MCU to do the logistical work of the design. The group also decided to use an SBC to handle the AI image processing.

Microcontroller (MCU)

MSP430FR2355

The design team's first option when it came to choosing an MCU was the MSP430FR2355, commonly known as MSP430. This MCU is a low-power, 16-bit microcontroller developed by Texas Instruments. This MCU features an MSP430 CPU core operating at up to 24 MHz, as well as 32 kB of non-volatile FRAM and 4 kB of SRAM for quick memory access. The device supports multiple communication interfaces such as UART, I2C, and SPI, offering flexibility in how data is sent and received. This device features 44 I/O, a 12-bit ADC, timers, and advanced power-saving modes. This device also offers an on-chip 32 kHz RC oscillator as well as an external 32 kHz crystal oscillator. [28]

ESP32 WROOM 32

The ESP32 WROOM 32 is another option that the design team considered when choosing an MCU for the device. The ESP32 WROOM 32 features both Bluetooth and Wi-Fi capabilities; however, due to design constraints, the design team is unable to utilize these features. The ESP WROOM 32 also features a D0WD-V3 chip, which features a dual-core 32 bit Xtensa LX6 processor with clock speeds up to 240 MHz. This device integrates 4 MB of external SPI flash memory and includes multiple serial interfaces, similar to the MSP430, such as UART, SPI, and I2C. The minimum power supply voltage of this device is 3V, with a maximum power supply voltage of 3.6V. The minimum typical current draw of this device is 2.8 mA, with a maximum typical current draw of 379 mA at its peak.

MSP430FR6989

The last MCU option the design team considered for our design was the MSP430FR6989 made by Texas Instruments. This MSP430 model is a 16-bit low-power microcontroller that features a CPU core that runs up to 16 MHz, with 128 KB of non-volatile FRAM and 2 KB of SRAM, allowing fast and low-energy memory access. This device includes a 12-bit ADC with up to 16 channels and multiple serial interfaces, including multiple UART, SPI, and I2C ports. This device supports up to 83 GPIOs, and includes an integrated LCD driver capable of handling up to 320 segments, in case the project design calls for an LCD display. The features of this MCU make it a flexible and effective option.

Additionally, since the design team has experience with this device, it is a very viable option for the project.

Table 3.1.2: MCU Comparison

Criteria	MSP430FR2355	ESP32 WROOM	MSP430FR6989
Core Architecture	16-bit RISC	32-bit Xtensa dual-core	16-bit RISC
Power Supply Voltage	1.8V-3.6V	3.0V-3.6V	1.8V-3.6V
GPIO	44 pins	26 pins	83 pins
FRAM	32 KB	4 MB Flash	128 KB
SRAM	4 KB	520 KB (internal RAM)	2 KB
Current Draw	~100-350 uA/MHz	~20-130 mA	~100-300 uA/MHz
Operating Temperature	-40 - +85 °C	-40 - +85 °C	-40 - +85 °C
Overall	Excellent	Good	Good

SBC Comparison

Raspberry Pi 5 vs BeagleBone AI-64

An integral part of this system is image processing using trained AI models that can confirm whether a discrepancy on the road is in fact a pothole. This makes choosing a device for handling image processing very important. There are many options when it comes to choosing such a device, but two devices ultimately stood out: the Raspberry Pi 5 and the BeagleBone AI-64.

Raspberry Pi 5

The Raspberry Pi 5 is powered by a 64-bit quad-core Arm Cortex-A76 processor operating at 2.4 GHz, offering a 2-3x increase in CPU performance over the previous Raspberry Pi 4. This performance gain is complimented by an enhanced 800 MHz VideoCore VII GPU, dual 4Kp60 HDMI outputs, and an updated Image Signal Processor (ISP) that provides advanced support for

high-resolution cameras. These upgrades significantly improve the system's capabilities for desktop workloads and extend its applicability to industrial and embedded use cases.

A key advancement in the Raspberry Pi 5 architecture is the introduction of the in-house designed RP1 "southbridge" chip, which delivers a substantial increase in I/O performance and functionality.

Aggregate USB bandwidth has more than doubled, resulting in faster data transfers to external storage and high-speed peripherals. The previous two-lane 1 Gbps MIPI camera and display interfaces have been replaced with dual four-lane 1.5 Gbps MIPI transceivers, tripling the total bandwidth and supporting any combination of up to two cameras or displays.

BeagleBone AI-64

The BeagleBone AI-64 is built around Texas Instruments' TDA4VM system-on-chip (SoC) from the J721E family, part of the K3 Multicore SoC architecture designed for automotive, industrial, and robotics applications. At its core, the SoC features a dual-core 64-bit Arm Cortex-A72 processor that can run up to 2.0 GHz, delivering up to 24,000 DMIPS of performance.

The BeagleBone AI-64 also features a variety of specialized processing units, like the C71x DSP with 80 GFLOPS, multiple C66x DSP cores, and a Matrix Multiplication Accelerator (MMA) capable of delivering up to 8 TOPS (Tera Operations Per Second) for advanced AI interfacing, machine vision, and signal processing with a low-power envelope, making this platform adaptable to edge-computing applications. The SoC integrates a set of hardware accelerators and peripherals to support industrial and automotive use cases. A Vision Processing Accelerator (VPAC) and a Depth and Motion Processing Accelerator (DMPAC) enable real-time processing of high-resolution camera data for applications such as advanced driver-assistance systems (ADAS) and robotics.

The integrated GPU, multi-standard HD video encoder/decoder, and flexible display interfaces (eDP, DSI, DPI) support complex multimedia workflows. The SoC also provides dual four-lane MIPI CSI interfaces for camera inputs, which can handle simultaneous multi-camera operations. The TDA4VM also emphasizes functional safety and security. Safety features include dual lockstep Cortex-R5F microcontrollers, error-correcting code (ECC) protection on critical memory, watchdog timers, and temperature monitoring sensors. An effective security architecture supports secure boot, trusted execution environments (TEE), and hardware cryptographic acceleration (AES, SHA, RSA). These features help ensure system integrity and reliability, even in harsh environments.

When it comes to connectivity, the platform includes multiple high-speed interfaces such as PCIe Gen3, USB 3.0, and Gigabit Ethernet, along with I/Os like CAN-FD, SPI, I2C, UART, and GPIO. Simplified power management with on-die LDOs and adaptive voltage scaling (AVS) helps reduce the complexity and cost of the system. All of these features make the BeagleBone AI-64 a powerful platform for real-time vision analytics, deep learning applications, and embedded control.

Table 3.1.3: Development Board Comparison

Criteria	Raspberry Pi 5	BeagleBone AI-64
Main CPU	Quad-core 64-bit ARM Cortex-A76	Dual Cortex-A72
Frequency	2.4 GHz	2.0 GHz
AI/Vision Accelerators	VideoCore VII GPU at 800 MHz	C71x DSP, C66x DSPs, and MMa up to 8 TOPS
Interfaces	PCIe 2.0 x1, USB 3.0 x2, GbE, microSD	PCIe Gen3 x4, USB 3.0, GbE, dual CAN, PRU ICSS
Memory and Storage	LPDDR4X up to 16 GB	4 GB DDR4 + 16 GB eMMC
Power Consumption	Up to 6-7 W peak	Not Specified
Cost	~\$80 (8 GB) - \$132 (16 GB)	~\$228
Overall	Excellent	Good

Ultimately, the design team chose the Raspberry Pi 5 as the SBC for this project. This is due primarily to its cost, as well as to its comparable specs with the BeagleBone AI-64. The Raspberry Pi is significantly cheaper which, in this project, is a high priority.

3.1.2 Device Power Source

At first, our group considered using a detachable and chargeable battery pack that would be part of our device, which would increase our device's portability. This would also create a safer testing environment, since not all users

are comfortable wiring a device to a fuse tap in their vehicle. Power stability might also improve, as the input power would likely be less affected by vehicle voltage noise.

There are also some cons to powering the device via battery. Having to remember to remove and charge it approximately once a day is inconvenient, and is another thing for our user to do each day. Runtime would be much more limited due to the fact that the battery would die quickly, relative to wiring to a fuse tap, depending on the size. For the purposes of this device, and how it would be used on a daily basis, the pros of using a battery were not priorities for the project.

Another option we had was utilizing the battery from the car, creating a DC connection via fuse tap, and using DC-DC buck converters to step down the 12V to the appropriate levels for the device's peripherals.

One of the benefits to using the car's battery as a power source is the runtime relative to using a removable battery, which would require frequent recharging. The power supplied by the car's battery should be more than enough to power our device and its peripherals. The overall system requires less space due to not having a battery pack attached/inside the system.

There are also some cons with using a direct connection to the car's battery as a power source. The device would be less portable and not every user is comfortable wiring a device to a fuse tap in their vehicle. The power is also less stable and predictable compared to that of a battery, meaning the PCB is likely to be more complicated because the device is not using a steady source of power. Both the car's battery and a separate battery are viable options for the device, each posing many pros and cons. For the purposes of what this device is hoping to achieve, and the ideal way that this device is to be used, the design group chose to use a DC-DC connection, utilizing the vehicle's battery.

Table 3.1.4: Energy Source

Criteria	Car Battery	Battery Pack
Runtime	Excellent	Fair
Cost	Excellent	Fair
Size and Weight	Excellent	Poor
Power Quality	Fair	Excellent

Portability	Poor	Good
PCB Design Complexity	Fair	Good
Overall	Good	Fair

Ultimately, our group decided to utilize the car's battery as the power source for our device if possible. We chose this because of the convenience that the user would experience versus using a battery pack, which would die much quicker.

Switching Regulators vs Linear Regulators

A crucial piece of technology that our design will utilize is regulators. The two types of voltage regulation technology that the design team narrowed their choices down to are switching and linear regulators. Both voltage regulating technologies provide a stable voltage output, but take different approaches in doing so.

Switching regulators use an electronic switch that rapidly toggles on and off, transferring energy rapidly through inductors and capacitors in order to control the output voltage. Since switching regulators are constantly switching between on and off, a minimal amount of energy is dissipated as heat, this means the efficiency is quite good relative to linear regulators.

On the other hand, switching regulators require inductors and capacitors, which makes the total voltage regulation design more complex. This also makes the design physically larger, leading to less space on the PCB.

Switching Voltage Regulators

LM2576S-5.0

The first 5V voltage regulator the design team considered was the LM2576-5.0, a buck voltage regulator made by Texas Instruments, which was recommended by an advisor. This is a common voltage regulator that has been widely used for many years due to its reliability. This voltage regulator operates at a relatively low switching frequency of about 52kHz, it is known for being robust and fairly easy to implement.

This voltage regulator's efficiency and physical footprint are less favorable than the other voltage regulators that were considered for this device. Its low switching frequency requires larger external inductors and capacitors, which

results in a bulkier design. This voltage regulator is known to generate more heat than other, new-generation voltage regulators, which poses a threat to surrounding components that are sensitive to heat.

Since this voltage regulator is known to be dependable, even at the cost of efficiency, the design team will still consider it when it comes time for the final design. And since the design will not be in a small, enclosed space, the heat that is generated from this voltage regulator should not pose such a vital threat to the system or its components.

LM2679-ADJ

The second voltage regulator the design team considered was the LM2679. This is also a buck voltage regulator created by Texas Instruments, but the key difference between this voltage regulator and the LMS2576-5.0 is that this voltage regulator allows the design team to adjust the output voltage according to whatever voltage is required by the peripherals or sensors. This is a significantly improved design when it comes to flexibility of output voltage relative to the previous voltage regulator.

For the purposes of this design, the fact that this regulator can accept inputs up to 40V is a huge pro due to the voltage range of a typical car battery. This means this voltage regulator can handle both low voltage and higher voltage events from a car's battery.

This voltage regulator is rated up to 7A, which is enough amperage to supply our peripherals with sufficient current. This voltage regulator also operates at 260 kHz, which is a much more stable frequency to operate at.

The LM2679 uses advanced high-speed circuitry that allows it to step down an input voltage as high as 40V to provide a stable 3.3V output, 5V output, and 5.2V output while maintaining a switching frequency of 260 kHz.

This regulator operates using a fixed-frequency PWM (Pulse Width Modulation) control architecture. At higher loads, the device maintains a constant 260 kHz switching frequency, which ensures stable operation, predictable ripple performance, and straightforward filter design.

Under lighter loads, the LM2679 enters a reduced-conduction mode that lowers switching losses and improves efficiency, although it does not use the same automatic PWM/PFM hybrid scheme found in most modern regulators.

Packaged in a thermally enhanced TO-220 or TO-263 package, the LM2679 provides good heat dissipation and low electrical parasitics. The short, wide leads of these packages help reduce series inductance and resistance, improving switch-node behavior, reducing voltage overshoot, and maintaining efficiency.

MP2338

The last 5V regulator the design team looked at is the MP2338 from Monolithic Power Systems. This voltage regulator is a 3A synchronous buck regulator with a high switching frequency of 1.4 MHz which improves transient response or a faster reaction to changes in load, and enables the use of small inductors and capacitors, which would make it better for keeping the size of the PCB down.

Table 3.1.5: Voltage Regulator Comparison

Criteria	LM2576S-5.0	LM2679-ADJ	MP2338
Max Output Current	3A	7A	3A
Input Voltage Range	7V-40V	8V-40V	4.5V-24V
Output Voltage	Fixed (5.0V version)	Adjustable (1.23V-Vin)	Adjustable (.925V-Vin)
Switching Frequency	52 kHz	260 kHz	1.4 MHz
Efficiency (Peak)	65%-75%	90%-95%	90%-95%
Thermal Performance	Heatsink required at higher loads	Minimal heat	Good, needs airflow at 3A
Cost	\$3.46	\$7.09	\$1.55
Overall	Poor	Excellent	Fair

This voltage regulator can handle an input voltage range of 8V to 40V, which makes it viable for this design due to the volatility of the supply voltage received from the car's battery. One feature of this voltage regulator is internal soft-start, which gradually ramps up output voltage when enabled. This is

important because it prevents large inrush currents at startup. It also reduces stress on input capacitors which can help prevent brownouts.

As you can see from the table above, the LM2679-ADJ beats the other two regulators by many factors. Due to its flexibility and adjustability of its output voltage, the design team has chosen the LM2679-ADJ as the regulator for this device.

3.1.3 Pothole Data Access

The initial design of the device utilized the Bluetooth module attached to the Raspberry Pi. Due to design constraints, the design team had to consider other options of data transfer/storage.

The Bluetooth module would be soldered to the PCB, and have a corresponding LED to indicate that it is receiving power. The Bluetooth module will receive data from the MCU, where it will then be in contact with the user's phone in order to mark a GPS location via an app.

One option considered was local data storage, such as storing pothole data including GPS location on an SD card or another physical memory. This leads to extra time and effort required to retrieve such data. The user would have to upload the data onto a server or app at the end of each day, which would be inefficient.

Some pros of using an SD card or other physical memories would be the cost, power, and simpler implementation of the memory storage itself. It would also eliminate the need for constant wireless connectivity if the user was in an area with poor reception, or if the user's phone was out of power. It would be a viable option as a backup in case the Bluetooth option had failed.

Another option that was explored is the integration of a cellular communication module. This would allow the device to transmit pothole data directly to a remote server via a cellular network, which would eliminate the need for phone pairing or manual uploads via local data storage. While this option would add convenience and enable real-time data transfer from pretty much anywhere with cellular service, it would also bring with it challenges such as higher power consumption, increased cost, and the need for a SIM card, which would require a monthly subscription for a data plan. When the design team weighed these pros and cons, the cellular module was a strong candidate for wireless communication for the pothole detection system, but in the end, the design team chose to use the Bluetooth module instead.

Table 3.1.6: Pothole Data Access Comparison

Criteria	Bluetooth Communication	Local Data Storage	Cellular Communication
Range	Excellent	Poor	Excellent
Bandwidth	Good	Excellent	Good
Real-Time	Excellent	Poor	Excellent
Power Use	Excellent	Excellent	Poor
User-Friendly	Excellent	Poor	Good

As mentioned above, using the pre-installed Bluetooth module on the Raspberry Pi was initially heavily considered, but it was not possible due to design constraints. After further thought and instruction, the design team felt that using a Bluetooth module separate from the Raspberry Pi was the best option.

Bluetooth Module

Microchip Technology BM71

Upon initial research into finding a Bluetooth module for the project, the design team came across the Microchip Technology BM71. This is a compact BLE (Bluetooth low energy) module that integrates a Bluetooth 5.0 baseband controller, RF transceiver, and onboard antenna.

The BM71 typically draws around 10 mA during transmission at 0 dBm at room temperature, with current consumption rising to about 13 mA under higher temperatures. Its receive current closely matches its transmit current, maintaining efficiency across wireless operations. The module operates from a 1.8V to 3.6V supply, which is standard for ultra-low-power BLE modules. [41]

RayTac MDBT50Q-1MV2

The second Bluetooth module that the design team found was RayTac's MDBT50Q. The 1MV2 indicates that the module uses an antenna located on the chip rather than the PCB, which is the desired model for this design. The MDBT50Q is a Bluetooth 5.2 stack module which features Bluetooth low energy or BLE. This design is based on the Nordic nRF52840 SoC solution, which incorporates GPIO, SPI, UART, I2C, I2S, PMD, PWM, ADC, NFC, and USB interfaces for connecting peripherals.

This device features a 2.4GHz, Bluetooth 5, IEEE 802.15.4 transceiver. It also supports data rates of 125 kbps, 500 kbps, 1 Mbps, and 2 Mbps, with 1 and 2 Mbps using the proprietary 2.4 GHz frequency. This device also features a flash memory of 1 MB, and a RAM of 256 KB, along with an ARM Cortex – M4 processor with FPU, 64 MHz.

The power management system of this device is also flexible, with a supply voltage range from 1.7V to 5.5V, on-chip DC-DC regulators and LDO regulators, as well as the ability to power other peripheral devices with a voltage of 1.8V to 3.3V, making this device highly versatile. [40]

u-blox NINA-B306-01B

The final Bluetooth module that the design team considered for the device is the u-blox NINA-B306-01B. This Bluetooth module has comparable specifications to the Bluetooth modules above, more specifically to the RayTac Bluetooth module, since both are based on the Nordic nRF52840 solution.

Table 3.1.7: Bluetooth Module Comparison

Criteria	Microchip Technology BM71	RayTac MDBT50Q-1MV2	u-blox NINA-B306-01B
Clock Speed	2.4 GHz	2.4 GHz	2.4 GHz
Flash/RAM	N/A	1 MB/ 256 KB	1 MB/ 256 KB
Input Voltage Range	1.8-3.6V	1.7-5.5V	1.7-3.6V
Current Draw (TX/RX Mode Peak Current)	13 mA/ 13 mA	2.7-32.7 mA/ 3.7-11.1 mA	4.9-14.1 mA/ 4.8 mA
Cost	\$9.24	\$6.15	\$8.13
Temperature Range	-40 - +85 °C	-40 - +85 °C	-40 - +85 °C

This Bluetooth module also has a clock speed of 2.4 GHz, a flash memory of 1 MB, and a RAM of 256 KB. The supply voltage range for this device ranges from 1.7 to 3.6 V, with a current draw in TX mode from 4.9 to 14.1 mA, depending

on the power output used, which ranges from 0 dBm to 8 dBm. The current draw for RX mode is about 4.8 mA.

The features of this Bluetooth module make it incredibly attractive for the design of the system. However, this device's lack of flexibility when compared to the RayTac Bluetooth module makes it difficult to choose.

GPS Module

Choosing a reliable GPS module was crucial in the design of this system. There were many options, here are the pros and cons of each option.

Adafruit Ultimate GPS Breakout

The Adafruit Ultimate GPS Breakout was the first GPS module looked at by the design team, due to a recommendation by a project advisor. It is a GPS module built around the MTK3339 GNSS engine, a sensitive receiver with an excellent reputation.

This breakout board includes an antenna, integrated voltage regulation, level shifting, and a UART interface. It can support update rates of 10 Hz, which works well for this project because it is on a moving vehicle, requiring frequent position updates.

Adafruit PA1616D GPS Module

The Adafruit PA1616D GPS module was the second GPS module looked at by the design team. It is a compact and low-cost alternative to the Adafruit Ultimate GPS breakout. This board also includes an onboard antenna, voltage regulation, and level shifting, which also makes for easy integration.

SparkFun ZOE-M8Q (u-blox) GPS Breakout

The SparkFun ZOE-M8Q GPS breakout was the final GPS module option looked at by the design team. This is definitely the most expensive, yet the most robust option.

Table 3.1.8: GPS Module Comparison

Criteria	Adafruit Ultimate GPS Breakout	Adafruit PA1616D GPS Module	SparkFun ZOE-M8Q GPS Breakout
-----------------	---------------------------------------	------------------------------------	--------------------------------------

Update Rate	10 Hz	10 Hz	10 Hz
Onboard Antenna	Yes	Yes	Yes
Input Voltage	3.3 V	3.3 V	3.3 V
Price	\$30	\$25	\$50
Overall	Excellent	Poor	Fair

The design team ultimately chose the Adafruit Ultimate GPS Breakout, this is due to performance metrics, as well as ease of integration into the system.

3.1.4 Buzzer Selection

For the audible alert system on our main pcb, the design team chose to go with a buzzer. This buzzer sounds when there is an SPI connection made, one of the final steps of the system being ready to perform. The design team looked at three different buzzers, here are the comparisons.

PUI AudioSMT-0827-TW-3V-R

The SMT-0827-TW-3V-R made by PUI Audio is an SMD magnetic buzzer with a footprint of about 8.5 mm x 8.5 mm, and a height of about 5.5 mm. It delivers a sound pressure level of about 85 dB, with an operating temperature of -40 C to 85 C.

SameSky CMI-9705-0380-SMT-TR

The CMI-9705-0380-SMT-TR made by SameSky is a surface mount magnetic buzzer with a footprint of 9.7 mm x 9.7 mm and a height of about 5 mm. The sound pressure level of this buzzer is about 80 dB.

Mallory Sonalert ASI09N27MFD-03TRQ

The ASI09N27MFD-03TRQ buzzer made by Mallory Sonalert is a magnetic, internally-driven buzzer with a footprint of about 9.6 mm x 9.6 mm and a height of about 5.3 mm. It has an input voltage rating of about 3 V, and produces about 80 dB.

Table 3.1.9: Buzzer Comparison

Criteria	Mallory Sonalert	PUI Audio	SameSky
----------	------------------	-----------	---------

Rated Input Voltage	3 V	3 V	3 V
Current Draw	30 mA	25 mA	30 mA
Sound Pressure Level	80 dB	78 dB	80 dB
Price	\$3.50	\$2.25	\$2.30
Overall	Excellent	Poor	Fair

The design team chose to select the Mallory Sonalert buzzer for the final design, both for simplicity of integration, as well as overall performance.

3.2 Optical Technology and Part Selection

3.2.1 Detection Technology Selection

LiDAR

LiDAR makes use of the travel time of a laser pulse to determine the distance to an object. This is done by illuminating an area, usually with a pulsed laser, and measuring the time it takes for each pulse of light to return to the system.. LiDAR often takes advantage of extremely quick pulsed lasers and avalanche photodiodes to create a 3D image of the target area. Due to the nature of light being extremely quick LiDAR systems need to use photodiodes that can take many measurements within a short period of time such as an avalanche photodiode. The components needed to use LiDAR require extreme precision and very fast response times and are oftentimes very expensive.

LiDAR systems primarily use near-infrared light, as there is far less potential harm to humans if a stray beam were to enter someone's eyes. As LiDAR is often used in urban environments, the use of visible light also introduces the potential of creating dangerous distractions. Additionally, LiDAR devices have very large FOVs allowing for a lot of area to be scanned even when low to the ground.

LiDAR is very effective at high speeds since light is extremely fast and many points of data can be collected very quickly with negligible distance between data points. This means that the collected data will be very detailed and detections would be easier to make. However, the large amount of data collected

using LiDAR presents an issue when it comes to processing power and data storage as it will be more taxing on a computer to receive data points at such a high speed.

Since LiDAR mapping is entirely based on time of flight information, there is little risk of a false detection of something like an oil slick, since the light being received by the system will have extremely negligible differences in time of flight.

Machine Vision

Machine vision involves the use of a camera and software to identify information about a desired area. There are many things that can be measured with the use of machine vision but for this project it would be used to measure whether or not something is in the area being scanned and how large that something is. The camera receives the light reflected from the road and converts that information into digital data. This data is then transferred to a computer to be processed by a trained algorithm.

The camera converts the image data into digital data. The machine vision program converts the image data into a bitmap. Using this bitmap data, the program can look for inconsistencies in the received light to detect objects or depressions on the road. However, when using visible light, machine vision requires a lot of processing power. This is because the image data contains 3 different sets of values for red, green, and blue light. By converting the images into grayscale the image data is significantly reduced in size. Unfortunately, since the data is only in black and white the program is unable to differentiate between a pothole and any other road depression. To combat this, another camera can be used to capture images whenever a detection is made and a computer vision program can be used to confirm whether or not the depression is a pothole. The addition of another camera that only activates upon detection allows for better object detection without needing a lot of processing power.

At high speeds, it can be difficult to measure the target area with a high resolution since, when using a camera with a low FPS, there may be a large distance between frames. However, increasing the FPS will require an increased amount of processing power which may be difficult to achieve without a dedicated CPU.

The illumination system allows the program to gather more data from the grayscale images by increasing the contrast of the scanned area. There are a variety of ways to illuminate the target area including lasers, or LEDs. To properly illuminate the area there are a variety of things to consider such as angle, wavelength of light, and distance from the scanned area.

The camera will need a proper lens system to capture the transmitted light. When picking the lens system there are a lot of important characteristics to take into account. In order to capture a meaningful portion of the lane, the camera will need to have a large FOV so the use of a wide angle lens may be beneficial. However, this would introduce distortion to the image, so there needs to be a balance. The most common types of camera sensors are Charge-Coupled Devices (CCD) and Complimentary Metal Oxide Semiconductors (CMOS). Both of these sensors work primarily in the visible spectrum of light but can still detect Near-Infrared light. This allows a lot of flexibility when it comes to choosing the operating wavelength of the system.

Detection System Comparison

For this project there are a variety of factors that we need to take into account when designing the detection system. One of these factors is the image resolution when the device is operating at high speeds. With machine vision, the resolution is reliant on the frame rate of the camera being used. However, with an increased frame rate, there is a need for more processing power. LiDAR is entirely based on time of flight for a laser beam, meaning the distance between each data point being taken is reliant on how quickly the laser pulses, unless using a continuous wave laser, which would make the distance between data points negligible. For these reasons, LiDAR beats out machine vision when it comes to resolution.

Since our device will be relatively small, we will be severely limited on processing power. As such, it is important that our detection system requires minimal processing power. The processing power required for machine vision is quite variable due to many different parameters of the device but it is very possible to keep the processing power low. LiDAR, on the other hand, generally requires a lot of processing power. This is due to the large amount of data that is collected and processed by the LiDAR system. Most LiDAR systems require a dedicated CPU which is not something that is feasible for this project. For these reasons, machine vision beats out LiDAR in terms of processing power.

Another important factor to consider is the FOV of the device. For this project, we will need to scan a large portion of the road (6-10ft) while limited to being relatively close to the ground. Most LiDAR devices operate at high FOVs without distortion. Cameras, on the other hand, will start to have distortion at high FOVs. Due to the lack of distortion, LiDAR wins out over machine vision when it comes to FOV.

Table 3.2.1: LiDAR vs Machine Vision

Criteria	LiDAR	Machine Vision
Resolution	Excellent	Good
Processing Power	Poor	Good-Excellent
FOV/Distortion	Excellent	Good
Flexibility	Fair	Good
Cost	Fair	Good
Overall	Fair	Good-Excellent

The ability to operate at a variety of wavelengths is an important factor for this project, since having the option to change wavelengths would offer a safety net in case one design falls flat. LiDAR usually operates in the NIR region and the wavelength cannot be changed easily, as it would likely come packaged in one device. Machine vision allows for a variety of wavelengths to be used with very minimal changes needed for the device to work at other wavelengths. As such, machine vision beats out LiDAR in terms of its flexibility.

Since we are working on a tight budget, the issue of cost is the most important one. For machine vision, we will need to acquire a camera with an appropriate lens that can work at a high FPS along with a light source and the necessary equipment needed to operate it. For LiDAR, we would need to acquire and build an array of avalanche photodiodes, and a laser system to illuminate the road. Due to the high cost of the equipment needed for LiDAR, machine vision would be ideal when considering the budget.

While it may struggle when it comes to distortion and high speed resolution, machine vision is a much better option for this project as it can perform on nearly the same level as LiDAR whilst being cheaper, more flexible and requiring significantly less processing power. The increased flexibility of machine vision also allows for quick changes if something doesn't work out.

Wavelength of Light

Visible Light

Visible (VIS) light is the portion of the electromagnetic spectrum that is visible to the average human being. It is generally defined as the wavelengths of

light from about 400nm to about 700nm. Visible light is used in a variety of detection and/or scanning systems as well as countless other devices. Because of its visibility, it is extremely easy to work with and is often used for testing before switching to a non-visible wavelength of light. Visible light comprises a little over 40% of the light radiated by the sun and due to the small size of the visible spectrum it is the most densely radiated by the sun.

Infrared Light

Infrared (IR) light is the portion of the electromagnetic spectrum that ranges from around 780nm-1mm. Infrared light makes up a majority of the light emitted by the sun at around 50% but due to how large the IR spectrum is it is less densely radiated by the sun than the visible wavelengths. The infrared spectrum can be broken down into 5 main categories: Near-infrared (NIR), Short-wavelength infrared (SWIR), Mid-infrared (MIR), Long-wavelength infrared (LWIR), and Far-infrared (FIR).

NIR is the smallest portion of the infrared spectrum, ranging from about 780nm-1400nm, and is able to be captured by most visible-light cameras. It is most commonly used in fiber optic telecommunications and night vision devices. The SWIR spectrum ranges from about 1.4 μ m-3 μ m and is most commonly used for long range telecommunications. The MIR spectrum ranges from about 3 μ m-8 μ m and is often used for heat seeking applications. LWIR ranges from about 8 μ m-15 μ m and is used primarily for thermal imaging, since these are the wavelengths of light emitted by objects around -80-89°C, and thermal imaging devices don't need any kind of illumination, since the light is emitted with heat. FIR light ranges from about 15 μ m-1mm and is mostly used for medical purposes such as skin treatments.

Wavelength Comparison

Since SWIR, MIR, and FIR are not very practical for imaging devices they will not be considered for the comparison. We will be comparing the VIS, NIR, and LWIR spectrums for use in our illumination system. There are 5 factors that this comparison will be based on: cost, noise, ease-of-use, simplicity, safety.

Due to the constraints of our project, keeping the cost low is very important. NIR and visible light are very commonly used, making them much cheaper than LWIR devices. However, they require some sort of illumination. Although LWIR devices remove the need for illumination, they are still more expensive than VIS and NIR. Between VIS and NIR illumination sources, VIS devices tend to be cheaper, meaning that VIS wins in terms of cost.

In order to make accurate detections, we need to be able to clearly make out the differences in light levels on the road. For this reason, noise is a big concern for our project. The visible light spectrum is the most densely radiated spectrum of light by the sun, leading to a lot of noise. Though not as much as the visible light spectrum, NIR also suffers from a lot of noise from the dense radiation by the sun. LWIR has the most amount of noise, since almost everything being imaged will be releasing light in this spectrum. However, this is how LWIR imaging is done without illumination. Due to the sheer amount of LWIR light, it can be hard to make accurate measurements, which means NIR wins in terms of noise.

Ease-of-use is very important for testing our design. Given that visible light is easy to monitor, it is also very easy to test with and make fine adjustments. The same cannot be said for NIR or LWIR, since they both require special instruments to be able to see. NIR is slightly easier to work with than LWIR, since it can be viewed with most visible-light cameras. Visible light is by far the easiest to work with and wins in terms of ease-of-use.

Keeping the project simple will help to make mistakes more manageable and changes easier to make. VIS and NIR light are the same in terms of simplicity; they both require some sort of illumination source and a camera to view the light. LWIR is very simple with it only requiring a camera, thus making it the most simple wavelength to work with.

Potentially the most important factor to consider is the danger posed by using any of these wavelengths in a public environment. NIR and LWIR are both invisible to the human eye, which makes them highly unlikely to distract drivers on the road. Visible light can cause a distraction for other drivers but it also has the advantage of being noticeable for the human eye. LWIR is the safest, since no light is being sent to the road and the risk of distracting or injuring a bystander is not present. NIR is the next safest option as it does not run the risk of distracting a bystander, but it has an increased chance of permanent eye damage by a stray beam. However, considering that by the time the light hits the ground it will be extremely weak, there is a miniscule chance of the visible or NIR light causing damage to the bystander's eye. For these reasons, I believe that NIR and LWIR are the safer options.

Table 3.2.2: Wavelength Comparison

Criteria	Visible Light	Near-infrared Light	Long-wavelength Infrared Light
Cost	Good	Good	Poor
Noise	Fair	Good	Poor
Ease-of-use	Excellent	Good	Fair
Simplicity	Fair	Fair	Excellent
Safety	Good	Excellent	Excellent
Overall	Good	Excellent	Poor

While it might be a bit difficult to work with, and slightly more expensive than visible light, NIR light is safer to use on the road and there is less noise to worry about, making it an excellent choice for this project.

3.2.2 Detector Part Selection

Camera Selection For Machine Vision Detection System

IR Camera Requirements

When selecting the cameras for this project, there are a variety of requirements to consider. Since this device will be operating at high speeds, our camera will need to be able to capture good images at high intervals with minimal image blur. Additionally, our project requires that we scan a large area of the road from a set distance, so the camera we pick will need to have a high FOV lens to capture the desired area. Due to the large amount of processing power needed for machine vision, it would be ideal for the camera to record the image data in monochrome to save on bandwidth and processing power. Having a high resolution is important for our design, but, due to constraints on the processing power of our device, a camera that has variable resolution with a high limit is preferred. For this project, we will be running cables through a car's physical firewall, so a camera that can transfer data through without the need to convert to a different cable type would be preferred. Finally, in the detection system, we will be using infrared light, so our camera will need high sensitivity to infrared while still being relatively cheap.

To capture video at high speeds, our camera will need to have a high framerate. Our target max speed for operation is 40mph, which equates to just under 60 feet per second. Since our laser sheet will be about 2 feet in length, we will need our camera to have a framerate of at least 30 FPS. It is possible that we will want to increase our accuracy, so a camera with the option to increase frame rate would be preferred. When working at high speeds, image blur starts to become a concern. Image blur occurs when image data for individual pixels is recorded at different times. This is dependent on whether the camera uses rolling shutter or global shutter. When a camera sensor uses rolling shutter technology, an image is generated by reading the pixel data row by row. While working at high speeds, the image data from the starting pixels and the later pixels will be recorded at different positions, causing the resulting image to blur. Sensors with global shutter technology read the image data of every single pixel at the same time, resulting in a blurless image. A camera with a sensor that has global shutter is much preferred for this project.

For the camera to capture the entire desired area of the road, it will need a lens with a high field of view (FOV). Based on our calculations, the camera lens must have a vertical field of view (FOV) of at least 28° and a horizontal FOV of at least 74° or 103° , depending on whether the target area is 2×6 ft or 2×10 ft, respectively, when viewed from a height of 4 ft above the ground. If the lens that comes with the camera does not fit these specifications, a camera with an easily swappable lens will be acceptable.

Machine vision requires a sizable amount of processing power, meaning that the ideal camera can make up for this in some way. To keep the necessary processing power low, we may need to limit the resolution of the camera. But, having a low resolution will decrease the accuracy of our measurements, so a camera with variable resolution with a high max resolution would be ideal. Having the option to change resolution will make it easy to implement quick changes to the design. Another way to limit the amount of data coming from the camera is to find a camera that records in monochrome. Without the color data, the amount of information being transferred is reduced and less processing power is needed to process the image data.

The image data from the camera needs to be sent over the length of a car in cables small enough to fit through a car's physical firewall. To avoid signal loss, our camera should be able to transfer data across the car without the need to convert from one cable to another. We have decided that USB 2.0 and 3.0 are the best choice for our device, since they have minimal signal loss and can be very cheap even at long lengths. As such, it is preferred that the camera we choose has a built-in USB connection, or, at the very least, can be converted to USB if need be.

The illumination system for the device will be using near-infrared light, which can be captured by visible light cameras, although most filter out infrared light. If our camera is a visible-light camera, to make infrared light detection easier, our camera should not have an IR filter. Additionally, if a replacement lens is required, it should not have an IR filter as it would inhibit the functioning of the camera. Another option is to use a near-infrared camera sensor with a custom made mount for the sensor and lens.

IR Camera Comparison

When choosing a camera for the detection system, there were 3 main candidates: the Raspberry Pi Camera Module 3 NoIR Wide, the Arducam B0332, and the Mira050 NIR sensor with custom mount and lens setups. All of these options fit the basic needs of our project, but none of them are perfect. The pros and cons of each camera will be evaluated in the table below.

The Raspberry Pi Camera Module 3 NoIR Wide is designed to be used with a Raspberry Pi, making it easy to implement in our design. It consists of an IMX708 sensor as well as a high FOV lens already mounted to the device. The IMX708 sensor captures images in color and uses rolling shutter, making it suboptimal for the high speed environment of our device. The lens has a horizontal FOV of 102° and a vertical FOV of 67° on an M12 mount, making it suitable for the smaller target area and potentially the larger target area since the difference is minimal. This camera sends and receives information via a CSI ribbon cable. However, it can be converted to USB with an adapter. This camera can be operated at or above 30 FPS in 480p, 720p, and 1080p resolution. There is no IR cut filter in this camera, so it can pick up infrared light quite easily.

The Arducam B0332 is also designed to work on the Pi and other similar devices, and, as such, is easy to implement in the design. It consists of an OV9281 sensor and a 70° FOV lens. The OV9281 image sensor captures monochrome images and uses global shutter, making it a good option for high speed applications. The lens has a horizontal FOV of 70° and a vertical FOV of about 48°, which wouldn't work for either target area. However, the camera has an M12 lens mount, making swapping lenses quite easy. The Arducam has a built-in USB connection for data transfer, which is ideal for our design. This camera operates at up to 100 FPS in 240p, 360p, 800x600, and 1280x800 resolutions, and has no built-in IR filter.

The Mira050 image sensor is designed to capture near-infrared light and is the best option in terms of detecting NIR light. It captures images in monochrome and uses global shutter. This sensor can operate at up to 120 FPS with a max resolution of 600x800. The biggest downside to this option is that it

requires that we build a custom mount for the sensor and lens. We would also need to create a board that allows the data to be transferred through a USB cable. This is the most complicated option, but it meets the detection requirements the best.

Table 3.2.3: Camera Comparison

Criteria	Raspberry Pi NoIR Wide	Arducam B0332	Mira050 NIR Image Sensor
Framerate	50 @ max res	100 All	120 All
Resolution	1080p; Variable	800p; Variable	800x600; Variable
Shutter Type	Rolling	Global	Global
Lens Field of View	102° (H), 67° (V); M12 Mount	70° (H), 48° (V); M12 Mount	None
Image Color	Color	Monochrome	Monochrome
Cable Connection	CSI Ribbon Cable	USB 2.0	None
IR Sensitivity	Good	Good	Excellent
Cost	Good (USB Adapter)	Good (Alternative Lens)	Poor (Many Additional Costs)
Overall	Good	Excellent	Fair

While its resolution is a bit low and it needs a replacement lens, the Arducam B0332 is the clear winner here. All of the options have good IR sensitivity, acceptable frame rates, and acceptable resolutions. But, the Arducam stands out, since it uses global shutter and a built-in USB connection. The Mira050 image sensor is best suited for NIR imaging, but it requires a ton of extra work and additional costs that other options do not. The Pi camera is an excellent option, but the use of rolling shutter is too big of an issue for this project.

RGB Camera Requirements

Choosing the right RGB camera for a vehicle-mounted detection system requires balancing field of view, data format, bandwidth, frame rate, and motion stability. To capture lane-level images from approximately 4 feet above the road,

the camera must have a horizontal field of view (HFOV) of around 90° to span at least 8 feet of road width. This allows the images to capture the majority of a 10-12 foot lane. It is also important to consider that exceeding 100° diagonal FOV increases distortion, which can reduce detection accuracy [93]. Therefore, lenses in the 85-90° HFOV range provide a practical compromise between coverage and low distortion, making them ideal for real-time road inspection tasks. Additionally, we determined that a resolution of 1920x1080 is ideal for our object detection model, as it provides a sufficient pixel density to preserve pothole features that models like YOLOv5 use to distinguish between objects [91].

The required frame rate is at least 30 FPS, which makes sure that multiple frames of each pothole are captured even when traveling at speeds of 30 mph. While the theoretical minimum is just under 5 FPS for detecting a single pothole, using a 30 FPS camera gives us at least 3 to 5 frames per event, which is entirely dependent on the size of the pothole. This improves object detection confidence in models such as YOLOv5, which depend on frame consistency to reduce false positives and increase classification accuracy [91]. Studies have shown that frame rates below 20 FPS can negatively impact CNN-based object detection, especially in embedded systems where frame dropping can occur [92].

The data format and USB bandwidth are equally important. We selected the YUY2 format because it is uncompressed and reduces CPU load compared to MJPEG, which requires decoding. However, transmitting YUY2 at 1920x1080 and 30 FPS requires USB 3.0, since USB 2.0 cannot maintain sufficient throughput for uncompressed high-resolution streams [94]. It is also important to mention that the Raspberry Pi's USB 2.0 ports use a shared data bus. Since our design will feature two cameras, we must use USB 3.0 for the RGB camera due to bandwidth constraints. Additionally, we prefer cameras with a global shutter, which avoids motion blur that occurs with rolling shutter sensors when the vehicle is in motion. For applications involving movement, such as road scanning from a vehicle, global shutter sensors improve object recognition performance [95].

RGB Camera Selection

The first camera we looked at was the Arducam Global Shutter USB 3.0. This camera has 1920x1080 resolution at 30 FPS using USB 3.0 and supports the YUY2 format. It uses a global shutter, which reduces motion blur when capturing fast-moving images, such as roads during vehicle motion. It also features an 85° horizontal FOV and a 98° diagonal FOV, which fits well within our target range. This camera is compact and well-documented, making it easier to use in the design.

The second camera we looked at was the ALPCAM 1080p USB Camera. This module supports USB 3.0 and 1920x1080 resolution and outputs in YUY2, so it meets our data format and bandwidth criteria. However, it uses a rolling shutter sensor, which is more prone to motion distortion. While it has a 3.6 mm lens, which should provide roughly 85-90° HFOV, the exact FOV and lens distortion are not well documented. It is much cheaper than the Arducam, but its motion-handling limitations could impact detection reliability during use.

The final camera we evaluated was the IFWATER HDMI+USB Camera. This camera has a resolution of 3840x2160 at 60 FPS and supports USB 3.0, which is more than what we need. However, there is limited information on whether it supports YUY2 or uses MJPEG compression by default. It also lacks information about its shutter type, which means it likely uses a rolling shutter. Also, while it includes HDMI output, that feature is not very useful for our project. It's the most expensive of the 3 cameras, you can't change out the lens as needed, and its physical size and lack of sensor detail made it less appealing for a compact, embedded vision application.

Table 3.2.4: RGB Camera Comparison Table

Feature	Arducam Global Shutter	ALPCAM USB Camera	IFWATER HDMI+USB
Resolution	1920x1080	1920x1080	3840x2160
Frame Rate	30 FPS	30 FPS	60 FPS
USB Version	USB 3.0	USB 3.0	USB 3.0
Output Format	YUY2	YUY2	MJPEG / YUY2
Shutter Type	Global Shutter	Rolling Shutter	Rolling Shutter
Horizontal FOV	85°	~85–90°	Not Specified
Diagonal FOV	98°	Not Specified	Not Specified
Price	\$145.99	\$84.50	\$154.99
Lens Swappable	Yes	Yes	No
Overall	Excellent	Good	Fair

After weighing the requirements of our pothole detection system and comparing camera features, we selected the Arducam Global Shutter USB 3.0 Camera. It is the only camera that met all five of our important needs including 1920x1080 resolution, 30 FPS, USB 3.0, YUY2, and a global shutter. Its 85° HFOV and 98° DFOV allow us to cover close to 8 feet of road width from a 4-foot height, without introducing too much distortion. Unlike the other cameras, Arducam's feature set was made for embedded vision designs, offering the balance of performance, image quality, and motion stability required for accurate AI-based pothole detection.

Filter Selection

Type of Optical Filter

Since our project utilizes infrared light and we are using a visible light camera, it is very important that the visible light is filtered out before reaching the camera. It is important that the filter only lets through light of the same wavelength as our laser in order to minimize noise. The filter also needs to work in a high FOV optical system like that of our device.

We considered 2 different types of optical filters for this design: a dielectric bandpass filter and an absorptive color bandpass filter. These both have their advantages, but only one fits our project. The table below is used to explain what we picked.

Dielectric bandpass filters are composed of many thin films of dielectric material that make use of constructive and destructive interference to only allow a specific wavelength of light through. For this reason, they have very little error, often in the 10s of nanometers. However, at high angles of incidence(AOI) the center wavelength of the passband decreases due to the increased optical path length, changing the constructive interference condition. Due to this, these bandpass filters severely limit the FOV of an optical system. These filters tend to be quite expensive.

Absorptive color filters rely on a material's properties to absorb light of different wavelengths than the desired wavelength, resulting in the desired light passing through and undesirable light being absorbed by the filter. These filters tend to have very broad passbands since absorption properties are quite broad. The advantage to absorptive color filters is that their properties are minimally affected by the angle of incidence and they do not tend to limit the FOV of an optical system. Absorptive color filters tend to be much cheaper than dielectric filters.

Table 3.2.5: Optical Filter Comparison

Criteria	Dielectric Filter	Absorptive Color Filter
Passband Size	Excellent	Fair
High AOI Operation	Poor	Excellent
Cost	Fair	Excellent
Overall	Fair	Excellent

While they do have a broad passband, absorptive color filters are a better choice for our design overall as they do not limit the FOV of our device as much as a dielectric filter would. They are also a bit cheaper, which will help us stay within the budget.

Absorptive Color Bandpass Filter Selection

The process of selecting the absorptive color filter was entirely based on what was available and met the requirements for our design. In order to be chosen, the filter will need to center on a wavelength of light in the NIR region, with a reasonably small passband. The 2 options I landed on were: the Hoya RT830 Colored Glass Bandpass Filter and the Schott RG905 Colored Glass Bandpass Filter.

The Hoya filter is centered on a wavelength of 830nm, with a full width at half maximum(FWHM) of 260nm and does not pick up light with a wavelength smaller than 700nm. The Schott filter, on the other hand, is centered on a wavelength of 880nm with a much larger FWHM. It also picks up some light around the 500nm range.

Table 3.2.6: Absorptive Filter Comparison

Criteria	Hoya RT830	Schott RG905
Passband	Good	Poor
Center Wavelength	Excellent	Excellent
Cost	Good	Fair
Overall	Excellent	Fair

The Hoya bandpass filter is a much better option for our design as it has a much smaller passband and costs less than the Schott filter.

3.2.3 Illumination Technology and Part Selection

Light Emitting Diode (LED)

Light Emitting Diodes consist of 3 regions: an n-type, negatively charged semiconductor; a p-type, positively charged semiconductor; and a depletion region between the n-type and p-type semiconductors. By applying a current through the LED from the p-type semiconductor to the n-type semiconductor, the depletion region begins to narrow as the negatively charged electrons and

positively charged holes move closer together. With this movement, the electrons and holes start to come into contact with each other and release energy as photons in a process called spontaneous emission.

Due to the variable nature of spontaneous emission, the light emitted by the LED has a high amount of variability. LEDs emit light omnidirectionally, are relatively low cost, and are low energy. In order to generate enough light, many LEDs can be used in an array. This will also decrease the coherency of the emitted light, since there will be multiple light sources in the array.

Laser

Laser is an acronym meaning “Light Amplification by Stimulated Emission of Radiation.” As is evident by the name, lasers take advantage of stimulated emission to generate light. In order to achieve stimulated emission a laser requires 3 major parts: a pump source, a gain medium, and a laser cavity.

The gain medium is the material used to achieve a population inversion. They are designed to have energy levels separated by the energy of photons of the desired wavelength of light. By exciting the gain medium with the pump source, the higher energy level becomes populated with excited electrons until there are more electrons in the excited state than the ground state; this is known as population inversion. However, when population inversion is achieved, most of the excited electrons will spontaneously emit photons. This is where the laser cavity comes in. The gain medium sits inside of the laser cavity with 2 mirrors on either end; these mirrors have different reflectivities with one mirror being 100% reflective and the other being less than 100% reflective.

As the light is emitted from the gain medium, it bounces off of the mirrors in the laser cavity and back into the gain medium. When the light returns to the excited electrons, it causes them to emit photons of the same wavelength. This leads to a cascading effect where the light coming through the medium increases and, with it, the amount of excited electrons being hit causes more and more photons to be emitted. As the amount of light in the cavity increases, the light is eventually able to escape the cavity through the mirror with <100% reflectivity. Because of the way that the photons are emitted, there is very little variance in the wavelength of the emitted light. This results in the laser emitting coherent light of a single wavelength.

Illumination Technology Comparison

There are three important variables to consider when picking an illumination source for this project: cost, coherency, and divergence. Keeping the

cost low is important to this project, since we are funding the project ourselves. LEDs are a very cheap option, even in the infrared light ranges. Lasers are generally quite expensive relative to LEDs, but they have gotten a lot more affordable for the average person, although IR lasers tend to be a bit more expensive. LEDs beat out lasers in terms of the cost.

In order to be able to properly image the light with minimal interference, we will need to have a light source that is highly coherent. While LEDs don't have a lot of variance when it comes to the wavelength of light emitted, an array will be needed, which introduces a lot of incoherency with the light source. Lasers, on the other hand, produce a single wavelength of light with minimal variance and as such are very coherent. Lasers are much more coherent than an LED array.

Having an illumination source with minimal divergence is important for this project, since being able to control the light into the specific area we desire is important to save on the necessary power for the device. LEDs emit light omnidirectionally and thus have a lot of divergence, which will result in a lot of the emitted light not being picked up by our detection system. Lasers, on the other hand, have very little divergence and the light is very easy to manipulate, allowing us to prevent wasting energy on light that we are not imaging. For these reasons, the laser wins out over LEDs in terms of divergence.

Table 3.2.7: Illumination Source Comparison

Criteria	Light Emitting Diode	Laser
Cost	Excellent	Fair
Coherency	Fair	Excellent
Divergence	Poor	Excellent
Overall	Fair	Excellent

While it may cost a bit more, the laser light is coherent with minimal divergence, making it a much better option overall.

Laser Selection

When choosing a laser for this project, there were two important factors to consider: the power and the wavelength of the laser. Our project relies on the use

of near infrared light, so our laser wavelength will need to be within that range of wavelengths. The power of the laser will need to be enough for the camera to pick up over the noise from the sun.

The only necessary requirements when it comes to choosing the wavelength of our laser is that it be within the near infrared range of light and captureable by our chosen camera. The near infrared range of light is about 780-1400nm [ISO Source] and silicon-based CMOS sensors like the one used in our chosen camera, the Arducam B0332, are less effective starting around 950nm [Si Photodiodes Source]. Due to this, our laser should have a wavelength between 780-950nm. This is still quite a large range of wavelengths so we narrowed it down even more based on the bandpass filter that we chose. Since the bandpass filter was chosen based on availability we are able to narrow down our ideal wavelength for the laser to be 830nm based on the properties of the bandpass filter.

Choosing the power of our laser is much more nuanced than picking the wavelength. When deciding on the power of our laser, we need to consider not only the requirements of our project, but also the safety of its operation. Since we are using an absorptive color bandpass filter with a large passband, the power requirements for our laser become much more significant than when using a narrowband filter. In order for our laser to clearly show up through our broader filter, we will likely need a class 3b laser somewhere in the 100s of milliwatts range. To have a strong enough laser while staying as safe as possible, the ideal laser would be in the 200-300mW range.

Table 3.2.8: Laser Diode Selection

Parameter	Civil Lasers 808nm 150mW Laser Diode	ThorLabs L820P200 Laser Diode
Power	Good	Excellent
Safety	Fair	Fair
Modes	Single	Single
Wavelength	Good	Excellent
Overall	Good	Excellent

While the decreased power of the 808nm laser diode does make it a bit safer, the 820nm laser diode fits the requirements of our project better. Being closer to the center wavelength of the bandpass filter makes it transmit just that

little bit better that we may need in this project, and the increased power will allow it to stand out much more in a high-noise environment.

Beam Expander

Our project requires us to expand laser light over a large area of the road. The expansion system for our design has a few important factors that need to be considered when picking the lenses. Our target area is very large so the first important factor to consider is the fan angle of our expander. This project is designed to cover a specific horizontal area of the road, so our line generator requires the ability to expand the beam enough to meet our specifications

The target of our beam is an 8 ft line of the road. Our device will operate about 4ft above the ground, meaning that the laser beam will need to be spread out 90° in the horizontal direction. There are 2 types of lenses that can properly achieve the requirements of our project: cylindrical and Powell lenses. Both of these lenses come with a variety of fan angles that match the criteria of our design, but they do vary in some important ways.

Cylindrical Lens

Cylindrical lenses are designed to focus light in only one direction. They accomplish this by being curved in only one direction unlike a spherical lens that has a consistent curve all around. By passing a laser into a cylindrical lens it can be focused in the same direction as its curve. This process can be done in reverse to diverge the lens in the direction of its curve. By doing this, the laser beam is stretched into a line. However, cylindrical lenses do not spread the light uniformly, creating a bright spot in the center of the line and a power drop off at the edges of the line. Additionally, laser lines created by cylindrical lenses with large fan angles become curved at the edges.

Powell Lens

Powell lenses are a type of aspherical cylindrical lens that are designed to spread laser light into a line with a uniform brightness across the entire line. Unlike with cylindrical lenses, the laser line produced by a Powell lens is the same brightness all the way through. Due to field aberration the line will not appear fully uniform in brightness but the consistency is very useful.

Beam Expander Selection

This project will be using machine vision to monitor for changes in the intensity of the light on the road in order to detect potholes. The device will be

operating in a high-noise environment, making detection very precise. The powell lenses offer uniform distribution. The cylindrical lenses can achieve the desired line but will result in an uneven distribution of light across the target area. The table below describes the factors considered when choosing between the expansion methods.

Table 3.2.9: Beam Expansion Selection

Parameter	Cylindrical Lens	Powell Lens
Fan Angle	Excellent	Excellent
Light Distribution	Fair	Excellent
Overall	Good	Excellent

Overall, both of these options are good choices for this project. However, the ability to have the light uniformly distributed across the line is what makes the Powell lenses the better choice for this design. The precision of our detection system is what made us decide on the Powell lenses, since it would simplify detection.

Powell Lens Selection

When it comes to choosing which Powell lenses to use in our design, the main considerations are the fan angles of the lenses and the cost. In order to choose the correct powell lenses for this project we needed to calculate the necessary fan angles of our lenses. With the target area being an 8 ft line and the position of the laser at 4ft above the ground, we were able to calculate the necessary fan angles using trigonometry. These calculations resulted in our required fan angle being at least 90°. From this information, we decided on getting lenses with a fan angle of 90°. The decision is detailed in the table below.

Table 3.2.10: Powell Lens Selection

Parameter	Edmund Optics Powell Lens	Jolooyo Powell Lens
Fan Angles	Fair	Excellent
Cost	Poor	Excellent
Overall	Poor	Excellent

We chose the Jolooyo Powell lens for its affordability and availability at our required fan angle.

Beam Shaper and Collimator

Powell lenses work best when the incoming light is collimated. While we will not be able to collimate the light coming into the second lens in our system, having the light entering the first lens be collimated will help the lenses perform their intended function as well as possible. These lenses also require that the incoming beam be a certain size. According to the specifications of the Jolooyo Powell lenses, the diameter of our input beam should be between 0.8 and 1.0mm. Collimating our laser diode is a simple process and can be done by placing an aspheric lens one focal length away from the diode's beam waist. The diode we have chosen has an average fast axis divergence of 17° and an average slow axis divergence of 9° . Due to the difference in divergence the output beam will be elliptical and without an additional beam shaper both axes will not meet the requirements of the powell lens we selected. A fast axis collimator was considered, however we were unable to find any with a small enough form factor and focal length to fit our system. A cylindrical beam shaper was considered as well.

Cylindrical Beam Shaper

A cylindrical beam shaper takes advantage of the properties of cylindrical lenses to either expand or reduce the size of the beam on one axis. They are composed of 2 cylindrical lenses of different focal lengths to expand/reduce the beam based on equation 3.1 below.

$$M=f_2/f_1$$

Eq. 3.1: Magnification of a 2 lens beam expander

With the divergences of our laser diode being 17° and 9° , our beam shaper will need to change one axis by about a factor of 2 in order for the axes to most closely match. This system will require 2 cylindrical lenses and the initial aspheric collimating lens. The beam shaper, however, requires 2 additional lenses in the system and the precise alignment of the two cylindrical lenses so that the axis is properly expanded/compressed, making it a very costly and expensive addition

Ultimately, it was decided that our beam did not need to be shaped as long as one axis could meet the requirements of our lens. The effects on the Powell lens output were minimal and our system cost would stay low.

Collimating Lens Selection

For the collimating lens that we will be using in our system, it will need to be an aspheric lens so as to not introduce any aberrations to the collimated beam of light. The two other significant factors are whether it can collimate the slow axis to be within the 0.8-1mm range of our Powell lenses, and the cost of the lens.

The Thorlabs 355392 Molded Glass Aspheric Lens has a focal length of 2.8 mm, is designed for use with 830nm light, and is sold for about \$65. Using equation 3.2, we found that this lens would collimate the slow axis to a size of about 0.44 mm and the fast axis being collimated at about 0.84 mm. The Thorlabs 354230 Molded Glass Aspheric Lens has a focal length of 4.5 mm, is designed for use with 780 nm light, and is sold for about \$69. Using equation 3.2, this lens would result in the slow axis being collimated to a size of 0.71 mm and the fast axis at about 1.35 mm. Although this lens is designed for use with 780 nm, it is only slightly off from our wavelength of 830nm, meaning there will be a very slight change in the focal length of the lens, but it will be nearly unnoticeable. The Thorlabs 354330 Molded Glass Aspheric Lens has a focal length of 3.1mm, is designed for use with 830 nm light, and is sold for about \$69. Using equation 3.2, we found that this lens would result in the fast axis being collimated at a size of about 0.93 mm and the slow axis being collimated at a size of about 0.49 mm. These three lenses are compared in the table below.

$$D=2f*\tan(\Theta/2)$$

Eq. 3.2: Collimated beam size

Table 3.2.11: Aspheric Collimating Lens Selection

Parameter	Thorlabs 355392	Thorlabs 354230	Thorlabs 354330
Collimated Beam Size	Fair	Poor	Good
Designed Wavelength	Excellent	Good	Excellent
Cost	Good	Good	Good
Overall	Good	Fair	Excellent

The Thorlabs 354330 aspheric lens provides a great collimated beam size by keeping the fast axis within the desired range whilst making the slow axis as large as possible. It is also designed for 830 nm light making it very compatible with our 820 nm laser. The 355392 lens is also a good option however it results

in a smaller beam on the fast axis potentially causing more unwanted effects on the output of the Powell lens

3.3 Software Technology Comparisons

3.3.1 Computer Vision Models

Single-Stage vs Two-Stage Object Detection Models

To determine the best object detection model for our design, it is important to consider the specific constraints and needs of the system. Typically, the main factors when selecting a model include speed, accuracy, hardware demand, model complexity, and training effort. In our case, accuracy and hardware demand are the top priorities. Accuracy is critical because the model will directly influence whether sensor data is kept or discarded, which has a significant impact on the overall reliability of the system. Hardware demand is equally important, as the detection model will run on a Raspberry Pi, which has more limited processing power compared to what many object detection models are built for [17]. Since our design does not require real-time detection, speed is still relevant, but not the most important factor. The system cannot experience major slowdowns, but the software is not dependent on detections being faster than the frame rate of the RGB camera.

Object detection models are generally grouped into two categories: single-stage and two-stage models [11],[13]. Two-stage models operate in two parts: a region proposal stage and an object classification stage. During the region proposal stage, the model uses a lightweight algorithm, such as selective search, to scan through the image and identify areas that might contain objects. At this point, the model does not attempt to identify the objects; it only highlights regions that stand out, such as a person or a tree in a field. The second stage then refines these proposed bounding boxes to fit more tightly around the objects and tries to classify them. Common examples of two-stage models include R-CNN, which uses selective search, and Faster R-CNN, which replaces selective search with a Region Proposal Network (RPN) [12]. Two-stage models generally offer higher accuracy, better object localization, and improved noise filtering [13]. However, they tend to run slower, place more demand on hardware, and involve a more complex training process due to the need to train both stages separately [13].

Single-stage models, in contrast, perform both object localization and classification in a single pass [11]. These models take the full image and divide it into a grid, with each grid cell responsible for detecting objects whose centers fall within it. A backbone Convolutional Neural Network (CNN) is used to convert the

image into a feature map, which helps the model pick out useful information [13]. Each cell predicts one or more bounding boxes and assigns coordinates and dimensions to them. The model then calculates a confidence score for each bounding box, representing the likelihood that an object is inside it. In addition, it produces a probability for each class type, showing how likely the object belongs to each class. Some models also use anchor boxes to account for different object sizes and shapes [16]. Any bounding boxes with low confidence scores are filtered out and, finally, Non-Maximum Suppression (NMS) is applied to remove overlapping boxes in favor of the one with the highest score [16]. Popular single-stage models include YOLO (You Only Look Once), which typically uses Darknet as its backbone, and SSD (Single Shot Detector), which often uses VGG-16 [14],[15],[16]. The benefits of using a single-stage model include: faster processing, easier training, less demand on hardware, and better handling of various object sizes [14],[19]. The main drawbacks are: slightly lower accuracy overall, and a reduced ability to detect partially hidden objects [13], [18].

Table 3.3.1: Single-Stage vs Two-Stage Object Detection Models

Criteria	Single-Stage	Two-Stage
Speed	Good	Poor
Accuracy	Fair-Good	Good-Excellent
Complexity	Fair-Good	Poor
Hardware Demand	Fair-Good	Poor
Flexibility	Good	Excellent
Training Time	Fair	Poor
Overall	Good	Fair

Although accuracy is our main concern, the 10-15% improvement offered by two-stage models doesn't make up for the trade-offs in hardware demand, speed, complexity, and training time. Two-stage models place too much strain on a 64-bit quad-core Arm Cortex-A76, unless major hardware upgrades are made [12],[13],[17],[21]. While speed isn't a top priority, since this isn't a real-time detection system, a two-stage model can still take over 10 seconds to process just 5-10 images [12],[19]. This delay could seriously impact the accuracy of recording pothole locations, as we rely on the user's GPS data immediately after detection. Based on the data, it's clear that a single-stage model is a much better fit for our design.

YOLOv5 vs SSD Single-Stage Object Detection Models

To determine the most suitable single-stage detection model, we focused on two of the most widely used options: YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector). Our goal was to find a model that offers high accuracy, low hardware demand, reasonable speed, and manageable training complexity. Since our task is binary (determining whether a specific object is present in the frame or not), this simplicity may positively affect model performance compared to more complex, multi-class detection tasks [25].

The SSD300 architecture begins its detection process with a backbone network, such as VGG-16, which extracts feature maps from the input image [27]. These maps are then passed through a series of extra convolutional layers that progressively reduce in spatial resolution. Higher-resolution layers are better at detecting smaller objects, while lower-resolution layers focus on larger ones [27]. Default anchor boxes are applied at each feature map location, and for each anchor, the model predicts bounding box offsets and class probabilities. Finally, Non-Maximum Suppression (NMS) is used to remove low-confidence and overlapping detections. SSD300's advantages include: lower hardware requirements, a simpler and more compact design, fast initialization, and low memory usage [16],[25],[27]. However, it also has some drawbacks; it struggles with small object detection, generally has lower accuracy, offers limited built-in training features, and is increasingly considered outdated when compared to more modern models like YOLOv5.

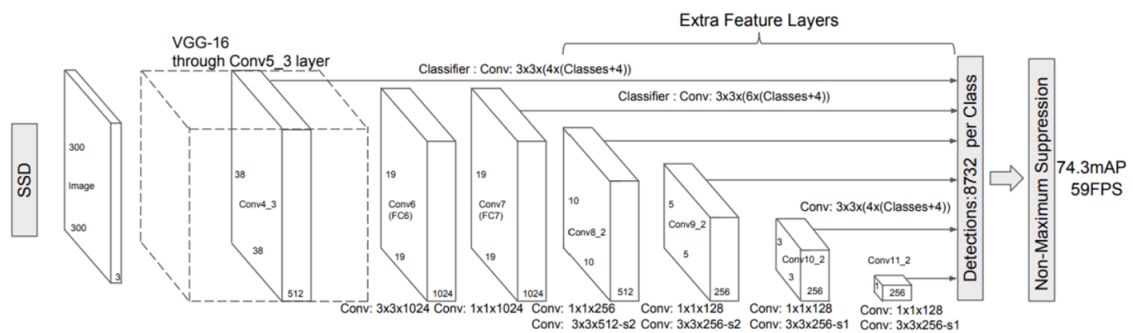


Figure 3.1: Architecture of the SSD300 model. Adapted with permission from [19].

The YOLOv5 architecture is divided into three main parts: the Backbone, Neck, and Head. The model begins with the Backbone, typically CSPDarknet, which is responsible for extracting important features from the input image. These features are then passed to the Neck, where information is combined from different layers using a Path Aggregation Network (PANet) and Spatial Pyramid Pooling-Fast (SPPF). This helps the model detect objects of varying sizes more

effectively. The Head is responsible for predicting bounding boxes, confidence scores, and object classes. After this, anchor boxes are applied and Non-Maximum Suppression is used to remove overlapping and low-confidence detections. The advantages of the YOLOv5 architecture include: support for multi-scale training, automatic anchor tuning, higher accuracy, faster inference, and an active development community [23],[24],[26]. However, its disadvantages include a greater architectural complexity and slightly higher hardware requirements compared to simpler models such as SSD300 [23],[26].

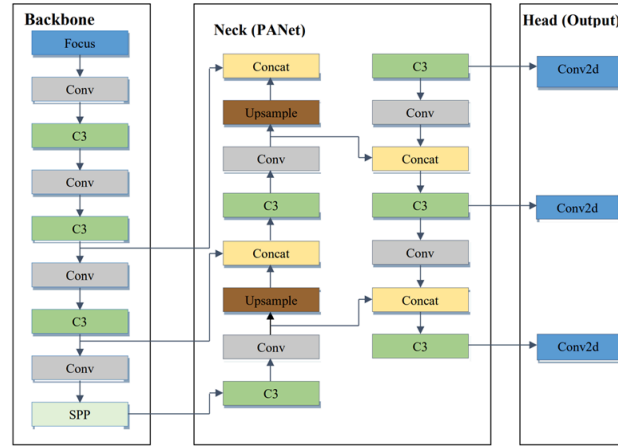


Figure 3.2: YOLOv5 Architecture (Nepal & Eslamiat, 2022)[14], with permission from [15].

The difference in hardware demand is not a strong enough reason to choose the SSD300 model. While the YOLOv5 model does require slightly more hardware resources, its significant improvements in accuracy, small object detection, training support, and multi-scale detection make it a far better choice for our application [17], [21],[27]. YOLOv5 comes in five versions: n, s, m, l, and x, ranging from smallest to largest in size and complexity, respectively [18]. The YOLOv5n model is capable of running at approximately 5 frames per second on the Raspberry Pi 5, while still allowing the two cameras and the laser detection algorithm to operate in parallel [17],[23]. On the other hand, YOLOv5s can achieve around 2 frames per second and offers up to a 20% increase in accuracy. However, the added resource demand makes it nearly impossible to run any other detection logic on the Raspberry Pi at the same time. While YOLOv5s remains a viable option, YOLOv5n offers a better balance of speed, efficiency, and system-wide performance, making it the most suitable model for our design [17],[23].

Table 3.3.2: YOLOv5 vs SSD300 Object Detection Models

Criteria	YOLOv5	SSD300
Accuracy	Good–Excellent	Fair
Small Object Detection	Good	Poor
Hardware Demand	Fair-Good	Excellent
Speed	Good	Good
Training	Excellent	Poor
Ease of Implementation	Fair	Good
Multi-Scale Detection Support	Excellent	Fair
Community & Documentation	Excellent	Fair
Complexity	Fair	Good
Overall	Good	Fair

YOLOv5n Optimization Methods for Raspberry Pi

The native deep learning framework used for the YOLOv5 model is PyTorch, which is primarily intended for building, training, and testing neural networks. While PyTorch is easy to use and requires no additional setup, it presents significant limitations when it comes to runtime efficiency on devices like the Raspberry Pi 5. Specifically, PyTorch lacks optimizations for embedded hardware, which leads to lower inference speeds and inefficient use of system resources [21],[23]. To improve performance without noticeable accuracy loss, several common optimization methods can be applied. These include converting the PyTorch model to a proprietary inference engine like ONNX Runtime or NCNN, and applying INT8 quantization. It's worth noting that, while using a USB hardware accelerator (like a Coral TPU) is another viable approach, the added cost of around \$100 makes it less attractive for budget-sensitive designs [26].

Exporting the PyTorch model to ONNX has been shown to increase inference speed by approximately 3 to 4 times on devices like the Raspberry Pi 4, with less than 1% accuracy loss in most cases [24],[25],[42]. ONNX enables multithreading, quantization support, more efficient CPU usage, and reduced memory consumption, making it a well-balanced option in terms of performance and resource usage [23],[42]. The primary downside is the reduced flexibility, as the model must be exported and integrated into a slightly more complex pipeline.

For even greater optimization, the ONNX model can be converted to NCNN, a high-performance neural network inference framework designed

specifically for ARM devices. NCNN delivers the fastest performance on CPU-only systems, making it ideal for final production deployment on mobile devices and Raspberry Pi boards [23],[24],[26]. However, NCNN can be complex to implement, offers less flexibility, and has limited community support compared to PyTorch or ONNX [23],[24].

Another major optimization strategy is quantizing the model from FP32 to INT8. This technique can be applied to both ONNX and NCNN formats. INT8 quantization reduces the numerical precision of weights and activations, significantly improving inference speed (up to 2–4× faster), cutting model size by ~75%, and lowering power consumption [22],[24],[42]. The tradeoffs include a slight drop in accuracy (typically <1%), a more complex workflow, and additional library or hardware dependencies depending on the runtime environment [23],[24],[42].

Table 3.3.3: Object Detection Model Optimization Methods

Criteria	PyTorch	ONNX	NCNN	PyTorch (INT8)	ONNX (INT8)	NCNN (INT8)
FPS	~1.5–2.5	~4-5	~7-10	~3-5	~8-10	~12-15
Accuracy Change	Baseline	Baseline	Baseline	~≤1% drop	~≤1–2% drop	~≤1% drop
Ease of Setup	Excellent	Good	Fair	Good	Fair	Poor
Memory Use	Poor	Fair-Good	Fair	Fair-Good	Excellent	Good - Excellent
Multicore	Poor	Good	Excellent	Fair	Good	Excellent
Overall	Poor-Fair	Good	Good	Fair-Good	Excellent	Excellent

Although it may seem that the best option is to use NCNN with INT8 quantization, the additional speed gain may not be worth the increased complexity involved in training and implementing this model. Instead, using ONNX with INT8 quantization should be sufficient to provide the memory efficiency, speed, and multicore support needed for our application. With these optimizations, we expect to achieve less than 25% of the original memory usage, enough inference speed to run the model in under one second, and only a minimal, likely unnoticeable, drop in accuracy [16],[24],[42]. If our system eventually requires more performance, we can still transition to NCNN with INT8 quantization as a future upgrade path.

Dynamic vs Static INT8 Quantization of Object Detection Models

When applying 8-bit quantization to an object detection model, it's important to understand that there are two main methods for doing this. The first method is called dynamic INT8 quantization and the second method is called static INT8 quantization. Both methods convert the model's weights and activations from 32-bit floating point values to 8-bit integers [42]. The difference is how and when they do this. Choosing the best method depends on the developer's needs, and where they are in the design process.

With dynamic quantization, only the model's weights are converted to 8-bit integers ahead of time. Activations stay as 32-bit floating point values and their ranges are calculated at runtime. This makes setup much simpler, usually requiring just a single Python command to convert the model's weights to 8-bit integers. Since no calibration data is needed, this method is especially useful during the prototyping stage. Dynamic quantization typically boosts model speed by 2-3x, reduces RAM usage, and still maintains accuracy that's very close to the original FP32 version when run on edge devices [23],[42].

For static INT8 quantization, it converts both weights and activations to 8-bit integers before inference. This is done using something called calibration, which is the only way to quantize activations ahead of runtime. Calibration involves passing a set of sample data through the model to help it learn the typical range of activation values for each layer. These minimum and maximum values are then used to define factors that allow the model to run while only using integer operations. While this adds an extra step and increases complexity, it also gives the best performance. However, it relies on using good calibration data, otherwise accuracy can decrease. Static quantization is best suited for final deployment when the model needs to be as fast and efficient as possible. It can improve inference speed by 2-4x, reduce model size and RAM usage by around 75%, lower latency, and greatly improve power efficiency [12],[23],[27],[42].

While it may be more complicated to implement, static quantization is clearly the best option for models that need high speed, low RAM usage, and minimal CPU load [27],[42]. For ease of development, it's often better to start with dynamic quantization during the prototype stage, since the model is likely to change frequently. If the dynamically quantized model performs well and the Raspberry Pi can run it consistently without issues, then it may be perfectly acceptable to stick with dynamic quantization without needing further optimization [42].

Table 3.3.4: *Dynamic vs Static Quantization*

Method	Speed	Latency	Accuracy	Setup	RAM	CPU	Overall
Dynamic IN8	2-3x faster	200-250 ms	Less than 1% drop	Good	Good	Fair	Good
Static INT8	2-4x faster	125-167 ms	Less than 1-2% drop	Fair	Excellent	Excellent	Excellent

3.3.2 Modern Mobile Development IDEs and Frameworks

As part of the pothole detection and tracking project, there is a mobile application that communicates, via Bluetooth, with the MCU responsible for controlling the system. The app allows users to review and verify detections made by the system on a user-friendly map. This requires a development environment that can use both Android and iOS mobile platforms, perform reliably, and allows for quick interface development. To find the best option, we reviewed several well-known development tools and frameworks.

The options reviewed include Android Studio, Xcode, Flutter, React Native, and .NET MAUI. Each of these tools offers several different features that affect development speed, compatibility, system requirements, and user interface consistency.

Android Studio

Android Studio is the main development environment for building Android apps using Kotlin or Java. It is supported by Google and gives a wide range of tools for debugging, interface design, and system profiling. Müller [35] found that Android Studio's Gradle build system is flexible and effective for managing complex Android applications.

Ivanović [33] analyzed several real-world Android applications and observed that performance bottlenecks and power consumption issues are often tied to common design patterns used in the Android Studio environment. While its integrated tools are comprehensive, the study suggests that code optimization within the Gradle structure is essential to avoid resource inefficiencies.

However, Android Studio can be demanding on computer resources. Developers using mid-range machines may experience slower performance, particularly when dealing with large projects. The Gradle build system, while capable, may slow down rapid development because of long compile times. Android Studio is best used for projects focused entirely on Android rather than cross-platform solutions.

Xcode

Xcode serves as Apple's standard development environment for building applications tailored to their various platforms, including iOS, macOS, and watchOS. It provides comprehensive support for both Swift and Objective-C and comes equipped with a suite of essential tools for debugging, designing user interfaces, and testing. A study done by Tan [37] indicated that Xcode's integrated intelligent coding tools, such as code completion and navigation, significantly enhance developer efficiency, particularly when working with newer Swift projects. This comprehensive and integrated environment streamlines the development workflow, enabling developers to build and refine their applications effectively.

However, Xcode operates exclusively on macOS. This poses a challenge for development teams working across multiple operating systems. Managing larger applications within Xcode can be complex, partly due to the way it handles visual tools and interface files. Despite its limitations, Xcode remains the preferred development environment for Apple platforms due to its integration with Apple's native SDKs and the App Store deployment [37]. This allows developers to take advantage of the latest platform features and streamline the release process.

Flutter

Flutter is Google's UI toolkit designed to create applications for Android, iOS, web, and desktop using the Dart codebase. It is consistent for rendering and high performance across platforms, along with powerful tools for building visually rich and responsive user interfaces. A study by Lovrić [36] showed Flutter's reliable cross-platform performance and how features like hot reload significantly boost development speed. Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM).

A study by Lee found that Flutter outperformed Kotlin and React Native in areas like UI rendering and response time [34]. And while Abu Zahra and Zein noted that test automation frameworks work well with Flutter, they did observe some minor differences in test behavior across platforms [31]. Some limitations present while using Flutter include the fact that apps can be larger in file size, and the support for web and desktop platforms is still under development. Even so, Flutter remains well-suited for cross-platform development, especially when code reuse and interface consistency are high priorities.

React Native

React Native is a framework developed by Meta that allows mobile development using JavaScript and the React codebase. It connects JavaScript code to native components, which can help improve performance compared to web-based frameworks. In a study done by Biørn-Hansen [32], he observed that React Native is useful for developers who already know JavaScript, allowing for faster and easier app development.

Uddin [38] performed a study of performance overhead in cross-platform tools and observed that React Native often introduces high-load delays due to its JavaScript-to-native bridge. Abu Zahra and Zein [31] also observed test inconsistencies between platforms when using automated test suites with React Native.

The main issues with React Native are related to performance in resource-intensive tasks and potential delays caused by its JavaScript-to-native bridge. Creating consistent behavior across platforms may also require writing platform-specific code, which can reduce some of the benefits of shared development.

.NET MAUI

.NET Multi-platform App UI (MAUI) is Microsoft's toolkit designated for building applications across platforms including Android, iOS, Windows, and macOS. It provides support for the C# programming language and builds on the foundation of Xamarin.Forms. The framework is integrated into Visual Studio and is packaged as a suite of tools for designing user interfaces, debugging, and managing platform-specific configurations. A study by Abdullwahed and Tran [35] observed that .NET MAUI's unified project structure and integrated development tools help streamline both the creation and maintenance of applications across platforms by simplifying cross-platform workflows within the .NET ecosystem.

One of .NET MAUI's key advantages is its project structure, which allows shared resources, styles, and logic to be managed more efficiently. Developers familiar with the .NET environment benefit from seamless access to .NET libraries, Azure services, and Blazor integration. However, as a relatively new framework, .NET MAUI is still under development. While .NET MAUI continues to be developed and offers a strong option for teams already working within the .NET ecosystem, projects that demand high performance or deep integration with device-specific features may still be better suited to native development tools.

Table 3.3.5 Comparative Evaluation of Mobile Development IDEs

IDE/ Framework	Cross Platform Support	Language Used	Ease of Use	GPS Integration	Database Integration	Performance
Flutter	Yes	Dart	High	Easy via plugins	Good (Firebase, SQLite, Hive)	High
Android Studio	No (Android only)	Kotlin/Java	Medium	Full native control	Flexible (Room, Firebase)	High
Xcode	No (iOS only)	Swift/Obj-C	Medium	Full native control	Comprehen sive (Core, Firebase)	High
React Native	Yes	JavaScript	High	Easy via libraries	Good (SQLite, Firebase)	Moderate
.NET MAUI	Yes	C#	Medium	Unified API abstraction	Good (SQLite, Firebase)	Moderate

IDE Selection

Based on the requirements of this project, React Native was selected as the primary framework for mobile development. It enables deployment to both Android and iOS using a single JavaScript and TypeScript based codebase, which significantly reduces development effort and ensures interface consistency across platforms. This cross-platform capability aligns well with the project's goal of maintaining identical behavior on both operating systems without requiring multiple separate implementations.

React Native provides robust support for BLE communication, GPS integration, and remote database interaction all of which are central to this application. Location services, cloud connectivity, and secure user authentication can be integrated through well-supported libraries such as react-native-ble-plx, react-native-maps, and Axios for backend communication. These tools allow the application to process real-time SPD packet data from the MSP430, synchronize pothole records with the MongoDB cloud database, and associate each detection with accurate GPS coordinates, all without needing low-level platform-specific code.

The framework also includes flexible tools for building responsive user interfaces, managing state, and rendering interactive maps using Apple Maps on iOS. React Native's performance characteristics, combined with its extensive ecosystem of open-source modules, make it well-suited for handling background BLE listeners, offline caching, and dynamic UI updates required for real-time pothole visualization. Its large community and mature documentation further simplify debugging, integration, and long-term maintainability.

Each of the alternate tools reviewed has its strengths and limitations. Android Studio is highly capable for Android-specific development but creates fragmentation when building for multiple platforms. Xcode remains essential for iOS deployment but is restricted to macOS systems and is not optimized for cross-platform workflows. Flutter offers strong UI performance but introduces additional overhead because the team would need to learn Dart and modify existing BLE and GPS logic. .NET MAUI provides a unified C# environment but is still maturing and lacks the plugins and ecosystem depth needed for this project's specialized sensor integration.

React Native ultimately provides a practical and efficient solution for the mobile application. Its cross-platform support, extensive library ecosystem, and strong performance characteristics align well with the technical needs of the pothole detection system. These advantages make it the most suitable choice for implementing the mobile portion of this project, ensuring both reliability and ease of future expansion.

3.3.3 Mobile Application Architectures

In evaluating options for implementing the mobile application of the project's system, it was also important to research all possible application architectures. Beyond the development tools themselves, we had to decide whether to deploy the application as a native app, web app, cross-platform app or hybrid app.

Each option offers different trade-offs in terms of device access, offline capabilities, performance, and application maintenance. Since the application needs to process GPS data, communicate with hardware through Bluetooth, and display real-time location on a map, it was essential to select a solution that supports responsive performance and consistent behavior across devices.

Web Apps (Including Progressive Web Apps)

Web applications, particularly Progressive Web Apps (PWAs), are built to run within a web browser using technologies such as HTML5, JavaScript, and

CSS. PWAs use special tools built into modern web browsers to do more than just show websites. These tools let the application work even when there's no internet, save information on the device, and use phone features like GPS to find your location. PWAs are being considered more consistently for cross-platform applications due to their simplified deployment, reduced overhead, and ability to run on multiple platforms from a single codebase [43].

Unlike native applications, PWAs are accessed directly through a URL, eliminating the need for distribution through app stores. This approach allows for a shorter development life cycle and easier maintenance. This also allows for lower overhead when frequent updates or wide accessibility are necessary. Their relatively small footprint reduces storage demands on the user's device. In a study by Lingolu and Dobbala [44], PWAs can enhance user engagement and simplify version management—qualities that are particularly valuable in civic and infrastructure-related applications where accessibility and update timeliness are essential.

Despite these benefits, PWAs still have many drawbacks when used in applications that require high-accuracy GPS, consistent hardware access, or low power consumption. Studies have shown that the real-time location accuracy of web apps can vary between browsers and devices, often falling short compared to native apps [44]. In addition, PWAs generally use more CPU and memory, which can lead to increased battery drain. In a study by Horn, he demonstrated that native applications consistently outperformed web apps in both power and system resource usage [45].

Although IndexedDB supports offline storage, it is limited by how much space browsers allow, and doesn't support advanced features like spatial queries, which are often needed for complex government data tasks. Additionally, web apps are restricted from accessing more advanced device functions, including background location tracking and combining data from multiple onboard sensors, like the GPS, accelerometer, and gyroscope, to improve motion or location accuracy. These capabilities are essential for real-time data collection and infrastructure monitoring.

For Department of Transportation (DOT) efforts like pothole detection and infrastructure mapping, web apps may work well for administrative dashboards, citizen reporting portals, or public-facing tools that need simple interactivity and easy access. However, their limitations in GPS precision, background data logging, and offline performance make them a poor fit for fieldwork. While web apps can complement native systems, they are generally not recommended as the primary platform for mission-critical tasks involving sensors and real-time data capture in transportation operations.

Native Apps

Native applications are developed for a specific operating system using languages tailored to that platform—such as Kotlin or Java for Android, and Swift or Objective-C for iOS. Once built, the app is compiled into code that runs directly on the device, giving it fast performance and full access to hardware features and built-in system functions. This results in better performance, more consistent behavior, and tighter integration with device capabilities.

In GPS-based government applications, native apps provide several important benefits. They deliver more accurate and reliable location data by using high-frequency GPS updates and platform-supported features that combine sensor data for better tracking. Horn [45] found that native apps consistently outperformed web apps in both power efficiency and system resource use. This makes a major difference for field workers who depend on their devices to last through long shifts without needing to recharge.

When it comes to storing data, native apps can use powerful offline databases like SQLite or Spatialite. These systems support spatial queries and allow the app to function fully, even without internet access. Data can be synced automatically once the device reconnects, making them well-suited for remote areas or signal-dead zones. Native apps also support background tasks, push notifications, and built-in security tools, like biometric login, which are useful in protecting sensitive data and keeping users informed.

From a transportation standpoint, native apps match the practical needs of Department of Transportation (DOT) operations. Field crews can use them on rugged devices to record pothole locations, take photos, and upload information directly to GIS databases. The high accuracy of native GPS tools ensures that location data is precise, which is important for scheduling maintenance, planning routes, and prioritizing repairs. Offline support adds an increased reliability by ensuring that data isn't lost when the network connection is weak or unavailable.

Despite these strengths, native development has its drawbacks. Creating and maintaining separate apps for Android and iOS takes more time, resources, and development skills. Once updated, the application must go through application store review processes, which, depending on the application store platform, can be a lengthy process—especially for a government application, since they have more stringent requirements. However, tools like Flutter now make it possible to write code once and deploy it as a native app on both platforms. This helps balance the performance of native apps with the efficiency of cross-platform development.

In the end, native apps are the best choice for government projects that require high GPS accuracy, secure data handling, offline reliability, and full hardware access. Their features make them especially useful in tasks like pothole detection, road inspections, and infrastructure monitoring—where data quality and system dependability are key. While the development effort may be higher, the long-term benefits are often worth the investment, especially for field operations in the public sector.

Cross-Platform Apps

Cross-platform applications are built using tools like Flutter, React Native, and Xamarin. These tools allow developers to write most of the code once and use it on both Android and iOS. This shared approach helps save development time, and is useful for applications that need to be available on multiple platforms. These frameworks also give access to important device features like GPS, the camera, and local storage, making them more capable than basic web-based apps. Biørn-Hansen [40] notes that cross-platform development can significantly speed up deployment without causing major drops in performance.

Cross-platform frameworks create native-like experiences by using built-in components from the operating system, which may lead to better performance and smoother user interfaces. This makes them a strong option for applications that need to collect data, track GPS locations, or store information offline.

One limitation of cross-platform applications is the lack of support for advanced features. To implement these features, developers need to write custom native code. This can make the application more difficult to maintain as it becomes more complex. Performance of cross-platform applications may also suffer in applications with complex animations, graphics-heavy interfaces, or heavy background activity. Since cross-platform applications don't have access to the deep system integration and up-to-date protection features, their security is often as strong as other application implementation forms.

Overall, cross-platform frameworks offer a solid balance between cost, performance, and device access. They are a good fit for projects that need to work on both Android and iOS, especially when the goal is to keep development time and costs low without giving up essential mobile features.

Hybrid Apps

Hybrid apps are made using common web technologies like HTML, CSS, and JavaScript, and are placed inside a native wrapper using tools such as Apache Cordova or Ionic. This allows them to run on both Android and iOS

devices while sharing a single codebase. Users can download them through app stores just like regular mobile apps, but the experience often feels more like using a webpage inside an app shell.

These apps are typically used for projects that don't need deep interaction with the phone's hardware. While hybrid apps can connect to some native features like the camera, GPS, or file storage, the level of access is usually limited. As a result, hybrid applications may not perform as well as native or cross-platform apps, especially when dealing with tasks that require real-time data processing or background activity. This can be a serious drawback for systems that depend on location tracking, sensor data, or offline logging, such as those used in transportation and public infrastructure reporting.

One of the main reasons teams choose hybrid development is cost. Since most of the code can be reused for both platforms, development is faster and more affordable, which is useful for smaller projects or simple applications. Some applications that display static data, company information, or collect basic form data can be built and deployed quickly, without the overhead of full native development.

However, hybrid apps are not a good fit for more demanding tasks. They often struggle with performance, and updates to device operating systems may cause compatibility issues that require frequent plugin maintenance. This makes hybrid solutions less reliable for government or field-service applications where accurate data collection and uninterrupted functionality are critical. As Lingolu and Dobbala [44] explain, hybrid apps offer a budget-friendly path for mobile development, but they are better reserved for lightweight use cases that do not depend heavily on hardware integration or speed.

Table 3.3.6 *Core Mobile Application Types*

Application Type	Definition	Common Technologies	Key Use Cases
Native App	Built for a specific OS using platform-native languages (e.g., Swift, Kotlin).	Swift, Kotlin, Objective-C, Java	High-performance apps needing full device access
Cross-Platform App	Single codebase compiled into native binaries for multiple platforms.	Flutter (Dart), React Native (JS), .NET MAUI (C#)	Apps for both iOS and Android with shared business logic
Hybrid App	Web technologies wrapped in a native shell (e.g., using Cordova or Ionic).	HTML, CSS, JavaScript + Cordova/Ionic	Quick deployment apps with moderate device access

Web App	Runs in a browser, not installed, relies on internet and browser capabilities.	HTML, CSS, JavaScript	Public info portals, basic tools, mobile-friendly websites
Progressive Web App (PWA)	Web app enhanced with offline support, push notifications, and installability.	HTML5, JS, Service Workers, Web Manifests	Offline-capable public service tools and interactive sites

Application Architecture Selection

Based on the requirements of this project, a cross-platform mobile application architecture was selected for implementation. This approach allows deployment to both Android and iOS devices from a single codebase using React Native, significantly reducing development effort while ensuring consistent functionality across both platforms.

Cross-platform architecture provides the essential device-access capabilities required for this system. React Native supports Bluetooth Low Energy communication with the MSP430 subsystem, rendering of real-time GPS coordinates on native map components, secure local data storage for offline operation, and seamless communication with the cloud backend. These capabilities enable reliable pothole detection handling, BLE data reception, and location-aware visualization without the need for separate platform-specific implementations.

The architecture also supports background processing and modern user interface development. React Native's ecosystem allows the application to process BLE packets from the MSP430, display synchronized GPS coordinates from the onboard MTK3339 module, and upload detection records to the cloud whenever connectivity becomes available. Additionally, cross-platform frameworks such as React Native compile down to native components and operate outside a browser, offering stronger security and more predictable performance than web-based or hybrid applications that rely on embedded web views. The development team's familiarity with JavaScript, TypeScript, and React Native accelerated integration with the backend and reduced overall implementation complexity.

Each of the mobile architecture options reviewed has strengths and weaknesses. Native Android and iOS applications provide the highest performance and full access to hardware features, but require separate development paths, effectively doubling the engineering workload. Web applications offer simple deployment but lack BLE support, cannot reliably

access mobile GPS hardware, and introduce additional security risks due to browser-based execution. Hybrid applications allow cross-platform deployment, but suffer from performance limitations and inconsistent hardware access due to their reliance on web technologies.

Cross-platform architecture provides an optimal solution for our project. It enables deployment across multiple mobile platforms from a shared codebase, supports reliable access to BLE and mapping hardware required for pothole detection, offers strong performance for real-time data handling, improves security by avoiding web-based execution environments, and aligns with the development team's expertise. These factors make React Native the most practical choice for implementing the mobile component of our pothole detection system.

3.3.4 Database Architectures for Mobile Applications

MongoDB

MongoDB is a document-based NoSQL database developed to support flexibility and scalability in large-scale, data-diverse systems. It is available in two main editions: a free Community Edition and a feature-rich Enterprise Edition. MongoDB stores data in BSON format, which allows for semi-structured, JSON-like document storage. This structure is highly beneficial in field applications where data can vary across records, such as when storing pothole reports that may or may not include images, GPS metadata, sensor readings, or manual annotations.

In addition to its schema flexibility, MongoDB supports geospatial indexing through 2d and 2d-sphere indexes, making it capable of executing complex spatial queries. This is especially relevant in mapping pothole locations or querying field events within a geographic radius. Its native replication and sharding mechanisms support fault tolerance and horizontal scalability, ensuring high availability and consistent performance across distributed networks. For offline resilience, lightweight MongoDB instances can be deployed on edge devices, capturing data locally and synchronizing with central servers when a connection is restored.

However, MongoDB has limitations. It operates primarily with eventual consistency, which may lead to synchronization challenges in scenarios requiring strict transactional guarantees. Support for complex JOIN operations is limited compared to traditional RDBMS solutions. GridFS, MongoDB's mechanism for storing large binary files such as images, requires additional configuration and resources. Furthermore, MongoDB's use of the Server Side Public License

(SSPL) introduces legal and compliance concerns for agencies bound by open standards or procurement restrictions [46].

PostgreSQL

PostgreSQL is a robust, open-source relational database system known for its adherence to SQL standards, ACID-compliant transactions, and extensibility. For projects involving government oversight and infrastructure monitoring, PostgreSQL offers a structured and reliable solution for data integrity and regulatory compliance. One of its standout capabilities is PostGIS, an extension that brings a suite of spatial functions, such as coordinate transformations, distance computations, and polygon based mapping capabilities well-suited for applications that depend on precise geolocation and route tracking [47].

Data stored in PostgreSQL is easily normalized, and the relational model allows for clearly defined constraints and dependencies, reducing data redundancy and improving integrity. Its transactional guarantees ensure that updates to geographic and photographic data are preserved without risk of loss or duplication. PostgreSQL also allows the integration of multimedia content either directly via BLOBs or indirectly through URI references to cloud-based storage systems.

PostgreSQL, like any other software, has its drawbacks. To support replication and partitioning, scaling write-heavy workloads horizontally requires careful architectural planning. Unlike MongoDB or SQLite, PostgreSQL does not natively support embedded deployments, which complicates its use in offline mobile environments. In such cases, a synchronization layer must be added between the device and the server, increasing development complexity.

Oracle Database

Oracle Database is a leading enterprise-grade RDBMS that has long been used in government, finance, and defense applications where security, availability, and data fidelity are paramount. Its rich feature set includes support for Oracle Spatial and Graph, an integrated suite of spatial functions, and tools for geolocation, mapping, and routing. This functionality allows the database to manage extensive geospatial data with a high degree of accuracy and performance [48].

Oracle's architecture supports Real Application Clusters (RAC), Data Guard, and automatic failover, providing unmatched redundancy and uptime. These capabilities make it an excellent candidate for centralized data storage in

critical infrastructure systems. Moreover, it supports multimedia data handling via BFILE references and SecureFiles for efficient image storage and retrieval. Advanced access control, auditing, and encryption further reinforce Oracle's suitability for projects involving sensitive civic data.

However, Oracle Database is resource-intensive and requires specialized personnel for configuration and administration. Its licensing costs are among the highest in the industry, making it a practical choice only for projects with significant budget allocation and long-term maintenance planning. Oracle also lacks a lightweight deployment model, making it impractical for edge or mobile applications where embedded capabilities are required.

Microsoft SQL Server

Microsoft SQL Server is a relational database management system developed by Microsoft and widely adopted in government, enterprise, and institutional IT environments. It provides comprehensive support for structured data, transactional processing, and business intelligence, while offering enterprise-level features such as role-based security, data auditing, and automatic backup. SQL Server's robust architecture and long-term vendor support make it a strong option for centralized systems requiring stability and compliance.

One of SQL Server's significant capabilities is its integration of geospatial functions through SQL Server Spatial. This allows the database to store, index, and query geographic and geometric data types natively, enabling it to handle GIS-related workloads, including mapping and proximity analysis, which are central to pothole detection and infrastructure mapping tasks. It also supports rich data types, including XML, JSON, and BLOBs, allowing the integration of photos and unstructured sensor data into centralized workflows.

SQL Server can scale vertically on powerful hardware or be deployed in distributed, high-availability environments using Always On Availability Groups. Its integration with tools like Microsoft Power BI and ArcGIS Server further enhances its usability in government data pipelines, particularly where spatial data visualization and reporting are important.

Despite its strengths, SQL Server is a commercial product with licensing costs that may be prohibitive to smaller projects or departments. It is also more tightly integrated with the Windows Server ecosystem, which may be limiting in heterogeneous environments where Linux-based deployments are preferred. Additionally, while SQL Server supports many enterprise needs, it is not suitable

for edge deployments or embedded mobile applications, and is best positioned as a centralized backend.

SQLite

SQLite is a lightweight, embedded relational database widely used in mobile and edge-computing environments. Its minimal setup, file-based storage, and ACID-compliance make it ideal for scenarios where reliable data logging is needed without requiring a continuous network connection. In field applications such as pothole detection, SQLite can store text, GPS coordinates, timestamps, and even images as BLOBs, enabling a comprehensive record to be captured offline and transmitted later [49].

One of SQLite's greatest advantages is its simplicity. With no need for a server or user authentication, it integrates directly into mobile applications and provides immediate access to the data layer. It is also supported by many programming languages and mobile SDKs, allowing for consistent deployment across devices. For basic geospatial capability, extensions like Spatialite can be used to enhance SQLite with spatial data functions, though with more limited coverage compared to full GIS systems.

Despite these benefits, SQLite does not support concurrent writes effectively, which limits its use in high-throughput environments. It also lacks many enterprise features such as replication, clustering, and user-level security controls. While well-suited for edge use, SQLite is not intended as a primary data store for large-scale, multi-user systems.

Database Selection

Based on the requirements of this project, MongoDB was selected for database management. It provides a flexible, document-based NoSQL solution that supports the varied data types needed for pothole detection, including GPS coordinates, timestamps, images, and metadata, all within a single database system.

MongoDB includes native geospatial indexing capabilities through 2d and 2d-sphere indexes, which are essential for this application. These features enable complex spatial queries for mapping pothole locations and querying field events within geographic boundaries. The database can be easily integrated with cloud storage services and supports real-time data synchronization without requiring complex relational database structures.

Table 3.3.7 Comparative Table of Databases

Database	Type	Offline Capability	Geospatial Support	Image/Binary Support	Scalability	Licensing	Mobile Use
MongoDB	NoSQL	Yes	Yes (2d/2ds here)	Yes (GridFS)	Horizontal (Sharding)	Free /Paid	Yes
PostgreSQL	Relational (SQL)	Limited	Yes (PostGIS)	Yes (BLOBs/URI)	Vertical/Partitioned	Free	Limited
Oracle Database	Relational (SQL)	No	Yes (Spatial & Graph)	Yes (BFILE, SecureFiles)	Horizontal (RAC)	Paid	No
Microsoft SQL Server	Relational (SQL)	No	Yes (SQL Server Spatial)	Yes (BLOB, JSON)	Vertical/Clustered	Paid	No
SQLite	Embedded Relational (SQL)	Yes (Native)	Limited (via Spatialite)	Yes (BLOB)	None (Single File)	Free	Yes

The framework also offers excellent scalability and offline resilience options. MongoDB's lightweight instances can be deployed on edge devices for local data capture, automatically synchronizing with central servers when connectivity is restored. Most importantly, MongoDB Community Edition is completely free, making it cost-effective for this budget-conscious project. The team also has previous experience working with MongoDB, reducing development time and learning curve challenges.

Each of the database options reviewed has strengths and weaknesses. PostgreSQL offers strong ACID compliance and PostGIS spatial capabilities, but it lacks embedded deployment options. Oracle Database provides enterprise-grade features, but it has prohibitive licensing costs. Microsoft SQL Server integrates well with Windows environments, but it requires commercial licensing. SQLite is ideal for embedded use, but it lacks multi-user capabilities and advanced geospatial features.

MongoDB provides a practical and cost-effective solution for our project. It supports flexible document storage for varied pothole data, offers powerful

geospatial capabilities for location-based queries, operates effectively in both online and offline scenarios, and requires no licensing fees. Combined with the team's existing MongoDB experience, these factors make it the optimal choice for the database component of our pothole detection system.

3.3.5 Evaluation of GPS Frameworks for Mobile Application Integration

The mobile application design requires seamless integration of GPS capabilities for both real-time location tracking and historical route reconstruction (i.e., backtracking) to ensure reliability. The GPS framework must also consider compatibility with mapping services, resource usage, offline functionality, and the ability to interface with embedded hardware over Bluetooth. To ensure that the mobile application met the requirements for this project, we researched many GPS technologies, including those that are already available, such as Google, Leaflet, Waze, and other SDKs. A key consideration in this selection process is the ability to support persistent pins. Because multiple users and interfaces may access location data, it is necessary that pins representing important locations—such as pothole detections, remain visible and accessible across sessions. This section evaluates viable GPS service options for mobile platforms, referencing software development studies and field applications to guide design decisions.

Native GPS APIs

Both Android and iOS offer native APIs that provide direct access to the device's GPS hardware. Android uses the Fused Location Provider (FLP), which intelligently blends data from GPS, Wi-Fi, and cellular networks to maintain accurate positioning while preserving battery life. On iOS, the CoreLocation framework supports similar functionality through assisted GPS and motion sensors. As noted in [51], native APIs generally provide the best results for applications needing precise and continuous location updates, especially outdoors.

While the literature highlights that native APIs are well suited for logging position over time and enabling backtracking when combined with timestamps, this project instead aligns events using the external GPS module and MCU-side timekeeping. Similar synchronization strategies are discussed in [56], but here they are implemented within the embedded hardware pipeline rather than through direct phone-to-MCU time transfer.

One advantage of native GPS APIs is their high precision and responsiveness for user-centric location features. Although these APIs do not

provide visual pins or map overlays directly, developers can integrate them with external mapping libraries to create persistent markers and routes. Coordinates can be stored locally or remotely and reapplied on application launch to maintain continuity across sessions. When combined with visualization frameworks such as Mapbox, Leaflet, or Google Maps [58], native GPS APIs offer a flexible foundation for building location-aware interfaces. In this system, that role is complemented by React Native Maps, which leverages the same underlying native services while focusing on rendering and visualization rather than acting as the primary source of pothole detection coordinates

Google Maps SDK

The Google Maps SDK provides a comprehensive mapping interface, built-in GPS tracking, and integration with the Directions and Places APIs. It also supports drawing polylines that represent travel paths, making it easier to implement backtracking. The SDK ties into the Fused Location Provider, benefiting from Google's infrastructure for location accuracy.

As noted in [52], Google Maps' familiar UI and reliable tracking tools improve user engagement and reduce development time. Backtracking is well-supported through polylines, and time-stamped location points can be displayed with markers or animated paths. Time alignment for microcontroller logging can also be facilitated by sending a timestamp during the initial Bluetooth handshake, allowing for synchronized route playback.

This SDK's advantages include its strong visual features, well-maintained documentation, and real-time update capabilities. It supports persistent markers by allowing developers to store and later restore coordinate data, ensuring that important locations remain visible even after the session ends [59]. However, it incurs usage fees for applications that exceed the free tier and generally requires a stable internet connection to function fully.

Mapbox SDK

Mapbox offers a flexible mapping platform with offline support, customizable map styles, and robust GPS tracking. It uses native device sensors and allows drawing user paths using its route rendering features. Its ability to cache map tiles locally gives it an edge for applications in areas with unreliable connectivity.

As discussed in [53], Mapbox performs well in scenarios needing offline operation and stylized visualizations. Developers can log timestamped coordinates and plot them for backtracking, and the platform's support for vector

tiles enables smoother rendering of movement paths. Like other SDKs, initial time synchronization via Bluetooth ensures event consistency between mobile and embedded systems.

Mapbox works effectively in offline contexts and supports a high degree of design flexibility for the visual presentation of data. It also supports persistent markers through its annotation and data source layers. Developers can serialize pin positions and reload them using local or cloud-based storage, which is critical for maintaining location continuity across user sessions [60]. While it offers deep control, it may require more configuration for advanced services like address lookup or geocoding.

OpenStreetMap and MapLibre

OpenStreetMap (OSM), paired with SDKs like MapLibre or OSMDroid, allows developers to integrate open-source mapping into mobile apps. These tools can render maps, track user movement, and store travel history for backtracking. While they lack some proprietary features, they offer full control and low overhead.

As shown in [54], OSM-based tools are especially useful in constrained environments, where cost and offline access are important. Backtracking is supported through manual plotting of historical location points. Although they do not offer direct support for timestamp synchronization, implementing a one-time time exchange from the app to the MCU at Bluetooth connection remains effective and is simple to integrate.

OSM-based solutions are attractive due to their cost-free nature, offline readiness, and adaptability. Persistent pins can be implemented by saving and restoring marker coordinates using open libraries like MapLibre or OSMDroid, which support dynamic marker layers [61]. However, they may require developers to handle more of the routing and interface work manually, and the user interface may not be as polished as commercial offerings.

Leaflet

Leaflet is a lightweight, JavaScript-based, open-source library designed primarily for mobile and web map rendering. Although originally web-focused, it can be used in mobile applications through embedded web views or hybrid frameworks such as React Native or Capacitor. Leaflet leverages raster tile servers, including OpenStreetMap, and allows developers to implement markers, layers, and polylines for route visualization and backtracking.

Leaflet's open architecture supports full control over time-stamped route plotting. With proper implementation, it can log historical paths and replay routes using dynamic polyline updates. While Leaflet does not natively interact with the device's GPS sensor, it can ingest location data supplied by native APIs, making it flexible for cross-platform use. Bluetooth-based timestamp synchronization, similar to the method proposed in [56], can be implemented at the application layer.

Leaflet's strengths lie in its lightweight nature, strong developer community, and full compatibility with open-source mapping data. It supports persistent markers through manual storage and recovery of pin coordinates, enabling pins to be rendered consistently across sessions [62]. While Leaflet is effective for hybrid or web-based mobile applications, the reliance on WebView wrappers for mobile deployment may introduce limitations in performance and native integration when compared to fully native SDKs. However, the reliance on WebView wrappers for mobile deployment may introduce limitations in performance and native integration when compared to fully native SDKs.

React Native Maps Integration

React Native Maps is used in this project as the primary visualization layer for geographic data within the mobile application. The library embeds native mapping components such as Apple Maps on iOS into the React Native interface, allowing the application to render real-time pothole detections alongside the user's current location. Rather than generating GPS data itself, React Native Maps consumes coordinates produced by the vehicle-mounted GPS module and transmitted through the MSP430 and Raspberry Pi pipeline as part of the \$PD packets. These integer-formatted latitude and longitude values are converted into standard coordinate format and then plotted as pins on the map.

This integration provides a clear separation between data acquisition and visualization. The embedded subsystem is responsible for capturing accurate locations and timestamps for each pothole event, while React Native Maps focuses on presenting that information in an intuitive, map-based view. The framework supports dynamic updates, enabling new pothole markers to appear as detections are received, and allows the user's position obtained via the phone's native GPS APIs to be shown concurrently. This hybrid approach ensures that the application benefits from the precision of external GPS hardware for detections while still leveraging native phone capabilities for user context and navigation.

React Native Maps also simplifies the implementation of features that would otherwise require manual integration with platform-specific map SDKs. It supports customizable marker icons, adjustable zoom levels, and region focusing, allowing technicians, drivers, and administrators to quickly assess pothole density and navigate to areas of interest. Compared to directly integrating separate map solutions such as Mapbox, Leaflet, or Google Maps [58], React Native Maps offers a more streamlined workflow within the React Native ecosystem, reducing development overhead while maintaining the flexibility needed for future enhancements to the mapping interface.

Waze SDK and Intent-Based Navigation

Waze provides a Transport SDK for select partners that allows applications to trigger navigation in the Waze app. It does not offer map embedding or real-time location updates within the app itself. Developers can only send destination coordinates and receive basic feedback such as estimated time of arrival.

As reported in [55], this model is simple to implement, but limits the developer's ability to manage location tracking or backtracking. Since Waze does not expose its location stream, applications cannot visualize previous paths or associate time data with specific route segments. It is also incompatible with syncing timestamped data to an MCU.

Although easily integrated, the Waze SDK offers limited flexibility and is not suited for use cases that require in-app tracking, route visualization, or synchronization of movement data with embedded systems. Since it does not allow adding custom markers or accessing in-app location streams, it also lacks support for persistent pins, which makes it incompatible with the requirements of this project.

Table 3.3.8 GPS SDK Comparison

Technology	Real-time Tracking	Backtracking Support	Offline Support	Persistent Pins	Map Visualization	Cost
Native GPS APIs	Yes	Manual	Yes	Yes	No	Free
Google Maps SDK	Yes	Yes	Limited	Yes	Yes	Free with limits
Mapbox SDK	Yes	Yes	Yes	Yes	Yes	Free/Paid
OpenStreetMap	Yes	Yes	Yes	Yes	Yes	Free

Leaflet	Yes	Yes	No	Yes	Yes (via WebView)	Free
Waze SDK	No	No	No	No	No	Free
React Native Maps	Yes	Yes	Yes	Yes	Yes	Free

GPS Mapping Selection

Based on the requirements of this project, React Native Maps was selected for GPS mapping and visualization within our mobile application. This framework integrates seamlessly into the React Native cross-platform environment, enabling native map rendering through Apple Maps on iOS and Google Maps on Android. By embedding these native components directly into the application, React Native Maps provides a unified and responsive mapping experience without requiring a separate mapping application or custom platform-specific development.

React Native Maps delivers the essential mapping features required for this system. The library supports real-time rendering of MCU-provided GPS coordinates, custom marker placement for potholes, and dynamic map interactions such as zooming, panning, and region focusing. Most importantly, it allows the application to display persistent, customizable pins representing pothole detections, complete with associated metadata such as depth, area, severity, and confirmation status. These markers remain visible throughout user sessions by retrieving stored coordinates from the cloud database, ensuring continuity in pothole monitoring and administrative review.

The framework also provides sufficient flexibility for data visualization and integrates smoothly with the backend used in this project. React Native Maps supports color-coded icons to represent different pothole state such as unverified detections, confirmed hazards, and resolved repairs and incorporates device-side GPS only for displaying the user's current location. All detection coordinates originate from the MSP430's dedicated GPS module, and React Native Maps focuses solely on visualization. Its compatibility with the application's MongoDB backend makes it straightforward to retrieve stored detections and update markers dynamically as new \$PD packets are processed. Additionally, because React Native Maps operates natively within the existing React Native application, it reduces deployment complexity and eliminates the need for additional mapping engines or web-view-based rendering layers.

Each of the mapping solutions reviewed has its strengths and limitations. Google Maps SDK offers a polished interface and excellent performance but may introduce usage-based billing for high-volume applications. Mapbox provides strong customization and offline capabilities, though it requires more complex configuration and larger bundle sizes. Leaflet is lightweight and flexible, but relies on web views and lacks the fluidity and native feel required for real-time visualization in a mobile environment. Native GPS APIs provide accurate coordinate acquisition but offer no built-in mapping or visualization tools, requiring substantial additional development effort.

React Native Maps ultimately provides the most practical solution for this project. It integrates directly into the React Native application, supports robust and customizable pins essential for pothole tracking, leverages native map performance, and works cost-effectively with the system's external GPS pipeline. These advantages make React Native Maps the most suitable choice for the mapping component of the pothole detection system.

3.3.6 Optimization of Parallel Threads

Python Parallel Processing Methods

The computer vision portion of our design requires an extensive number of parts, not only working in real time, but also using a large amount of processing power. It is critical that the organization and optimization of the software are strictly designed to prevent frame loss, bottlenecks, and system crashes. Real-time systems such as the two cameras, decoding, the lightweight detection algorithm, and the ring buffer must be able to run uninterrupted. These systems have mere milliseconds to complete a cycle; otherwise, bottlenecks begin to form. Parallelism is critical for allowing each critical process to run independently and simultaneously. Python and C++ are the most-used languages for computer vision, with C++ offering better parallelism at the cost of much greater complexity. This program, on its own, is already complex enough to heavily favor Python. However, Python does introduce some limitations, and only a few good options exist for working around them.

Pure-Python Multi-Process Pipeline

The main issue with using Python is that its compiler uses a Global Interpreter Lock (GIL), which ensures that only one thread executes Python bytecode at a time within a single process. The GIL makes parallel multithreading impossible for CPU-bound tasks since only one thread can run at a time [64],[65]. To work around this, multiprocessing can be used instead to leverage multiple CPU cores for heavy computing tasks. This works by giving each process its own

GIL, preventing CPU-bound work from one process from interfering with another [67]. This effectively dedicates CPU cores to specific processes, as if they are running entirely separate programs with their own set of memory.

This allows for true parallelism of processes, but it also introduces overhead in data movement, memory, and complexity. Since each process now uses its own memory, unlike threads, which share memory, images being passed between processes require pickling and unpickling, which can add 1-5 milliseconds per transfer if not managed well [64],[65],[66]. Each process also has its own Python interpreter, usually consuming around 20-30 MB of RAM, which could push the total RAM usage beyond several hundred megabytes, depending on the situation [64],[66]. Starting a new process can also take tens to hundreds of milliseconds on ARM devices, so it is important to use them for long-term processes rather than spawning new ones repeatedly for short-term events [64],[65].

Python with C++ Extensions Pipeline

An alternative option is to add C++ extensions into the Python multiprocessing pipeline for intensive computing routines such as image filtering, infrared image detection, or object detection models. This would use the same multiprocessing design but allows the GIL to be released for selected routines as if they were running in native C++. This will allow parallel threads to run as needed while also allowing multiple processes to run in parallel. The benefit of doing this is that it allows the high-level software to remain in Python for ease of development while still being able to use the low-level advantages of C++. Some studies have shown that using tools like pybind11 can improve speeds up to 2-3 times in real workloads and up to 95 times in synthetic benchmarks when utilizing ARM NEON optimizations [64], [65]. More recent benchmarking efforts show modern binding frameworks can reduce overheads by 2.7-4.4 times that of pybind11 and Cython [66].

The biggest downside to using this method is the added complexity of setting up a cross-compilation environment on the Raspberry Pi, managing build scripts, and debugging mixed-language use. Another issue is binding overhead when functions are too small or called too frequently, which can offset performance gains. It is also important to use correct integration, since bypassing the GIL can lead to race conditions, segmentation faults, or deadlocks [68],[69].

Python with Numba JIT Acceleration Pipeline

Another possible option is to use Numba's Just-In-time (JIT) compilation to accelerate Python routines, bypassing the GIL for numerical code, while also

keeping everything in a single Python process. Numba is a JIT compiler for Python used to speed up numerical code by converting math functions into fast machine code using a Low-Level Virtual Machine [70]. Numba works by compiling Python functions into native machine code at runtime and, while setting “nogil=True”, releases the GIL, allowing multiple threads to execute heavy loops in parallel on different CPU cores [68],[69]. This would be applied to performance critical routines, like the lightweight laser detection algorithm, allowing for parallel multicore execution without leaving the Python process.

Advantages of using this method include: low overhead, sub-millisecond handoff between threads, true parallelism, and simplification for embedded system deployment. The main disadvantages of this method include: adding possibly hundreds of milliseconds to the startup of the program, not supporting all complex Python logic, slowdowns when certain loops are applied, and higher debugging complexity. Overall, this approach reduces process overhead and simplifies deployment, but it also comes at the cost of JIT startup time and some limitations in dynamic Python features [66],[68],[69],[70].

Table 3.3.9: Comparing Parallel Processing Methods for Python

Category	Pure-Python Multi-Process	Python with C++ Extensions	Python with Numba JIT
CPU Throughput	Fair	Excellent	Good-Excellent
Latency	Fair	Good	Excellent
Memory	Poor	Fair	Excellent
Startup	Fair	Poor	Fair
Code Complexity	Good	Poor	Good
Debugging	Excellent	Fair	Fair
Deployment Ease	Excellent	Poor	Good
Parallelism Type	Multi-process	Both	Multi-thread

Using Python with Numba JIT is the most balanced way to achieve the parallelism our software needs. It achieves CPU-bound throughput results almost as good as using C++ extensions, without having to worry about the coding complexity. Since it only uses multi-threading, the RAM usage is considerably lower compared to the other options, since using multi-processes requires separate memory for each process [66],[67]. This also removes the 1-5 millisecond latency of sending information from one process to another. While the debugging process may be harder than using the pure-python multi-process method, it makes up for it by having very similar code complexity while achieving

much better results for memory, latency, and CPU-bound throughput [68]. If needed, C++ extensions can still be added, but with the cost of adding more complexity to the program [63]. If needed, switching to an entirely C++ program, would be the overall best option but with the cost of a much more complex overall program.

3.4 Mechanical Part Selection

The success of the Pothole Detection System is reliant on both the stability and protection of sensing components, including a laser and a camera. In order for smooth detection, the mechanical housing and mounting design needs to withstand real world conditions such as vibrations, bumps, moisture, temperature changes, and dust, all while maintaining correct alignment during operation.

Sensor Alignment

Within image based sensing systems, improper calibration and misalignment of optical elements can severely affect performance. This includes skewed frames, defocusing, or a reduced field of view, all due to deviations in orientation or position. According to the military optical enclosure studies, optical misalignments as small as 0.1 mm can result in target drift and data invalidation in high resolution systems [75]. In a system like our project, where the camera and laser system are subject to vibrations and shock loads due to being mounted on the car, these constraints become more strict.

In order to resolve these challenges, the mechanical design of the housing unit must suppress six degrees of freedom. This ensures that, after being aligned during assembly, the optical elements resist shifts due to vibrations, driving forces, or stress of the housing material.

Vibration Isolation

There's been a fair amount of research documenting the effects of low-frequency, road-induced vibrations on mounted systems used for sensing and imaging. These vibrations are usually generated from potholes, uneven roadways, suspension dynamics, and speed variations, the last of which are the most intense, being within the 5–30 hz range [76].

This is important data to consider when creating the mounts, since a natural frequency that overlaps 5–30 HZ can cause resonance, leading to misalignment or calibration drift.

Environmental Sealing and Water-Proofing

Ingress Protection (IP) standards defined by IEC 60529 that are used to classify levels for sealing enclosures from both solids and liquids. IP65 is regarded as the minimum acceptable level for protecting outdoor electronics from water and dust. This rating ensures that there is complete protection from dust and a strong resistance from water splashes, making it suitable for our project, which could be subject to spray from vehicle tires, rain, or sprinklers. IP67 is a higher standard that allows for brief submersion in water. However, with our sensor being suspended 5 feet in the air, this standard is not necessary.

Summary

Adjustable and Tool-Free Mounting: The mounts must allow for easy and simple adjustment of sensor angle and height to accommodate different vehicle types. Adding tool free mechanisms enable quick installation and repositioning in the field without specialized equipment.

Ruggedness Against Vehicle Dynamics: Mounting systems must withstand vibrations, shocks, and motion caused by driving. Vibration isolating materials and design considerations that avoid resonance in the 5–30 Hz range are essential for stability.

Sealed Enclosures: Housings must meet at least IP65 standards to protect against dust, rain, and splash exposure. Use of gaskets, weatherproof cable ports, and venting membranes help maintain internal integrity and prevent moisture buildup.

3.4.1 Mechanical Mount and Housing Material Selection

Material Selection: Mounting System

The mounting arm of the pothole detection system must be mounted externally on a vehicle using suction cups, and must position the camera and laser housing at a cantilevered length. It is thus subject to dynamic loading, vibration, shock, and thermal cycling. Structural stiffness, low mass, corrosion resistance, and manufacturability were key considerations in material selection.

6061-T6 aluminum was selected due to its excellent strength-to-weight ratio, excellent corrosion resistance, machinability, and relatively low cost. It is a well-proven structural alloy in the automobile and aerospace industries and has sufficient stiffness to resist bending or deflection.

Carbon fiber composite was considered for its extremely high stiffness-to-weight ratio, but was eliminated due to its brittleness upon impact, field repair problems, and electromagnetic interference problems.

ABS plastic, though very inexpensive and easy to print, was eliminated for structural mounting parts due to extremely poor stiffness and insufficient UV and heat resistance.

Table 3.4.1: “6061-T6 Aluminum vs Carbon Fiber Composite vs ABS Plastic”

Material	Pros	Cons	Final Selection
6061-T6 Aluminum	Strong, Easy to Machine, Corrosion Resistant	Heavier than composites	Selected
Carbon Fiber Composite	Lightweight & Stiff	Brittle, expensive, difficult to modify	Rejected
ABS Plastic	Cheap, Printable	Weak Under heavy Load	Selected

Material Selection: Housing Enclosure

The enclosure for the laser and camera must possess high resistance to environmental exposure without optical transparency loss. The material must withstand vibration, exclude water and dust, be machinable or 3D printable, and provide for long-term operation.

Polycarbonate (PC) was used due to its high weatherability, hardness, and clarity of light transmission. Used extensively in headlamp lenses and safety glasses, PC maintains clarity and strength at a wide range of temperatures and is resistant to UV yellowing.

Acrylic (PMMA) was also in the running due to its optical properties and being less expensive, but was ruled out due to lower impact resistance and stress-cracking susceptibility. ABS plastic, while being sufficient for interior, non-structural components, lacked environmental durability and transparency to be acceptable on the housing window.

Table 3.4.2: “Polycarbonate vs Acrylic vs ABS Plastic”

Material	Pros	Cons	Final Selection
Polycarbonate	Strong, impact resistant, optically	More expensive than acrylic	Selected

	clear		
Acrylic (PMMA)	Optically clear, cheap	Brittle	Rejected
ABS Plastic	Impact resistant, printable	Poor optical clarity	Selected

Material Selection and Testing

Characteristics such as functionality, reliability, and durability for our project are reliant on the mechanical and environmental characteristics of materials used to construct the sensor housing and the mounting arm. These components are the basis of the detection system, mounting the laser and camera physically to endure a series of stresses from vibration, road contact, weather and long term exposure. Because pothole detection is based on reliable optical alignment, any deformation, warping, or cracking would create detection errors. So, the selection of proper materials is crucial in the design.

To address these challenges, we conducted a materials science assessment that considered mechanical strength and fatigue resistance. The application of 6061-T6 aluminum for the mounting system and polycarbonate (PC) for the enclosure was decided after a careful comparison of mechanical performance characteristics, including tensile strength, toughness, ductility, and stiffness.

3.4.2 Mechanical Properties and Behavior

Understanding of the mechanical response of the materials when under stress is required in ensuring the pothole detection equipment runs under a number of realistic stresses. Materials will be classified into three stress regimes of deformation: elastic, plastic, and brittle.

1) **Elastic Behavior** is reversible deformation, meaning material will return to its original shape when the loading is reversed. This behavior is a linear function of stress and strain, as given by Hooke's Law:

$$\sigma = E \cdot \epsilon \quad [81]$$

σ is stress, ϵ is strain, E is Young's Modulus

2) **Plastic Behavior** is when a material exceeds its yield point. The material becomes deformed permanently and does not revert to its original configuration,

even upon unloading. This is particularly relevant for crash events or vibration exposure occurring in a repetitive nature, where long term exposure would result in sensor misalignment.

3) **Brittle Behavior** is when materials fail with minimal deformation. Acrylic or glass have brittle failure, which can be dangerous in an environment with potential for impact and vibration.

4) **Ductility**, which is the ability of a material to be able to stretch or deform plastically before it breaks. High ductility allows parts to bend or absorb energy without cracking.

5) **Toughness**, which is the overall ability of a material to absorb energy to failure, as reflected in the area under a stress-strain curve. [78]

For this project, 6061-T6 Aluminum possesses good ductility, moderate toughness, and a high strength-to-weight ratio. It is well suited for use in the external mount, which must be resistant to bending with minimal added load to the vehicle. Polycarbonate (PC) has high impact resistance, high optical clarity, and stability with thermal cycling. It is well suited for the housing that protects the laser and camera components.

These properties ensure the mounting system can resist the 5–30 Hz road induced vibration range without entering resonance or structural fatigue, and the housing can resist environmental loading, such as road debris impacts, water exposure, or thermal change.

3.4.3 Tensile Testing of Prospective Materials

To ensure that our material choices were made from quantifiable data, we relied on common tensile test data from the ASTM E8 (Metal Tensile Test) and ASTM D638 (Plastic Tensile Test) methods. Tensile testing establishes a material's elastic limit, yield strength, ultimate tensile strength, elongation at break, and modulus of elasticity. All of these are extremely crucial in selecting materials that undergo real loading.

1) 6061-T6 Aluminum possesses the following mechanical properties [79]:

Young's Modulus: 69 GPa — extremely rigid for structural integrity.

Yield Strength: ~276 MPa — designates beginning of permanent deformation.

Ultimate Tensile Strength: ~310 MPa — maximum stress before failure.

Elongation at Break: ~10–17% — allows for moderate ductility before failure.

2) Polycarbonate (PC) possesses the following mechanical properties [79]:

Young's Modulus: 2.0–2.4 GPa — rigid enough for enclosure walls, but also has flexibility for impact.

Yield Strength: ~60–70 MPa — sufficient to support sealing and compression loads.

Ultimate Tensile Strength: ~65 MPa.

Elongation at Break: 60–100% — displays extreme ductility, thus shatter-resistance.

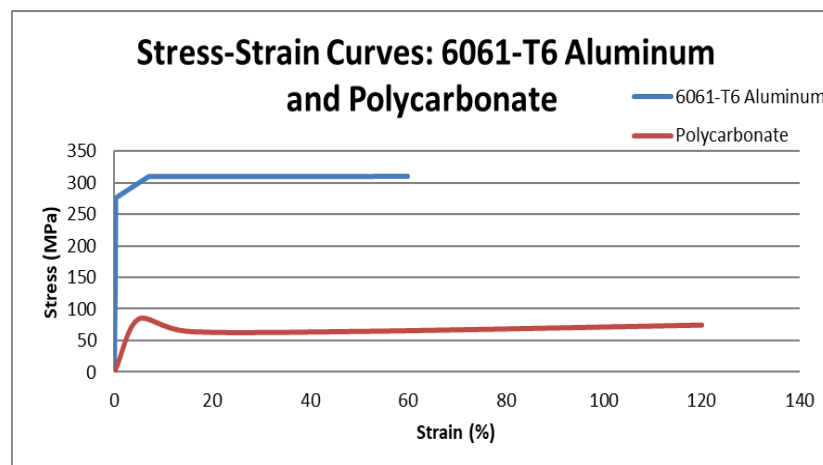


Figure 3.3: Stress-Strain Graph for 6061-T6 Aluminum and Polycarbonate

All these values cater to the system's need for structural rigidity, along with flexibility in defined areas. The aluminum ensures that the mount does not fail under load or fatigue due to road vibration, and the polycarbonate casing dissipates impacts without deforming or cracking.

Fatigue Resistance and Cyclic Loading

Because the system is on a moving vehicle with vibrational loading (5–30 Hz range), fatigue is a concern. Testing for fatigue resistance includes ASTM E466 (Metallic fatigue testing) and ASTM D7791 (Plastic tension fatigue testing) [80].

1) 6061-T6 Aluminum is fatigued at ~96 MPa at 500 million cycles and possesses good support for sustained vibration exposure.

- 2) Polycarbonate, not typically in load carrying, performs well for fatigue at low stress cyclic loading.

Impact Resistance and Fracture Toughness

Polycarbonate was selected for the enclosure window because it possesses excellent impact resistance and high fracture toughness. Unlike acrylic, when impacted, polycarbonate possesses the capacity to absorb a vast amount of energy before shattering. Impact testing includes ASTM D256 (Izod Impact Test) and ASTM D6110 (Charpy Impact Test).

Polycarbonate has the ability to withstand impacts of $>850 \text{ J/m}$ (Izod), making it an appropriate use in applications where debris or shock incidents can occur.

Compression and Shear Testing

Whereas tensile tests dictate design options, shear and compression properties are equally critical, especially for threaded joints, gaskets, and load transfer between modules.

- 1) ASTM D732 (Plastic shear strength): Polycarbonate shear strength $\sim 65 \text{ MPa}$.
- 2) ASTM E9 (Compression testing of metals): Aluminum compressive load can be withstood similar to tensile strength ($\sim 300 \text{ MPa}$).

3.4.4 Mechanical Design Calculations and Analysis

To guarantee that the mechanical components of the pothole detection system are reliable under real driving conditions, design calculations were conducted. Calculations included the structural and dynamic stability of the mounting arm and housing enclosure. Because the equipment is mounted and suspended off the back of a vehicle, it will have to undergo cantilevered loading, vibration, and temperature change without compromising alignment or structural integrity. We were able to validate our material selection, optimize size, and ensure that the design will operate safely within known tolerances through engineering analysis of loading, deflection, vibration, and thermal expansion.

Load and Deflection Analysis on Mounting Arm

The mounting arm supports the laser housing and camera at a fixed distance from the surface mass spring system of the car in a cantilevered beam configuration. The arm must accommodate both static weight and dynamic loads

caused by road-induced motion and vibration. To analyze the structural performance, we calculated the deflection of the arm. The applied load was the total weight of the housing system, estimated at 4.5 pounds (20 N). The arm was assumed to be 0.5 meters long and to be constructed out of 6061-T6 aluminum, a material selected for its strength-to-weight ratio. The moment of inertia (I) of the beam was also estimated at $1.2 \times 10^{-8} \text{ m}^4$ for a rectangular cross section. With Young's Modulus, (E) = 69 GPa, we employed the formula for the standard cantilever beam deflection:

$$\delta = \frac{WL^3}{3EI} \quad [81]$$

When using our values we got:

$$\delta = \frac{20 * (.5)^3}{3 * 69 * 10^9 * 1.2 * 10^{-8}} = 1.006 \text{ mm}$$

The calculated deflection of the tip was about 1.006 mm, which is considerably less than within acceptable limits for optical alignment. Considering the field of view and pixel size of the camera, this level of deflection results in very low angular misalignment, which does not affect performance. The calculation was done to ensure that the arm would be stable and stiff enough under static and dynamic loading without compromising fatigue or bending.

Natural Frequency of Mounting System

Another consideration was that the mounting system itself should not vibrate at frequencies encountered on normal roads. Most vehicle-induced vibrations occur between 5–30 Hz, so the natural frequency of the mounting system must be well above this to avoid resonant amplification, leading to structural fatigue or misalignment.

The natural frequency of a mass spring system is given by:

$$f_n = \frac{1}{2\pi} \sqrt{\frac{k}{m}} \quad [82]$$

Where k is stiffness of the arm and m is mounted gear mass. The stiffness k is found using the beam deflection formula:

$$k = \frac{W}{\delta} = \frac{20}{0.001006} = 19880 \text{ N/M} \quad [83]$$

Using a weight of 4.5 lbs or 2.04 kg for the housing and arm assembly:

$$f_n = \frac{1}{2\pi} \sqrt{\frac{19880}{2.04}} = 15.7 \text{ Hz}$$

This result shows that the natural frequency of the system is just within the critical range: an indication of risk of resonance. Because of this, features such as material optimization, rubber isolators, and damping bushings have been incorporated into the design to shift the natural frequency above 30 Hz.

Thermal Expansion of Optical Housing

Since the system is exposed to the outdoors, it must stay aligned and structurally sound even after daily thermal cycling. The housing's optical window, made of polycarbonate plastic, can thermally expand, which will have the potential to misalign the laser or place stresses on the mounting joints.

We used the formula for linear thermal expansion to estimate this:

$$\Delta L = \alpha * L_0 * \Delta T \quad [84]$$

Where: $\alpha = 65 \cdot 10^{-6}$ $L=100\text{mm}$, $\Delta T=45^\circ\text{C}$:

$$\Delta L = 65 \cdot 10^{-6} \cdot 100 \cdot 45 = 0.2925 \text{ mm}$$

The expansion was calculated to be less than 0.3 mm, which is not significant to the camera and laser alignment accuracy. Uniform expansion and dimensional stability of polycarbonate are useful for covers that have transparent coverings over housings that are subjected to varying temperatures. This is also in agreement with our use of flexible sealant compounds that can conform to such expansion without failure.

These calculations are important because they turn design ideas into something we can prove will work in the real world. By measuring things like deflection, stiffness, vibration response, and thermal expansion, we made sure the mechanical frame stays strong and aligned during actual use. The results helped us choose the right materials, and shaped the size of parts, the sealing strategy, and how we handled vibration. These choices improve the reliability of the system in outdoor environments and provide the confidence that it will perform safely over time.

3.4.5 Geometry Optimization for Structural Stiffness

The structural stiffness of the mounting arm is an important factor in maintaining proper alignment of the camera and laser throughout operation. Because the sensor housing is mounted in a cantilevered fashion off the rear of the vehicle, small deflections can cause significant angular deviation in the field of view. Misalignment could result in distorted scans, false positives, or detection failure, making geometric stiffness optimization an important design concept.

Cross-Section Design for Flexural Stiffness

In order to obtain deflection resistance from static and dynamic loading, a rectangular aluminium section was designed in the mounting arm. This shape gives an ideal relationship between low weight and high stiffness due to its high area moment of inertia.

The area moment of inertia for a solid rectangular section is:

$$I = \frac{bh^3}{12} \quad [83]$$

Where: b = section width & h = section height

For a hollow rectangular section, which was used in this design, the formula is:

$$I = \frac{bh^3 - b_i h_i^3}{12} \quad [83]$$

Where: b_i, h_i = inner width and height

This relationship shows that increasing the height of the cross-section exponentially raises bending stiffness, since the term in height is cubed. Because of this, the design was optimized with a taller, rather than wider, profile, allowing for stiffening without excessive material use or increased width.

This was balanced with constraints on profile size, weight limits, and manufacturability. A wider cross section would have provided greater torsional stiffness. However, it would've reduced stiffness per unit of weight. Our final designed maximized flexural stiffness while maintaining a lightweight form factor

Reinforcements and Adjustable Features

Many design enhancements were considered to preserve structural stiffness and ensure long term reliability. A telescoping extension arm was incorporated, allowing for on-site adjustment of arm length based on vehicle geometry. Once the desired extension is obtained, locking collars clamp the tube in place to restore full structural continuity. This feature helps to ensure that the arm remains rigid under various orientations, vibration frequencies, and loading conditions.

3.4.6 Mounting Configuration and Suction Cup Strategy

Five individual suction cups were incorporated in this design to balance the mount. There is a mechanical locking lever on each suction cup that, when pressed against a flat vehicle surface, creates and maintains a vacuum seal. The locking mechanism, when locked, provides strong adhesion without adhesives, magnets, or fasteners that are drilled into place.

While no physical testing has yet been conducted for this project, reference values for commercially-procured, high-strength suction systems suggest that each of the suction cups will resist approximately 100–120 N of pull-off force under ideal circumstances. With five cups being applied in concert, the theoretical holding strength is far greater than 500 N, or over 25 times the static weight of the mounted sensor system (approximately 20 N or 4.5 lbs). This wide margin of safety is required not only to accommodate vertical loading but to resist peel forces and off-axis torques that are experienced in daily driving.

For proper suction cup operation, surface conditions are paramount. For optimal performance, there should be:

- 1) Clean, smooth, non porous surfaces.
- 2) Minimum curvature across the mounting area.
- 3) Ambient temperatures within the cup's specified operating range.

In road usage, car surfaces should be clean and dry before mounting. Surface roughness will degrade suction capacity, so smooth surfaces are advised rather than rough surfaces. To maintain long term adhesion, suction status needs to be checked regularly and locking levers need to be engaged.

Dynamic Loading Scenarios and Theoretical Performance

Although the test has not yet been performed, the suction mount was selected based on its expected ability to withstand the following conditions:

- 1) Braking and Acceleration: Deceleration of the vehicle would be most likely to displace the center of mass of the sensor assembly. Five point design loads on the suction cups will ensure stability.
- 2) Road Vibration: Vibration within the 5–30 Hz frequency range, which is typical of normal driving, has the potential to loosen secured mounts. The mechanical locking aspect of the suction cups counters this with a high friction seal.
- 3) Wind Drag: At highway speeds, aerodynamic drag can generate considerable rearward force on the housing. Suction adhesion is in a direction perpendicular to the mounting surface, making it resistant to parallel drag forces.

Design Advantages Over Permanent Mounts

Compared to adhesive or mechanically mounted systems, this type of suction design has the following practical use advantages.

- Non-destructive: No screwing and drilling, and no vehicle permanent alteration.
- Reusable and adjustable: Capable of removal or relocation from one use to another.
- Platform versatile: Suitable for use on a wide variety of vehicles and configurations.

These benefits make it best for use in fleet tests, research studies, or mobile test facilities where modularity and serviceability are of great concern.

3.4.7 Failure Modes and Effect Analysis

Failure Modes and Effects Analysis (FMEA) is an engineering process to identify potential failure points in a system, evaluate the causes and consequences of the failures, and define mitigation strategies to reduce risk. (University of Cambridge) By analyzing each component of a mechanical assembly, FMEA allows engineers to anticipate what can fail, how severe the effect would be, how likely it would occur, and if it would be detectable before

causing major damage. FMEA is used throughout industries such as aerospace, automotive, and manufacturing, since it prevents design flaws, warranty issues, and ensures long term system reliability.

A FMEA was critical for our project because of the environmental exposure and real world vibration that the system experiences. Mounted externally on a moving vehicle, the mechanical assembly has to endure road-induced shock, moisture, debris, and temperature fluctuations. The laser and camera are subject to misalignment or damage, and their accuracy is crucial to the success of the detection algorithm. Because of this, the design team applied the FMEA approach to analyze each component, such as the mounting arm, housing enclosure, fasteners, and optical window, to determine how each could potentially fail in operation.

For each item, we identified the likely failure modes (cracking, loosening, or seal failure), as well as the causes (material degradation, vibration fatigue, improper assembly). We assessed the effect of each failure on the overall system, whether it would result in loss of alignment, water exposure, or system failure, and then recommended a method of detection for each case, such as visual inspections, vibration testing, or leak detection procedures. Each failure mode was then graded on three factors: severity (measures how severe the impact of the failure is), occurrence (measures how probable the failure will happen), and detection (measures how easily the failure could be detected before it does damage). These three figures are multiplied together to provide a Risk Priority Number (RPN). The larger the RPN, the more urgent the design revision, testing, or mitigation is required.

Table 3.4.3: “FMEA Table”

Component	Failure Mode	Cause	Effect	Severity	Occurrence	Detection	RPN
Mounting Arm	Fracture	Vibration / Impact	Sensor Misalignment/ Detachment	9	4	7	252
Housing Enclosure	Seal Failure	Gasket Degradation	Water Exposure	8	5	4	160

Fasteners	Loosening	Vibration	Component Shift	6	5	4	120
Optical Window	Cracking	Thermal Stress	Image Distortion	6	3	5	90
Laser Module Housing	Clamp Loosening	Vibration Fatigue	Laser misalignment	8	4	5	160

This FMEA will be revisited after prototype testing to reevaluate risk levels and effectiveness of mitigation strategies.

The completed analysis revealed the Housing Enclosure and Mounting Arm to be two of the most critical components, with the highest RPN values. This makes sense, as these components supply the structural integrity and environmental protection for the sensing hardware. Failure here would impact the alignment or ruin the electronics, potentially making the system unusable. As a result, we used design enhancements such as redundant seal mechanisms, superior fasteners with thread lockers, and material selection for high corrosion and fatigue strength. These mitigation strategies were taken directly from the FMEA results.

With a rigorous FMEA, we could reveal mechanical design weaknesses and supply justification for specific material and design choices. This task plays a role in achieving the project's reliability goals and in ensuring safe, accurate operation over time.

4. Standards and Design Constraints

4.1 Industry Standards

4.1.1 Electrical Hardware Standards

IPC-2221A

IPC-2221A was developed by the Association Connecting Electronics Industries and is a widely used standard that provides design requirements for

printed circuit boards (PCBs) and other forms of component mounting and interconnecting structures. It serves as a standard for making sure the electrical, mechanical, and environment performance of PCBs, focusing on aspects such as conductor spacing, trace width, and thermal management.

Some quantifiable recommendations from IPC-2221A include:

Conductor Spacing

- External Layers: 0.1mm for voltages greater than or equal to 30V.
- Internal Layers: 0.13 mm for voltages greater than or equal to 30V.

Current Carrying Capacity of Traces:

- A 0.25 mm wide external copper trace supports 0.5A with a 10 degrees Celsius temperature rise.
- For a 1A current, trace width should increase to about 0.5 mm under the same conditions.

By following IPC-2221A guidelines, designers can make sure that their PCBs will meet industry standards for safety and reliability.

4.1.2 Wired Communication Standards

The MSP430's USCI module supports SPI (Serial Peripheral Interface) communication in both master and slave configurations. SPI is a synchronous, full-duplex protocol, where data is exchanged simultaneously between two devices using a shared clock signal. The USCI module in SPI mode uses three key pins:

- UCLK - SPI clock (driven by master).
- SIMO - Slave In / Master Out (MOSI).
- SOMI - Slave Out / Master In (MISO).

The module supports many different configurations, including clock polarity (UCCKPL), clock phase (UCCKPH), MSB/LSB (UCMSB), and synchronous mode enable (UCSYNC). These parameters should match the settings of the communicating device. SPI baud rate can be set using prescaler registers (UCBRx), and data is transmitted/received via a single buffer register.

The interrupt can operate in interrupt-driven or polling mode. In interrupt mode, transmission and reception can be handled asynchronously using UCBxTXIFG and UCBxRXIFG flags. This allows efficient communication with very little CPU overhead. The SPI interface is ideal for applications that need fast, short-range communication, like sensors, memory devices, and other MCUs.

4.1.3 Bluetooth Communication Standards

The IEEE 802.15.1 standard originally established the low-level communication rules for Bluetooth, covering how devices connect and transmit data over short distances. It defines the physical and MAC (Media Access Control) layers of a wireless personal area network (WPAN) and was developed to allow multiple devices to communicate efficiently over the 2.4 GHz ISM band using techniques like frequency hopping spread spectrum (FHSS) and Gaussian frequency shift keying (GFSK). These methods were effective in limiting signal overlap and improving the stability of device connections.

This standard introduced a structure for short-range Bluetooth networks, where one device handles coordination and timing, while the others stay in sync with it. The MAC layer helps manage how each device takes its turn when sending data, which prevents overlapping transmissions. Though IEEE no longer actively maintains this standard (marked as withdrawn), its original structure still forms the basis for today's Bluetooth technologies.

Bluetooth 5.0, the version used in our project, is maintained by the Bluetooth Special Interest Group (SIG). It improves on the original design in several ways. Key enhancements include:

- Faster data rates up to 2 Mbps.
- Greater transmission range using coded PHY for improved signal quality.
- Support for larger packets (up to 255 bytes), which improves efficiency.
- Improved advertising capabilities for faster device discovery and pairing.

In the context of our project, Bluetooth 5.0 is used to send sensor and GPS data from the MSP430 microcontroller to a smartphone. Its improved range helps ensure stable communication in mobile environments like a moving vehicle, even when temporary obstacles block direct line-of-sight. The larger packet size and higher speed also reduce the number of transmissions needed, saving power—important for our battery-powered system.

Bluetooth 5.0 keeps a modular architecture with layers like L2CAP (Logical Link Control and Adaptation Protocol), ATT (Attribute Protocol), and GATT (Generic Attribute Profile). These help organize and manage data transmission, ensuring the phone and microcontroller can communicate efficiently and securely. It also includes stronger encryption and privacy protections, which are important for keeping transmitted data secure.

By combining the IEEE's foundational work with Bluetooth SIG's more recent improvements, our project benefits from a reliable and efficient communication standard that is widely supported by both embedded devices and smartphones.

4.1.4 Laser Safety Standards

When designing a device using a laser it is important to ensure the safety of anyone in the vicinity while the device is active. To do this we looked to the American National Standards Institute (ANSI) for their standards ANSI Z136.1 safe use of lasers and Z136.6 safe use of lasers outdoors. These standards emphasize the importance of MPE for safe laser operation.

MPE or maximum permissible exposure is the limit in power and/or time that a person can interact with a laser without adverse biological effects. For our design we are primarily concerned with ocular MPE as our laser has no risk of exceeding MPE for skin exposure. ANSI Z136.1 defines MPE for a continuous wave point source with a wavelength between 0.700 μm and 1.050 μm as $C_A \times 10^{-3} \text{ W/cm}^2$ with $C_A = 10^{2(\lambda - 0.700)}$, since our beam has a wavelength of 0.820 μm the MPE for our system is about $1.73 \times 10^{-3} \text{ W/cm}^2$.

Based on the MPE and divergence of the laser a nominal ocular hazard distance (NOHD) needs to be established during system operation with all personnel within the NOHD needing to wear protective goggles of OD3+ at 820nm. While there is an equation to calculate NOHD it is generally only useful for circular or elliptical beams while our beam is rectangular by the time it exits the laser housing. In situations like this it is generally considered best practice to calculate NOHD based on the axis of the beam with the least divergence as a worst case scenario, however our scenario is unique as one axis has extreme divergence whilst the other has virtually no divergence. Because of this we need to determine the NOHD for our laser manually. NOHD is defined as the distance from the laser where the irradiance is at or below MPE. As the MPE for our laser is $1.73 \times 10^{-3} \text{ W/cm}^2$ and the laser has a power of 200 mW it is considered unsafe up to an area of 115.61 cm^2 . Given that our line has a divergence of roughly 7.8 mrad vertically and about 1,571 mrad horizontally our laser is safe when it is about 0.67 cm thick and 172.72 cm or about 5.67 ft wide, with a fan angle of 90°

on our Powell lens that means it reaches this size about 2.84 ft from the exit aperture of the laser. From these calculations we found that our NOHD should be 2.84 ft in every direction from the exit point of the laser. As this is a prototype the NOHD was manually enforced by the group during testing and demonstration. For active deployment in a fleet a larger 5.67 ft x 5.67 ft enclosure or barrier would be needed to ensure public safety, possibly included in a more advanced version of the mount.

4.1.5 Mechanical Standards

Mechanical standards play an important role in ensuring safety within any engineering project involving physical structures, load bearing components, or environmental exposure. These standards developed by organizations such as the American Society of Mechanical Engineers (ASME), the International Organization for Standardization (ISO), and the American Society for Testing and Materials (ASTM), are pivotal for safe, repeatable, and reliable mechanical systems.

1. **Safety:** Safety is one of the most important drivers behind mechanical standards. These standards exist to ensure that mechanical components are able to withstand stresses, forces, fatigue, and shock without failure. Adhering to the proper safety margins and failure thresholds derived from these standards, helps to prevent and reduce any safety-related incidents within industries.

In our project, safety was addressed by referencing the **MIL-STD-810H**, a U.S. military standard that shows how to test equipment for durability, while subject to vibration and mechanical shock [77]. This standard was used when considering the type of material and thickness, vibration damping methods, and the support structure of our mount to ensure that it wouldn't fail when subjected to sudden braking or rough driving conditions. In the case of a collision or high impact vibration, our mount is built to remain both intact and secure to the vehicle, preventing damage to surrounding vehicles and pedestrians. This is especially important considering that our system is mounted externally and has direct interaction with motion.

2. **Interoperability:** Mechanical standards show common dimensions, tolerances, hole patterns, and thread sizes that allow parts from various manufacturers to fit and work together simultaneously. This makes it easier to add or switch out parts if needed, promoting flexibility.

Our design followed both standard fastener sizes and bracket mounting patterns based on **ASME B18.2.1** and **ANSI B1.1**, which ensured that replacement parts and components could be sourced without redesign. This

essentially allows our design to be used across different vehicle platforms or retrofitted when technology undoubtedly evolves. Following this standard also helps to streamline repair operations, thus reducing downtime.

3. Quality Assurance: Standardized specifications help to guarantee the quality of mechanical components by providing criteria for strength, dimensional accuracy, surface finish, and defect limits. They show outlines for testing procedures, which ultimately leads to improved product quality and reduced defects.

To maintain quality and durability within our project, we used data obtained in testing using standards such as **ASTM D638** for tensile strength of plastic and **ASTM B209** for strength of aluminum sheets. This helped to confirm that our polycarbonate housing and aluminum mount would be able to withstand mechanical loads without degradation.

4. Environmental Protection: Environmental Protection standards help engineers build systems that are weather resistant and could function reliably in conditions such as rain or dust.

Because our system is mounted externally to a vehicle, it was pivotal that we met the **IEC 605290 IP65** rating, ensuring that the housing was resistant to rain and water splashes, as well as dust. This also meant making sure all seams, joints, and ports were sealed using gaskets and weatherproof connectors.

4.2 Design Constraints

4.2.1 Time

The time constraint in the Pothole Detection and Tracking System is one of the most critical challenges in the project. Many of our components rely on real-time detection, and the overall speed of the system plays a major role in its success. The core functionality depends on how quickly the system can detect a road anomaly, confirm it using AI, and transmit that information while the vehicle is in motion. Because our system is mounted on a moving vehicle, these time constraints become more significant as speed increases. The system uses a GPS module on the main PCB to record the UTC time and GPS coordinates of each pothole, and the MSP430 continuously stores this information in a 30-second ring buffer. To ensure accurate location matching, the Raspberry Pi must deliver its detection results before the corresponding GPS timestamp falls out of this buffer, making end-to-end timing essential.

Our performance targets are based on a maximum vehicle speed of 30 miles per hour. At this speed, the camera systems must capture frames rapidly enough to guarantee that at least one clear frame contains the pothole. For the infrared camera, which handles both detection and measurement, the entire pothole must appear within the laser region in at least one frame. Since the laser projects a limited measurement area, this camera requires a higher frame rate. Based on our calculations, the infrared camera must operate at a minimum of 30 FPS to ensure reliable detection and measurement at 30 mph.

The RGB camera has less strict requirements, since it is only used for AI confirmation. It does not rely on the laser measurement region and does not need to capture the entire pothole; partial visibility is acceptable. To ensure at least one partial frame includes the pothole, we calculated that the minimum acceptable frame rate is 4.78 FPS at 30 mph. However, for increased reliability and to provide multiple frames for the object detection model, we set a target of at least 14.34 FPS so the pothole appears in roughly three frames during its visible window.

Another major part of the time constraint is the laser line deviation algorithm. Since the IR camera runs at 30 FPS, the detection logic must process each frame in under 33.3 milliseconds. In a real-world setting, headroom must be reserved for communication, buffering, and other system operations, so we target a processing time under 33 milliseconds per frame. While the measurement calculations themselves do not need to be instantaneous, they still must be completed soon after AI confirmation to prevent the system from falling behind while the vehicle continues moving.

Although the object detection model is not a hard real-time system, its processing speed is critical for maintaining overall timing accuracy. Since all location tagging relies on matching the Pi's detection timestamp with the MSP430's GPS timestamps, both devices must stay time-synchronized. The MSP430 provides periodic UTC information from the GPS module, and the Pi must update its internal timer to match these values to the millisecond. Because the GPS ring buffer only stores roughly 30 seconds of data, the entire detection pipeline, including AI inference, must complete within this window. At 30 mph, the vehicle moves approximately 44 feet per second, so any delay in detection or transmission can cause large positional errors if the timestamp does not correctly match a valid entry in the GPS buffer. To maintain accuracy, we determined that the object detection model must complete its analysis within 1 second.

Other factors that contribute to timing performance include USB communication rates between the cameras and the Raspberry Pi, the SPI communication speed between the Raspberry Pi and MSP430, the Bluetooth

transmission delay from the MSP430 to the user's phone, and the buffering time for the RGB frames. Although none of these components have strict deadlines individually, reducing their delays is important to avoid pushing the total system time close to the GPS buffer limit.

In addition to the technical timing, the project schedule itself is a major time constraint. As a Senior Design project with strict academic deadlines, delays in any part of the system can cause significant setbacks. Progress in hardware testing, firmware development, software integration, and report writing must align with required milestones. Each student must produce 20 written pages by the midpoint of the first semester, with a complete 150-page report due at the end of that semester. A prototype demonstration video is also required, meaning the core system must already be functional by that time. By the end of the second semester, the system must be fully finalized, including an in-person demonstration and presentation. Since these deadlines cannot be shifted, the team must stay organized, follow a clear schedule, and prepare for unexpected difficulties. Without strong planning and collaboration, even a technically successful system could fail to meet the required academic milestones.

4.2.2 Cost and Resources

The cost constraint is a major factor in the design and execution of the Pothole Detection and Tracking System. Since this is a Senior Design project, our budget is very limited. The University of Central Florida does not provide us with any financial support, leaving us with only two options: either securing a sponsorship or covering the costs out of our own pockets. It's also worth noting that one of the key selling points of our design is how much more affordable it is compared to other systems with similar functionality currently on the market. We're further limited by what is commercially available in small quantities, particularly when it comes to specialized technology. While much of the system is designed by hand by our group, some components would either take too long to design given our timeframe or would require manufacturing resources we simply don't have access to. This can limit the level of cost reduction we can implement.

We decided not to get a sponsorship for this project, relying entirely on personal funding. This makes our budget even tighter, since all costs must now be split between the five team members. Our goal is to keep the total project cost under \$1500. That way no one must pay more than around \$200 total. As mentioned before, what sets our design apart from other available designs is its cost-effectiveness in creating a fully autonomous pothole detection and tracking system. Every part of the design requires extensive research to identify the cheapest component that could still meet our performance requirements.

When working with technologies like cameras, sensors, and lasers, staying within budget can be difficult. These components often come at a high cost, and we constantly must balance quality against affordability while also exploring practical alternatives. For example, LiDAR is one of the most powerful and accurate technologies available for our application. However, even cheaper LiDAR units are extremely expensive, often costing several thousands of dollars. This raises the question of whether the performance benefit is worth the cost, and in our case, it is not. Since there are no cost-effective LiDAR options that meet our needs, we opted for a lower-cost alternative. We decided to pair an infrared laser with a filtered IR camera and add an RGB camera for AI confirmation. This approach allows us to get reasonably close to the detection accuracy of more expensive systems without exceeding our budget.

Resource availability also played a large role in our decision making throughout the design process. Many of the technologies we needed had very specific requirements and, when those features weren't available in cost-effective components, we were often forced to search through limited options to find hardware that matched our constraints. One example of this was the RGB camera used in the object detection model. Our design required a camera with a 1920×1080 resolution, a high frame rate, and a field of view between 90 and 100 degrees to match the IR camera. To transmit this data reliably, USB 3.0 was also required. That exact combination of specifications is uncommon, and it significantly narrowed our choices. After reviewing all available options, we were able to find a 1080p RGB camera that met the required resolution, frame rate, FOV, and USB 3.0 bandwidth. While it still cost more than we initially expected, it prevented us from needing to purchase a much more expensive 4K camera and avoided the need to downscale higher-resolution footage. In a large-scale setting with more time and funding, a custom camera could be sourced or designed to include only the necessary features, which would reduce overall hardware costs.

Other hardware costs were also considered throughout the project. While we do have access to more tools at the university than we would at home, we still don't have important resources such as PCB manufacturing. Having this done through a private company tends to be expensive, especially with additional fees like shipping costs and customs taxes. To help keep these costs low, we have options like ordering multiple boards at once, buying from suppliers outside the country, or combining orders with other teams in the class to reduce the per-unit cost. Another example is Bluetooth design. While the most practical and affordable solution would have been to use the Raspberry Pi's built-in Bluetooth, one of the class requirements is that components unrelated to computer vision must be included in our custom PCB. Because of that, we had to look for the cheapest possible Bluetooth design that was still reliable and compatible with our system. When looking at ways to offload computing for our computer vision

tasks, we also considered several options. However, most of them were either too expensive or too complicated to integrate, especially considering our soldering limitations in terms of experience. In the end, we had to focus more on performance and reliability than cost, since this is one of the most important parts of the hardware design.

Another major cost-related issue in our project involves training, testing, and prototyping. We can't just simulate hundreds of potholes, and we can't afford to rent out a road or create test potholes on private property. Because of this, we must collect our own training and testing data using potholes we find on public roads. Training an object detection model takes hundreds, sometimes thousands, of labeled pothole images, each with its own unique shape and appearance. While some images can be found online, most of the training data must be collected by us in person, which takes a lot of time and effort. The same problem arises when testing the system. During early testing, we can use artificially made potholes, but for the final version of the project, we need to use actual potholes on real roads, at realistic speeds to ensure it works as intended. This means spending a lot of time driving, testing in multiple locations, and recording accurate results, all of which take up valuable time and resources.

4.2.3 Processing Power

Processing power is a constant constraint that must be considered in nearly every decision we make. One of our project requirements is to have as many components as possible placed directly on the PCB. However, because our project depends heavily on computer vision and multiple cameras and sensors, we have no real options for standalone MCUs. The few that could work would require surface-mount soldering well beyond our current capabilities. Our only realistic option is to use a basic MCU, something like the MSP430, and offload all the processing required for computer vision onto a secondary device that can handle it.

We chose the Raspberry Pi 5 for this role due to its acceptable processing capabilities and variety of I/O ports. While it features a 64-bit 2.4 GHz quad-core Cortex-A76 and 8 GB of RAM, the hardware still comes with its share of limitations. Running two cameras with decoding, preprocessing, a lightweight detection algorithm, and a ring buffer in real time is already a serious challenge. On top of that, the device also needs to handle a computationally heavy object detection model, diameter and depth calculations, SPI encoding, and a range of smaller tasks. It's all technically possible, but only with carefully-planned hardware choices, software optimization, multithreading, and a few necessary trade-offs.

Since the Raspberry Pi is only permitted for vision tasks, it's likely outside the scope of the project requirements to include hardware accelerators. Something like the Coral USB Accelerator would be considered second-level offloading and may conflict with the hardware restrictions of the class. While it would give the Pi a significant boost in headroom, it would also narrow our model options and add about \$100 to the budget.

One of the biggest computational constraints is the camera and sensor handling. The cameras require a minimum frame rate and resolution to function correctly. The infrared camera needs a much higher frame rate due to its limited vertical field of view. But increasing frame rate and resolution also increases bandwidth usage. The Raspberry Pi 5 shares a USB bus for all USB 2.0 ports. So, if both cameras run over USB 2.0, it'll easily overwhelm the available bandwidth. On the other hand, only one camera should use USB 3.0 due to reported issues with dropped frames when both ports are used simultaneously for cameras.

USB 2.0 offers very limited bandwidth. We can reduce some of the load by using grayscale infrared frames, but even that's pushing it. Affordable camera options that support high-framerate YUY2, with no IR filter, and USB 2.0 compatibility don't exist. That leaves MJPEG as the only viable option. Encoding images as MJPEG reduces the bandwidth well enough to work over USB 2.0, but decoding them takes significantly more processing power and introduces delay. This becomes a real problem, because the image processing, decoding, and lightweight detection algorithm need to finish before the next frame comes in. With the limited processing power available, the only real solution is to isolate the infrared camera, its decoding, and the lightweight detection algorithm into their own dedicated thread or process.

The RGB camera used for object detection has different needs. It runs at 1920×1080 resolution, captures in color, and only needs half the frame rate of the infrared camera. This still requires a lot of bandwidth, making USB 3.0 the only realistic choice. Fortunately, the RGB camera uses YUY2, which is much lighter on the processor and easy to convert into a usable image. The image processing must happen in real time, but we only run object detection on the RGB images when the IR camera makes a detection. That gives us some breathing room for preprocessing and managing the image buffer, all of which are handled in the same dedicated thread. Using dedicated threads for both camera systems helps minimize bottlenecks, frame drops, and potential system crashes.

The object detection model itself is another major concern when it comes to computing limits. We're restricted in terms of model size and complexity, especially since the camera systems already take up about half of the Raspberry

Pi's processing power. We had to choose the lightest possible model that could still meet our accuracy requirements. Even with the smallest model, it still takes a lot of optimizations to run efficiently. To help with that, we made sure not to overload the model's thread with other heavy tasks. It runs in its own thread, along with any other code that won't get in the way. Everything else, such as calculations, encoding, and SPI transmission, runs on its own separate thread.

Finally, the MSP430 MCU has its own severe limitations. It's a very lightweight 16-bit single-core microcontroller with very limited processing capability. Its job is to act as the main control unit for the entire system. Its responsibilities include: controlling the laser, handling SPI communication with the Raspberry Pi, and sending measurements and pothole images to the Bluetooth module. Since this is a very light workload, the MSP430 ends up being a suitable choice.

4.3 Mechanical Design Constraints

The Mechanical system must conform to constraints imposed by real world usage, environmental exposure and interdisciplinary team integration. These constraints ensure that the mount and housing not only function under stress, but also remain aligned, safe, and maintainable. Each constraint listed below has been considered and integrated into the project design.

Table 4.3.1: Mechanical Design Constraints

Constraints	Requirements	Justifications
Vibration Tolerance	Withstand Vibrations	Prevent sensor misalignment
Environmental Sealing/WaterProofing	IP65 Standard or Higher	Protect against rain, splashes, and dust
Weight Limit	Less than or equal to 5 pounds for housing	Prevent excess load on mount and vehicle exterior
Operating Temperature Range	0°C - 45°C	Ensure operation in most common temperature range
Serviceability	Modular Design with interchangeable Parts and Easy removal	Helps with ease of repair to reduce complexity

The system is mounted on the exterior of a moving vehicle and is constantly exposed to vibrations, especially when driving over rougher surfaces, potholes, or uneven roadways. Research shows that load frequency road vibrations fall between 5-30 Hz, so the mounting system and housing needs to reflect this. If either of these are not adequately designed, the vibrations can cause sensor misalignment and drift, thus leading to inaccurate detection.

To account for this, vibration damping materials have to be used. Additionally, the design must ensure that the natural frequency of the mount is outside of the typical road vibration range to prevent resonance. This is ensured by strategic material selection and geometry optimization.

4.3.1 Environmental Sealing / Waterproofing

The system will operate in uncontrolled outdoor environments, often experiencing water spray, rain, dust, and debris resulting from tire-kicked up material. Meeting the IEC 60529 standard, an IP65-rated enclosure offers full protection against dust and protection against water jets from any direction. This level of sealing is necessary to seal off internal electronics inside the case, including the camera, the IR laser, and PCB, all of which are sensitive to moisture and debris. Sealed cable ports, rubber gaskets, silicone sealant, and possibly hydrophobic vent membranes can be employed in order to achieve this rating to allow internal pressure equalization without the entry of internal moisture. Under colder conditions, freeze-thaw cycling also induces condensation and micro-leakage, and thus material compatibility and expansive behavior are issues. Environmental protection maximizes long-term reliability, reduces failure rates, and reduces maintenance needs. Rigorous validation is attainable with lab testing to model rainfall, dust storms, and hose-down cleaning conditions to ensure performance validation.

4.3.2 Weight Limit

The mounting system is externally mounted onto a vehicle and cantilevered off the surface. Excess weight would create a greater bending moment on the mounting arm and add unnecessary stress to the vehicle body, especially when it accelerates, brakes, or vibrates. Keeping the housing at under 5 pounds retains mechanical integrity, reduces reinforcement needs, and averts damage to mounting surfaces. It also ensures that the system can be mounted without engaging structural additions or heavy-duty brackets. Polycarbonate, 6061-T6 aluminum, or reinforced nylon composites are materials with an adequate strength-to-weight ratio. Less weight also facilitates installation and reduces safety concerns involved in handling. If expansion is required (e.g., the

addition of sensors), the base system must stay within its quota so that things can remain in equilibrium.

4.3.3 Operating Temperature Range

The temperature range of 0°C to 45°C was selected because it covers the most common environmental conditions the system would be exposed to when operational. Most locations have wide temperature ranges, and components must be able to survive and function within spec through the range. Electronics such as the Raspberry Pi, battery packs, and image sensors must be rated to function within these ranges. The material of the mount and housing cannot warp, crack, or degrade as a result of thermal expansion and contraction. Polycarbonate, for instance, is insensitive to this range and will not yellow or deform easily. Careful thermal engineering, like ventilation or passive heat discharge paths, will be necessary in order to avoid overheating in warm environments. Startup reliability and battery life may be affected in cold environments, so cold-weather testing or preheat logic may be beneficial. Compliance with this temperature range ensures usability in several geographically separated regions and climates.

4.3.4 Serviceability

Field serviceability is critical in a system mounted on fleet vehicles or on city-owned equipment. Over time, elements may wear out, fail, or require upgrading. A modular construction with clearly established mounting points, removable connector cables, and standardized fasteners permits quick repair or substitution of parts without disassembling the entire unit. This reduces downtime and cost and simplifies service and maintenance logistics. Modularity also supports scalable manufacture and enhanced capabilities—new processor or sensor modules can be added without system redesign. Using readily accessible fasteners (e.g., Phillips, Torx), keyed connectors, and color-coded parts makes the system technician-friendly. Enclosures need to allow quick access to interiors without sacrificing environmental sealing on reassembly. Serviceability not only prolongs system life but also facilitates quick deployment and utilization in the actual operating environment where quick turnaround is critical.

5. Pros and Cons of ChatGPT

Artificial Intelligence (AI) refers to the creation of machines and systems that can perform tasks that would normally require a human to do them. These tasks include thinking, learning, problem-solving, and understanding language, all of which are used to enhance human capabilities. In today's world, AI has

become a commodity, deeply integrated into daily life and business operations (and school projects), from typical, routine tasks, to highly complex ones.

One of AI's advantages is its ability to process and analyze huge amounts of data quickly and accurately, and can do so at a much faster rate than a human can. This lets AI systems identify patterns, make predictions, and provide insights that influence decision making across a number of industries. Additionally, AI automates repetitive and mundane tasks, which improves efficiency and allows workers to focus on more complex or creative endeavors. Its consistency and precision also reduce the likelihood of human error, which leads to higher-quality outcomes.

Well-known examples of AI include: ChatGPT, by OpenAI; Copilot, by Microsoft; and Gemini, by Google DeepMind. ChatGPT, launched by OpenAI in November 2022, functions as a chatbot and virtual assistant. It uses machine learning algorithms to process and analyze massive datasets, enabling it to provide real-time, well-crafted responses to user inquiries. Whether it's writing an essay, solving math problems, or debugging code, ChatGPT delivers solutions in seconds and can complete tasks that would normally take a human hours to do.

Another notable AI platform is Gemini, formerly known as Bard, developed by Google and released in March 2023. Just like ChatGPT, Gemini can understand and generate human-like text, and respond in real-time. However, Gemini is designed to integrate closely with Google Workspace, which can improve productivity through suggestions and data analysis within the Google ecosystem.

Although AI has many pros, these systems have their cons as well. The reliability of AI heavily depends on the quality of the data that is used for its training. Biased, outdated, or inaccurate data can lead to poor or misleading results. Furthermore, AI often struggles with understanding nuanced context or handling ambiguous inputs. Models trained in cases outside their training data. Technical issues, such as software bugs or hardware malfunctions, can also compromise performance. Lastly, these systems can come with ethical and security concerns, which include potential copyright violations and privacy breaches, which pose additional challenges. Addressing these issues is crucial to developing AI technologies that are more reliable, ethical, and trustworthy.

What this chapter hopes to do is look at some of what you might find by posing questions, or prompts, to ChatGPT. The design team chose to use ChatGPT over other AI systems because it is the most widely used, and one of the team members has a premium account, which is likely to produce the best solutions or answers possible. In this section, the design team is going to pose

some prompts to ChatGPT, receive an answer, and comment on what it got right, what it got wrong, what the team liked about the answer the system gave, and what the team disliked about the answer that the system gave. This is important because of the power ChatGPT poses in helping the design team throughout their project.

5.1 Case Study 1

Prompt: “Tell me about MCUs”

Response: See Appendix C

Strengths of Response:

This response did a great job of explaining the concept of Microcontroller Units (MCUs) in a clear and specific way. The explanation of an MCU covered all of the essential components such as CPU, RAM, flash memory, I/O ports, and peripherals, without getting too technical, making the information digestible to even a novice, while covering enough that an experienced person can appreciate it. The bullet points are very helpful and effective with summarizing the typical use cases.

The inclusion of the popular MCUs table is a great addition to this response. It shows concrete examples from major manufacturers and highlights their strengths, which can help readers of all experience levels quickly understand how different families fit into different applications. Similarly, the MCU vs CPU vs Microprocessor is helpful in clearing up the distinction between these commonly confused terms. This comparison table ties directly to the narrative set by ChatGPT and reinforces the reader’s understanding.

Weaknesses of Response:

While ChatGPT’s response to the MCU case study was very detailed and informative, there were a few areas where it could have been better. For one, the structure of the content could have flowed more smoothly. The transitions between explaining what an MCU is, listing their uses, and then presenting the tables felt a bit short. Adding connecting sentences or brief summaries between these sections would have made the whole explanation feel more cohesive. Also, while the tables were helpful for quick reference, ChatGPT could have included real world examples for each MCU family to make it more relatable and digestible, such as explaining where an MSP could be found or used in a real world application.

5.2 Case Study 2

Prompt: “What are the benefits of using switching regulators vs LDO regulators?”

Response: See Appendix C

Strengths of Response:

The response to case study 2 on the benefits of using switching regulators vs using LDO regulators was overall very structured and informative. One of the key strengths was how it broke down the comparison into clear sections, outlining the advantages and disadvantages of each regulator type in a way that was easy to follow. The inclusion of the table was especially helpful for identifying which type of regulator is best for different use cases.

The response also provided real world examples, like powering an ESP32 or an MSP430, which made the information more practical and relatable. Additionally, the explanation provided important design considerations such as efficiency, heat dissipation, and noise, which are relevant for anyone working with embedded systems or power supply design.

Weaknesses of Response:

There were a few areas where the response could have been improved. While the content was very detailed, some parts could have been made more concise to better fit into a technical report without requiring much editing. The response could also have included references to specific datasheets or technical resources in order to strengthen credibility.

Additionally, while the examples were good, adding a small diagram or schematic snippet showing the typical components for an LDO vs a switching regulator would have provided more visual clarity. Lastly, a brief summary at the end to reinforce the key differences and design trade-offs would have made the section feel more complete and tied everything together.

5.3 Case Study 3

Prompt: “What are the pros and cons to use these serial interfaces for the Bluetooth module?”

Response: See Appendix C

Strengths of Response:

The response to case study 3 on the pros and cons of using UART, SPI, and I2C for a Bluetooth module was very easy to read and well organized. One of its main strengths was how it broke down each interface into clear sections, presenting advantages and disadvantages in well-labeled tables. This made the information easy to navigate and compare. The inclusion of a quick comparison table at the end was very effective in summarizing key features like speed, device support, and ideal use cases. Additionally, the recommendation to use UART for the Bluetooth module was very practical and I felt like ChatGPT backed this up with reasoning about typical BLE compatibility. This explanation also had a good mix of technical depth with accessibility, making it useful for readers with different levels of experience.

Weaknesses of Response:

For all of the strengths that the response had, there were some areas where the response could have improved. The response lacked citations from authoritative sources, such as technical datasheets and links to documentation from standards organizations like the IEEE.

6. Hardware Design

This section presents a comprehensive overview of the hardware design for our pothole detection system. It shows the careful selection as well as the integration of the different components selected by the design team. These components include: the IR laser with Powell lenses for the light projection, the IR camera for detecting distortions in the projected light, and the RGB camera used for visual confirmation via trained AI model. These optical elements are key to the system's ability to identify potholes accurately while the vehicle is in motion. Supporting this functionality is a control and power subsystem built around the MSP430fr2355, the LMR33630 regulator, and the Raspberry Pi 5.

In order to depict these hardware systems as accurately as possible, this section includes a variety of visual aids. Detailed schematic diagrams give insight into electrical connectivity, such as power routing, GPIO assignments, and signal flow between components. An overall system design is also included to show the spatial arrangement of hardware mounted to the back of the vehicle, as well as within the vehicle.

By combining component analysis with a clear visual documentation, this section shows how each element and component works together to fulfill the system's objective of real-time pothole detection and logging. The integration of

optical, power, and communication subsystems gives the user reliable autonomous operation.

6.1 Overall System Design

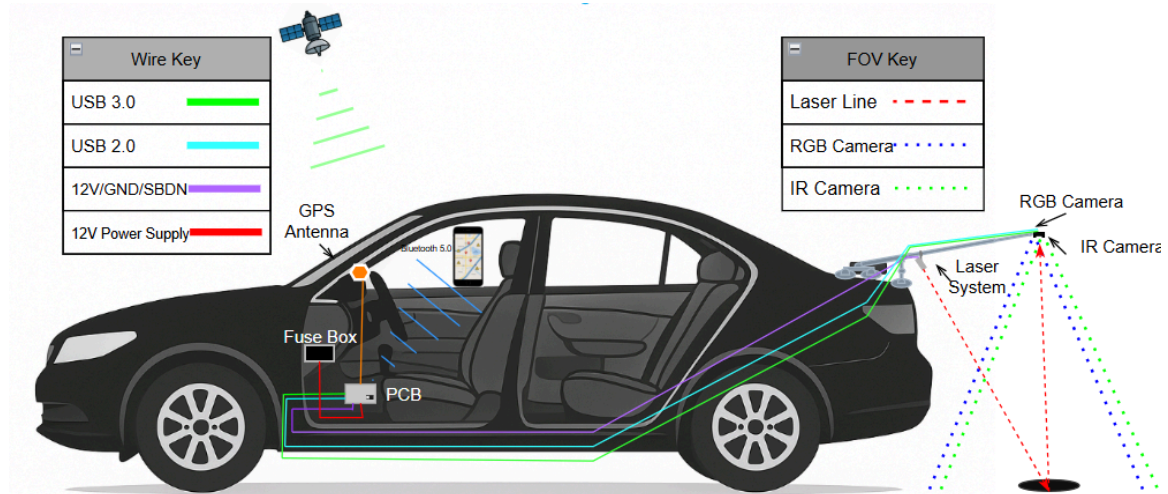


Figure 6.1: Overall System Design Illustration

The overall system design, as shown in the diagram above, is made up of six key components that all work together to ensure the system functions as intended. These components include the PCB housing unit, the user's mobile phone, the 12-volt power supply, the sensor mounting device, the sensor housing unit, and the sensor wire harness. Each one plays an important role in the operation of the system. The PCB housing unit handles processing and connectivity, while the user's phone acts as the interface and data display. Power for the entire system is delivered from the vehicle's 12-volt supply using a fuse tap connected to the 20-amp cigar lighter fuse located under the driver's dashboard, with the system's ground connected to a chassis bolt. The mounting device securely holds the sensor assembly to the vehicle, and the sensor housing itself contains the camera and other sensing components. Finally, the wire harness links everything together, providing both power and data connections routed from the driver's footwell toward the rear of the vehicle. All of these components were designed to allow for flexibility between different vehicle types while keeping cost as low as possible.

The PCB housing unit serves as the central hub of the entire system. It is mounted along the side of the driver's footwell, not only for easy access but also to keep it protected from the elements while ensuring proper ventilation. Inside the housing is the MSP430 microcontroller, a Bluetooth module, the system's voltage regulation circuitry, the onboard GPS module, and a Raspberry Pi 5

responsible for computer vision processing. The GPS antenna exits the enclosure and is placed on the vehicle's dashboard to maintain strong satellite visibility. The unit's primary role is to process all data from the sensors and transmit detection results to the user's device via Bluetooth. Power is supplied through the 12-volt fuse tap, with the ground wire secured to a nearby chassis bolt. A switch is built into the housing, giving the driver a simple way to turn the system on or off, and all voltage levels required for the system are regulated internally within this unit.

The wire harness plays a critical role in powering the sensor assembly and transmitting image data from the cameras back to the PCB housing unit. It consists of multiple interior-routed wires: a 17-foot line carrying the 12-volt supply, a 17-foot 3.3-volt SBDN laser-control line, and a 17-foot ground line, as well as two 24-foot active repeater cables, one USB 3.0 and one USB 2.0, to carry data from the infrared and RGB cameras. These wires are connected directly to the PCB housing, where the SBDN control signal is driven by the microcontroller and the USB cables plug into the Raspberry Pi 5's ports. The entire harness is routed through the driver's footwell and under the interior floorboards, following the factory wire harness toward the trunk. Once it reaches the rear of the vehicle, the harness exits the cabin and connects directly to the external sensor unit. Because the camera cables extend 24 feet, far exceeding typical USB limits, active repeater cables are required to maintain signal integrity. The 3.3-volt SBDN control line and 12-volt line must also maintain stable voltage over their 17-foot length to ensure reliable infrared laser operation.

The mounting device is a crucial component that ensures the sensor housing unit can be reliably and accurately positioned on the vehicle. One of its key requirements is universal compatibility, as it must be adaptable to a wide range of government vehicles, which can vary significantly in make and model. The design also prioritizes vibration reduction, which is especially important given the amount of movement and road noise generated during normal vehicle operation. This device also has several degrees of freedom, allowing it to be positioned exactly how it is needed. The mount includes a telescoping pole that allows the cameras to be extended far enough behind the vehicle to ensure that the car itself does not appear in any of the captured frames. This not only preserves the accuracy of the sensor data but also maintains consistency across different vehicle setups. The mount's design balances durability, flexibility, and vibration resistance, making it suitable for deployment across diverse vehicle platforms.

The sensor housing unit is specifically designed to securely hold both the infrared and RGB cameras. Since these components are mounted outside the vehicle, protecting them from environmental exposure is critical to system

reliability. The housing is waterproof, preventing moisture from entering and damaging the electronics. A key feature of the design is the camera window, which allows infrared light to pass through; this is essential for pothole detection, as the system relies on infrared imaging for accurate readings. One of the more complex challenges in the design is routing the wires into the housing while still maintaining a waterproof seal, ensuring the system remains protected under all weather conditions.

The final, main component of the system is the user's cell phone. The phone runs a custom-built app that connects via Bluetooth to the PCB's onboard module. When a pothole is detected, the MSP430 microcontroller transmits the detection data, including the measured area of the pothole, its depth, the GPS coordinates supplied by the onboard GPS module, the time of detection, and an image directly to the phone. The app uses its internal processing to display the information to the user and upload it to the server using the phone's cellular connection. Once compiled, the data is uploaded to our server, and the pothole is displayed as a pinned marker on a shared map, making it visible to all authorized users.

6.2 Electrical Design and Power Delivery

The electrical design and power delivery process of the pothole detection system starts with the vehicle's 14V DC supply sourced directly from a fuse tap on the vehicle, protected by a 15A fuse, which will supply input power to all of the components on the board. This 14V line is protected using a Schottky diode to prevent damage from a reverse polarity connection, which makes sure only the properly-oriented voltage reaches the rest of the circuit.

From here, the 14V input is routed to three LM2679 buck regulators, which are configured to step this 14V down to a stable 5V output, 5.2V output, and one regulator is configured to step this input voltage down to a stable 3.3V output. These switching regulators were selected for their high efficiency and reliability. Bulk input capacitors filter incoming power, and smaller ceramic capacitors near the output terminals suppress voltage ripple and noise with high frequencies. Feedback resistors set the output voltage, and soft-start capacitors manage smooth power up and switching operation.

Once this 14V input is regulated through the 3.3V regulator, the rail branches out to power all components requiring this input voltage such as the MSP430, Bluetooth module, GPS module and buzzer alarm. The MSP430 functions as the main controller for the system's logic, timing, and low level signal coordination. The Bluetooth module communicates wirelessly with a mobile application to log pothole data, such as time stamps, as well as the GPS

coordinates of said pothole. The buzzer alarm will sound within the cabin of the vehicle in order to let the user know when the system has detected a pothole. Additional components, such as buttons and programming headers, also draw from this 3.3V output.

The first 5V regulator steps down the 14V input voltage to 5V, specifically for the IR laser subsystem. This regulated 5V output supplies power to the laser driver and diode, so that the subsystem can project the light pattern onto the road.

Table 6.2.1: First 5V Regulator: Laser System

Voltage	Part #	Part Description	Qty	Max Current Draw
5 VDC	LMR33630	Buck Regulator	1	25 uA (max)
5 VDC	ATLS1.5A104D	Laser Driver	1	500 mA (max)
			Total Max Current Draw	~500 mA
			Total Max Power	2.5 W

The 5.2V regulator is responsible for powering the Raspberry Pi 5, which is responsible for high-level processing. The Raspberry Pi 5 processes the images taken by the RGB camera, using a trained AI model, in order to confirm whether or not the disturbance in the light sheet is a pothole. Once the model has its verdict, the binary result of “is this a pothole, yes or no?” is sent to the MSP430.

Table 6.2.2: Second 5V Regulator: Raspberry Pi

Voltage	Part #	Part Description	Qty	Max Current Draw
5 VDC	LM2679	Buck Regulator	1	25 uA (max)
5 VDC	B0CK2FCG1K	Raspberry Pi 5	1	5 A (max)
5 VDC	B0DBV8VFJZ	RGB Camera	1	290 mA (max)
5 VDC	B096ZSSCGH	IR Sensitive Camera	1	450 mA (max)

			Total Max Current Draw	5.74 A
			Total Max Power	28.7 W

By segmenting the system into three independent, regulated domains, the design ensures isolation of power-hungry components, like the Raspberry Pi and laser driver, from the digital control and communication components. Also, the ground is shared uniformly to help prevent potential ground loops or voltage offsets.

In summary, the power requirements of the system are distributed across three dedicated voltage rails, each supplied by its own regulator to ensure stability, isolation, and reliability. As shown in the tables above, the design table was able to pull the max current draw for each component from their respective datasheets. This allowed the design team to calculate the maximum power required for each component, as well as the total max power required for each regulator. The largest amperage required for a single component was 5 amps for the Raspberry Pi 5. The design team may have to change the regulator in the future to ensure the amperage rating for the regulator can handle that of the Raspberry Pi 5.

Table 6.3.1: 3.3V Regulator

Voltage	Part #	Part Description	Qty	Max Current Draw
3.3 VDC	LM2679	Buck Regulator	1	25 uA
3.3 VDC	MSP430FR2355	MCU	1	6 mA (max)
3.3 VDC	BM71	Bluetooth Module	1	16.5 mA (max)
3.3 VDC	CMT-5023S-SMT-TR	Buzzer	1	110 mA (max)
3.3 VDC	PTS645SM43SMTR92LFS	Bluetooth Pairing Button	1	Negligible
3.3 VDC	1528-1153-ND	GPS Module	1	48 mA
			Total Max Current Draw	180.5 mA
			Total Max Power	.596 W

6.2.1 Overall Electrical Design

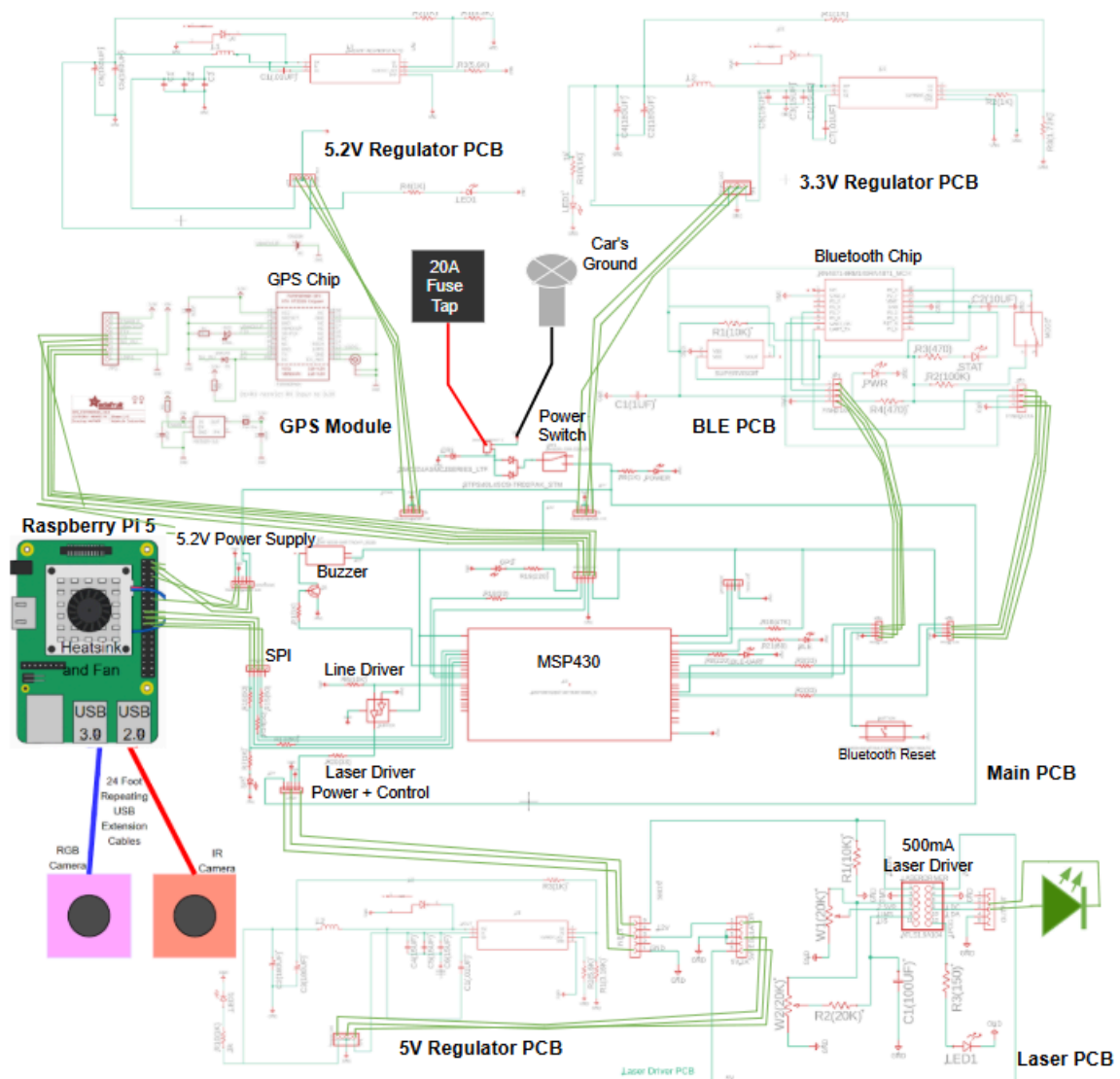


Figure 6.2: Overall Electrical Design

6.2.2 Main PCB Schematic

The initial PCB schematic for the pothole detection system was developed with Fusion 360 Electronics. This is a CAD tool that supports both schematic and PCB design in an integrated manner. Fusion 360 offers a broad range of standard components and symbols and also permits users to make their own parts. For instance, the design team customized the Raspberry Pi 5 header to indicate the pins of a real Raspberry Pi 5. Besides, Fusion 360 users can

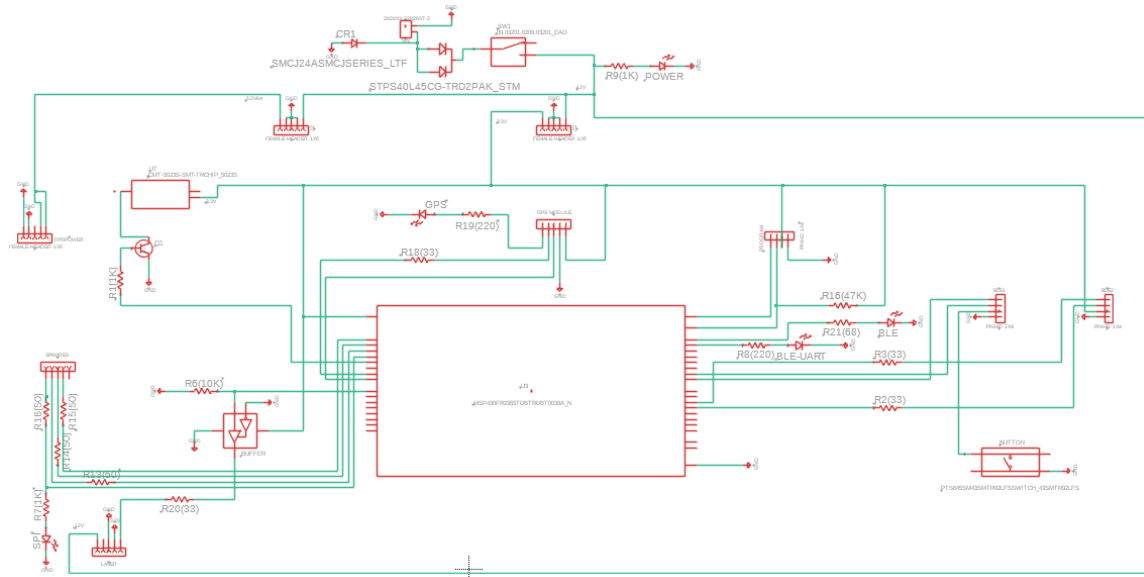
download footprints and 3D models from the external sources like Ultra Librarian, Mouser, and Digi-Key. So, the electrical schematic for this system has been accurately modeled with a mixture of the Fusion 360 in-built library, Ultra Librarian, and some custom components.

The main PCB schematic shown below represents the electrical design of the core control, communication, and system-level circuitry. Near the middle right side of the board, the MSP430FR2355 microcontroller, which serves as the brain of the system, is the central component of this circuit. It is the one to which all the other parts of the system connect for control signals management, peripherals triggering, system state monitoring, and external modules interfacing, along with the GPS module board mounted on the upper left side; the Bluetooth PCB positioned on the upper right side; the standalone regulator boards; and the laser driver board. The laser driver is therefore the one to be activated when the MSP430 tells it so, i.e., based on the state of the system and the communication between it and the Raspberry Pi, it determines the infrared laser is to be on.

Energy is fed into the arrangement of the system through a 12 V supply rail that is guarded by two diodes which protect the hardware from reverse polarity and vehicle-induced transients. The power is then sent to the regulator boards one by one. A Raspberry Pi is powered by a 5.2 V regulator PCB. This device with two 5.2 V wires and two ground wires is connected directly to its power pins, and thus the power is supplied. A 3.3 V regulator PCB is meant to provide power to the microcontroller, communication lines, and connected peripheral boards.

The Bluetooth module which is on a separate PCB to the right upper side of the base board and connected via UART lines is powered by the 3.3 V regulator board. A physical reset button and pull-down resistor have been added to the Bluetooth board so that the user can manually reset the device when pairing or reconnecting mobile devices. The GPS module on a different PCB to the left upper side of the layout is connected to the MSP430 via UART and is providing the UTC timestamps and GPS coordinates that the system is using.

Moreover, there is a buzzer alarm which is turned on when the SPI connection has been successfully established between the Raspberry Pi and the MSP430FR2355 microcontroller. This part is directly driven by a GPIO pin from the MSP430FR2355 and is used as a hearing confirmation that the communication between the two boards is going on. So, it gives instant feedback to the user or developer without the need to look at a display or debug terminal.



Essentially, the MSP430FR2355 is the center of the PCB schematic that controls the system logic. With the help of Fusion 360, the design team was able to come up with a thorough depiction of what the main PCB will look like. Since Fusion 360 provided the design team with the possibility of using already installed components, importing components, and creating custom components, it gave the design team a high degree of flexibility for the main PCB and supporting boards.

6.2.3 Laser Driver Schematic

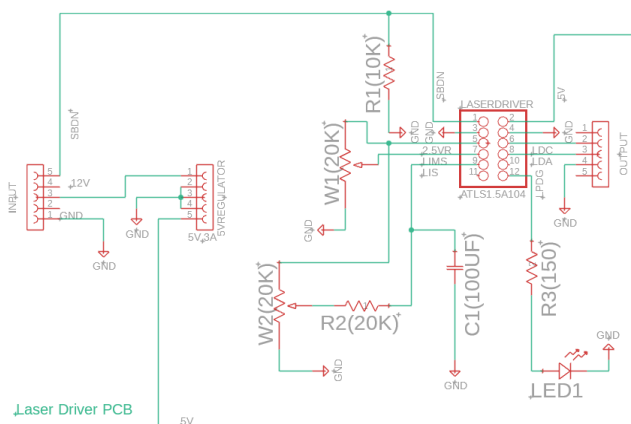


Figure 6.4: Laser Driver PCB Schematic

The other PCB in the pothole detection system is a control board for the laser projection system. It contains the laser driver and the associated circuit elements, which are the parts that both supply and control the laser diode used for the structured light projection. The board obtains its own regulated 5.2V power from a dedicated regulator PCB, which is arranged in such a way that it fits nicely onto the laser driver board itself, unlike the other regulator boards that are completely separate. By closely coupling the regulator and the driver, the design changes in the wiring complexity and the electrical noise near the laser circuit are minimized while the whole assembly is still isolated from the communication and processing systems on the main PCB.

Passive elements such as inductors and capacitors are employed to filter and stabilize the switching output. Two potentiometers located on the board are used to adjust the laser driver: one potentiometer is for the laser diode operating current, and the other potentiometer is for setting the current limit to protect the diode under all conditions. The laser is ensured to be working within safe electrical boundaries while the optical power required for accurate sensing is maintained.

The laser driver circuit is centered on the ATLS500MA module. As the exact part footprint could not be directly imported into Fusion 360, the design team decided to make a custom multi-pin header to represent how the driver connects to the system by sending power, ground, and control lines directly to the laser diode. The LED and the resistor together provide a simple visual indication of the activity. Generally, this PCB is committing the biggest part to the job of stabilizing the laser output and making sure that the pothole detection system works as it is supposed to.

6.2.4 5.2V Regulator Schematic

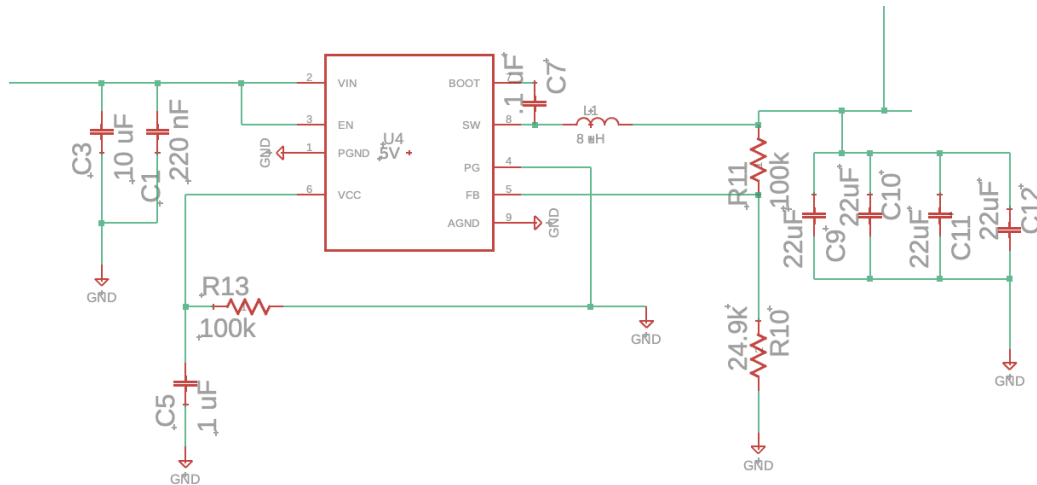


Figure 6.5: 5.2V Regulator Schematic

Shown above is the 5.2V regulator schematic the design team is using to reduce the overall input voltage from 12V to 5.2V in order to power the respective peripherals and components. This specific 5.2V regulator supplies a steady 5.2V to the headers which in turn power the Raspberry Pi 5.

At the center of this design is the switching regulator IC (LMR33630). However, the design team may have to adjust which regulator is used due to amperage requirements. The input state consists of capacitors C1 and C3, which help suppress the high-frequency noise and stabilize the input voltage. A soft-start or enable function is implemented with R13, pulling the EN pin high to allow usage. Bypass capacitors like C5 and C7 make sure the internal circuitry of the regulator will function reliably.

The output stage consists of a 2.5 μH inductor and four, parallel 22 μF capacitors, which filter and smooth the output voltage to reduce ripple. A resistive voltage divider made up of R10 and R11 connects to the feedback (FB) pin, which sets the output voltage to 5V. This layout ensures proper power filtering and low impedance routing for switching paths, which is crucial for maintaining regulator efficiency and output stability. This regulator circuit is essential for delivering a steady 5.2V to power devices such as the laser driver and Raspberry Pi 5.

6.2.5 3.3V Regulator Schematic

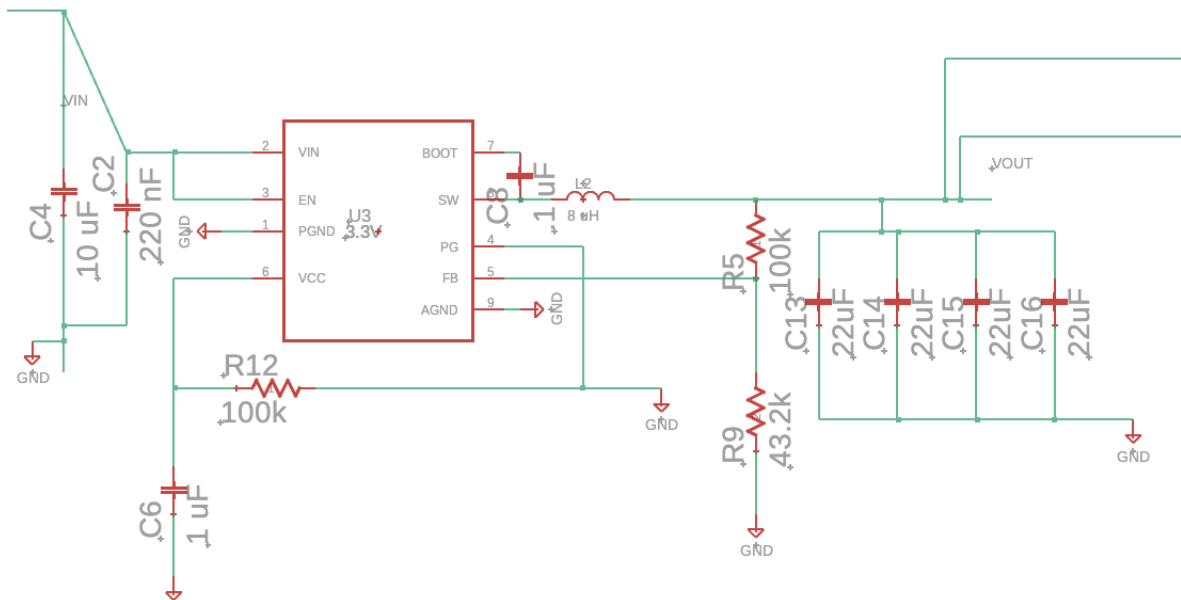


Figure 6.6: 3.3V Regulator Schematic

This schematic represents the 3.3V buck regulator circuit, which is designed to convert a higher input voltage, 12V in this case, down to a stable 3.3V supply. The circuit uses a switching regulator IC, like the 5V regulator previously mentioned. As it stands, the switching regulator the design team is using is the LM2679-ADJ. As previously stated, the design team may have to make a change at the 5V regulator due to maximum amperage requirements. This is not the case here, because the components powered by this regulator will have a summed amperage of about 660 mA, much lower than the maximum amperage rating for this regulator.

Input capacitors C2 and C4 provide noise filtering and input stabilization, while R12 pulls the EN high to turn on the regulator. Capacitor C6 is used for VCC bypassing, ensuring reliable internal regulator operation. On the output side of the circuit, an 8uH inductor smooths the switching pulses, followed by a bank of four 22uF capacitors in parallel, which reduce output ripple.

A feedback voltage divider composed of R5 and R9 sets the output voltage at 3.3V by referencing the internal feedback pin of the IC. The overall design ensures a clean and stable 3.3V output, which is crucial for powering low-voltage components in the system such as the MSP430FR2355 microcontroller, BM71 Bluetooth module, and the buzzer alarm. All of these

components require the voltage levels to be as precise as possible to ensure accurate digital communication and operation.

6.2.6 BLE PCB Schematic

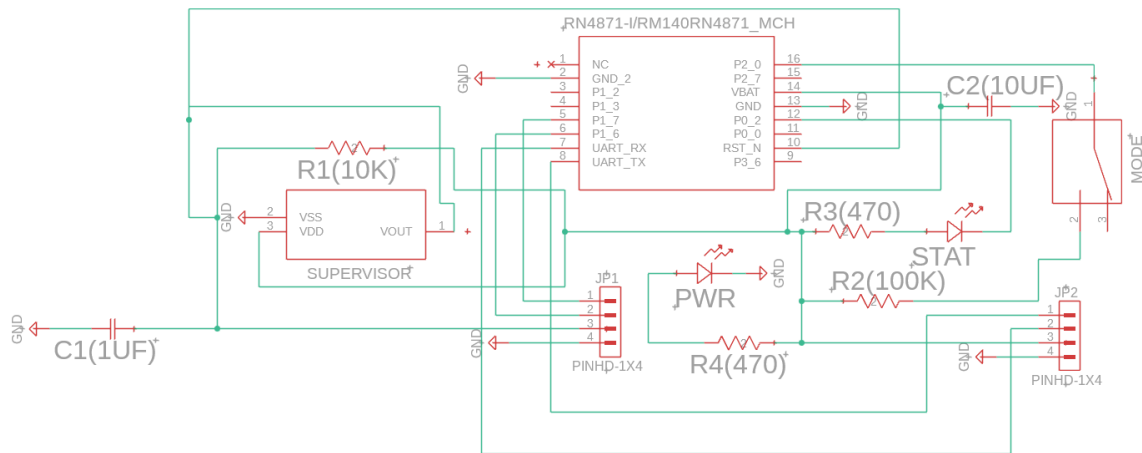


Figure 6.7: BLE PCB Schematic

This is the schematic of the custom PCB board designed to interface with the Bluetooth module in the system. In the circuit, there are the power-conditioning components that allow the module to start up reliably and operate with a stable supply. A supervisor device provides appropriate reset behavior, preventing the module from entering unstable states in periods of power fluctuation. The design also incorporates LEDs for power and status indication, giving clear visual feedback whenever the module is active or connected. A pull-down network keeps unused signals stable, while two header connectors break out the module's UART communication lines, ground, and mode-related pins for easy integration with the MSP430 and for external debugging. Also included is a mode switch, which allows the module to enter configuration mode if required. This schematic provides a clean and reliable interface for Bluetooth communication within the system.

6.3 Optical Design

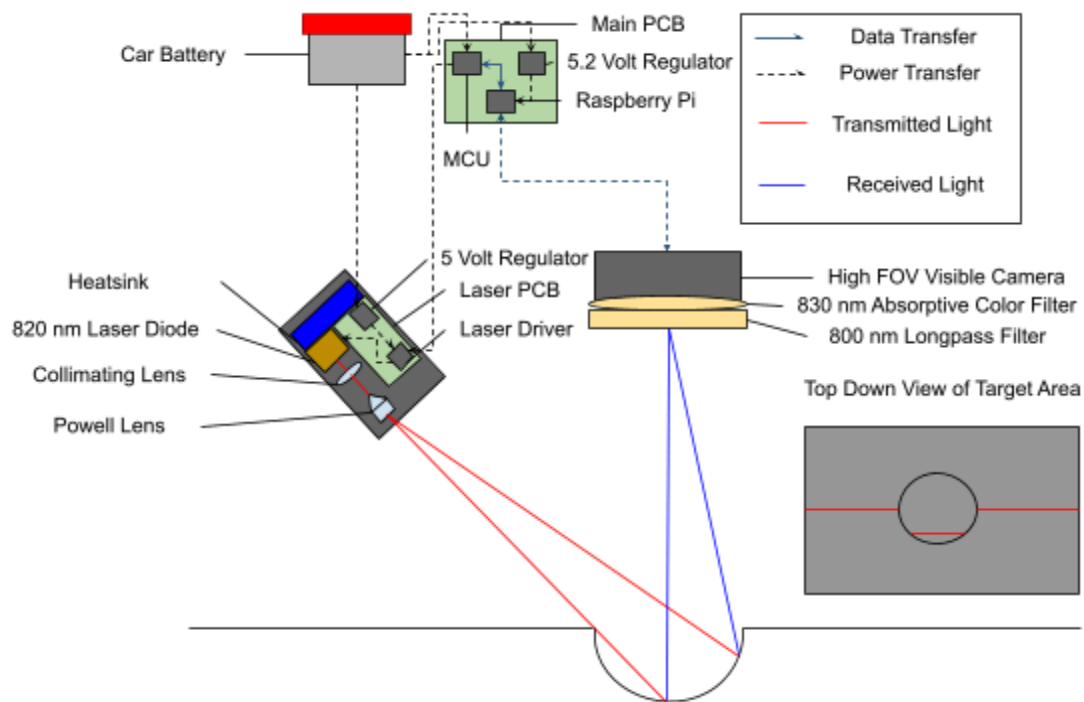


Figure 6.8: Optical Design Schematic

The optical design for this project is designed to illuminate a target area and scan said area using a camera to find disturbances in the road like potholes. Our design consists of two major parts, the illumination and detection systems.

6.3.1 Illumination System

For this project we will be illuminating a 8 ft horizontal line on the road with infrared light. In order to keep the light as bright as possible for easy detection the beam needs to be expanded as closely as possible to the size of the target area. By keeping the line as close to the size of the target area as possible there will be a minimal amount of unused light and the power density of the captured light will be greater allowing for easier detection.

Due to the unique requirements of our project a normal beam expander would not work. To properly shape the beam to fit the target area we decided upon using a Powell lens. Our Powell lens is most effective with a collimated input beam that has a diameter between 0.8 and 1mm. The laser diode we will be using is divergent and should be collimated for the purposes of our project. By

placing an aspheric lens one focal length away from the beam waist, the light will be collimated with minimal aberrations. However, due to the shape of the laser cavity in the diode, the beam's divergence is unequal in different directions, causing the collimated beam to be elliptical. While it would be more effective to have a circular beam the added cost is too much for the minimal benefit that circularizing the beam adds. The resultant beam has a diameter of 0.93mm on the fast axis and 0.49mm on the slow axis. The illuminator is detailed in figure 6.8 below.

$$\tan(\theta) = \frac{o}{a}$$

Eq. 6.1: Tangent of an angle on a right triangle.

$$D = 2f \cdot \tan(\theta/2)$$

Eq. 6.2: Collimated beam size derived from Eq. 6.1.

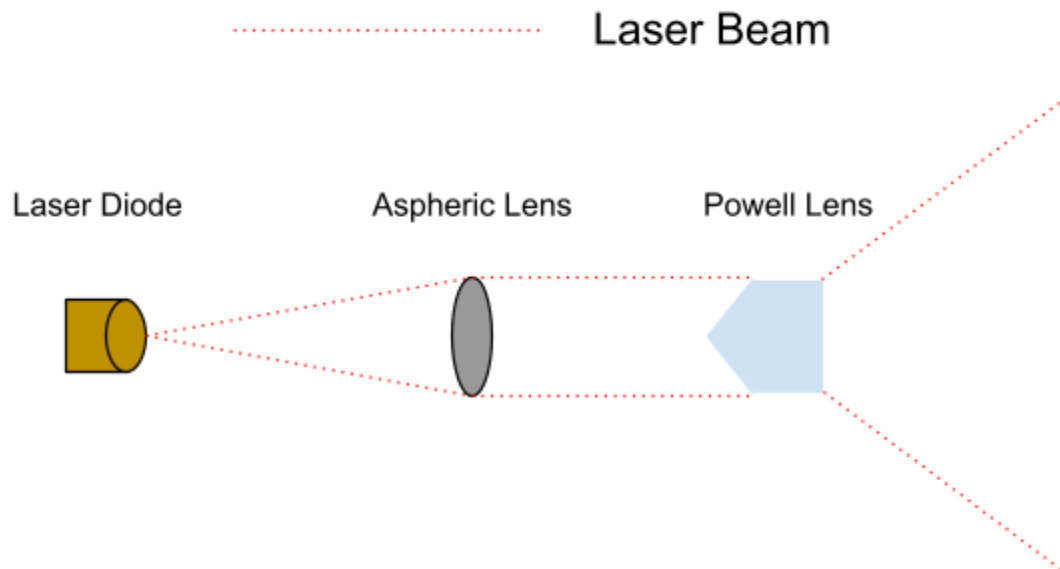


Figure 6.9: Laser diode illumination system.

6.3.2 Detection System

Similar to the illumination system, our camera needs to be able to capture the 8ft target area. From 4ft above the ground, our camera has a horizontal field of view of 90° to capture 8ft horizontally. Due to limited processing power we are only able to operate the IR camera at 50 FPS which will allow us to make a detection at least once every foot at a max speed of 30 mph. The camera we decided on captures video in black and white, which should lower the required processing power as well. Our device will be operating in the sunlight, so it is very important to keep the amount of noise to a minimum. To do this, our

detection system makes use of a bandpass filter centered on a wavelength of 830nm to match the laser diode. With this design we were able to block out enough light from the sun to ensure reliable detection in the center of target area, however due to field aberration detection near the edges was less reliable. To solve this issue we introduced an 800 nm longpass filter to block out more of the sun's light where it is more powerful, this as well as our camera's sensor naturally dropping off in effectiveness around 850 nm allowed for us to drastically reduce the amount of noise in our video and increase reliable detection. We used absorptive color filters for their effectiveness at a high angle of incidence. Figure 6.9 details the camera detection system.

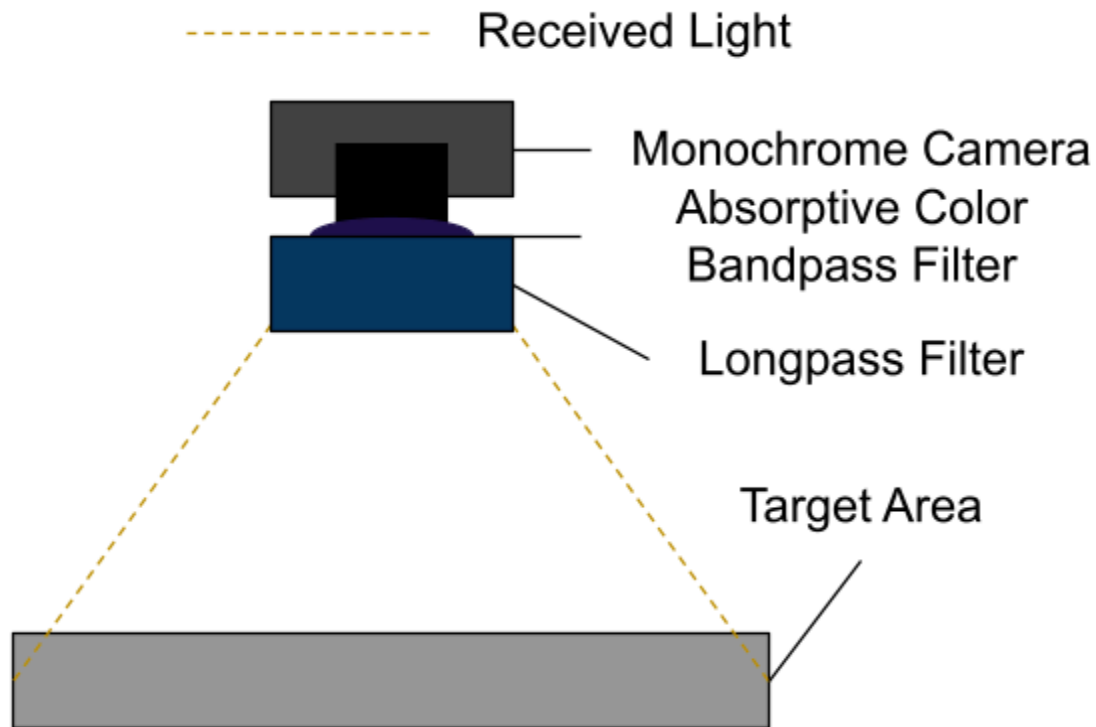


Figure 6.9: Camera detection system

6.3.3 Full System Operation

The entire optical design works in tandem to illuminate the target area and detect disturbances in the light cast over the target area. This is done in two parts. First the illuminator collimates and expands the beam from a laser diode to cover an 8 ft line on the road. The laser is positioned at a 30° angle to the road, causing the line to break whenever there is a difference in height on the roadway. Figure 6.10 shows the operation of the illuminator.

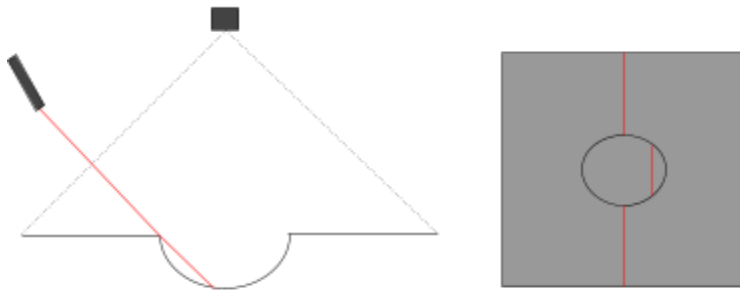


Figure 6.11: Laser illuminator incident over the target area.

Next, the detection system captures the target area head-on, allowing for the shadows to be more easily detectable. When a break is detected by the machine vision program, the disturbance is flagged and the amount of pixels between the break and the baseline is counted. Using equation 6.1, the known angle of incidence of the laser light along with the actual size of each pixel, calculated using the pixel measurements of the camera and the size of the target area, we are able to estimate the depth of the pothole. The operation of the detection system is detailed in the figures below.

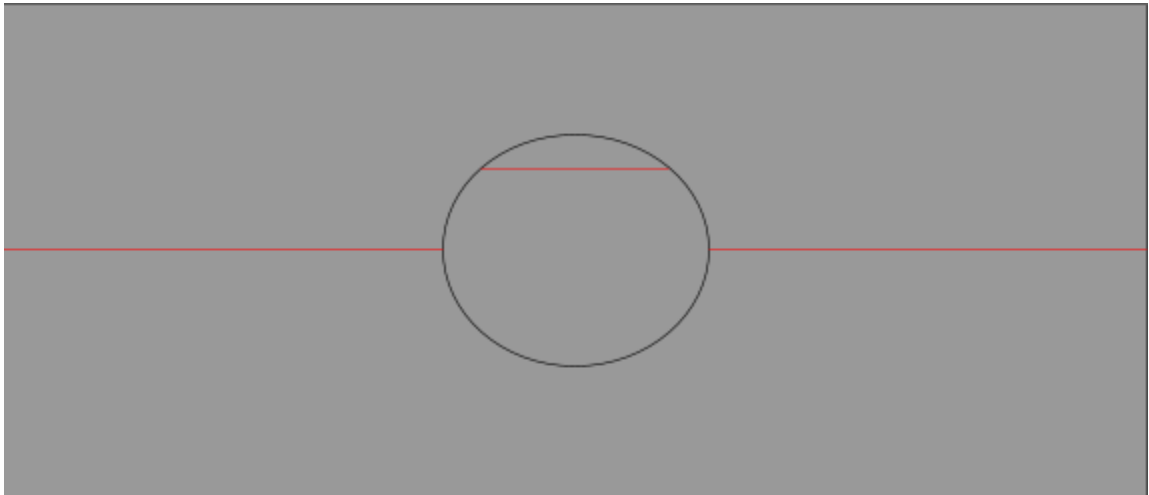


Figure 6.12: View of the target area from the camera system

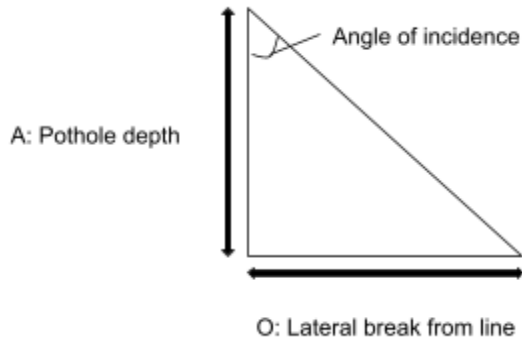


Figure 6.13: Pothole depth measurement using trigonometry.

6.4 Mechanical Design

6.4.1 Mounting for Camera and Laser Housing

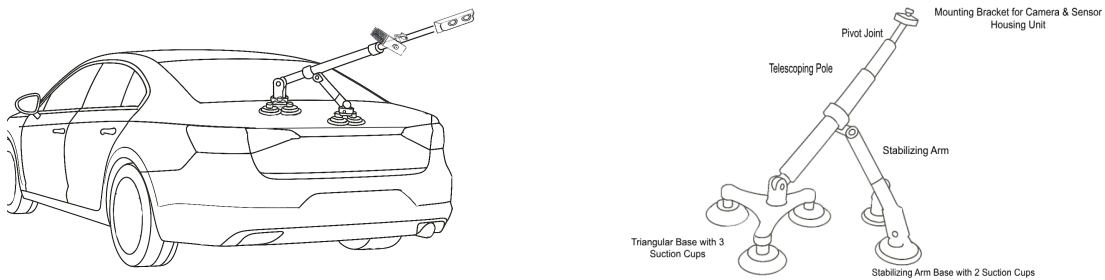


Figure 6.14: Schematics of Camera and Laser Mount

The mounting system for our project is designed to support our sensor housing unit, which is located on the vehicle exterior. The mount ensures secure attachment, weather protection, and easy adjustability. The system is non-invasive, allowing for use between various vehicle types.

The mount features a telescoping pole, allowing for length adjustment, in order to position the sensor housing at any desired height above the road surface. The telescoping function is able to accommodate many vehicle heights, while also allowing for an optimal angle of incidence for the sensors. The triangular base is made of three suction cups arranged for maximum surface contact area and optimal adhesion to the rear windshield or trunk of a vehicle. The suction cups were chosen for their flexibility, vacuum retention under vibration, and resistance to high and fluctuating temperatures.

Above the main pole is a pivot joint, allowing the camera and laser module to be pointed precisely in the direction of the road. A mechanical lock is provided to ensure that the optimal angle is set, and remains fixed under vibration or wind.

A laser module enclosure was designed to accommodate both an internal PCB and laser emitter. Within this enclosure, a precision clamp mount holds down the laser, positioning it precisely in relation to its lens assembly. The machined aluminum clamp provides passive heat dissipation and maintains the path of the beam stable under vibration or thermal excursion. This unit is attached to the pole of the mount using a clamp, allowing for easy adjustments. By making fine angular alignment adjustments during mounting, the clamp offers utmost alignment prior to securing in position, offering stable laser projection throughout use.

To improve stability structure in movement and eliminate torsional twist, a stabilizer arm branches out of the center pole and is attached to the vehicle via a two-suction cup base. The arm creates a triangulated system of bracing with a more uniformly-distributed, applied load and reduced stress at points of connection. The stabilizer arm is beneficial for resisting lateral motion and eliminating dynamic deflection as the vehicle travels over road bumps, potholes, or corners.

The majority of the structure is manufactured using 6061-T6 aluminum. This material was selected because it possesses high strength-to-weight ratio, outstanding corrosion resistance, and good machinability, making it an excellent choice for exposure to the outdoors where structural stiffness and light weight are both critical. Aluminum is also a passive thermal conductor, meaning it is beneficial in dissipating heat away from the unit.

This multi-component mounting assembly addresses the required vibration isolation, reconfigurability, and longevity for successful field deployment. It offers precise sensor alignment, rapid installation and removal, and compatibility with a wide range of vehicles to allow experimental testing as well as fleet deployment of the pothole detection system.

6.4.2 Main PCB Housing Unit for Internal Control System

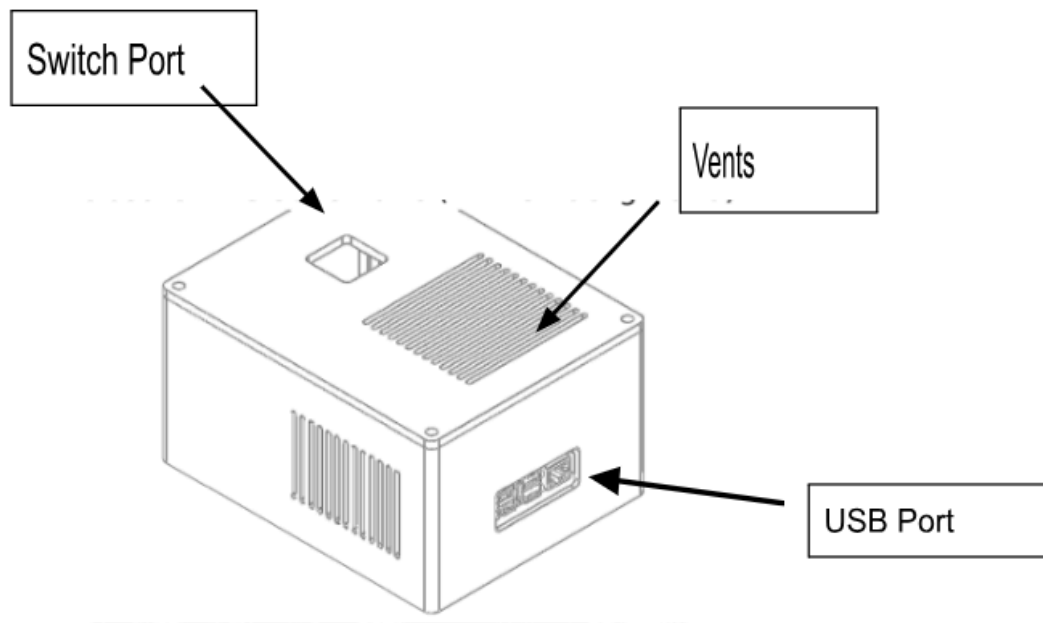


Figure 6.15: *Schematic of the Main PCB Housing Unit .*

The schematic above displays the primary internal housing unit design for the printed circuit board (PCB), acting as the central processing and power center for all of the sensors and electronics. This housing is permanently installed in the vehicle interior and provides regulated power input, wired connections, and wireless Bluetooth capability. The goal of this unit is to put all electric communication in a single protected interface, therefore making it the focal point of the system's control structure.

The housing is made in the shape of a rectangular prism with a heat resistant exterior. Its structure is reinforced by four mounting screw points in the corners, allowing it to be securely mounted on a flat mount, shelf, or internal mounting bracket if needed. These screws help prevent movement during vehicle vibrations, helping to keep connections stable and secure.

At the top of the enclosure are a series of ventilation slots that allow natural convection to dissipate generated heat away from internal electronics. Thermal regulation is necessary because this system will be actively powered when the vehicle is running and performing AI detection software and video data

processing. Passive airflow through these vents allows for consistent operation within the desired temperature range.

The housing front contains all user-controllable ports and interfaces. There are two USB ports stacked on the left-hand side: one USB 3.0 and one USB 2.0. The USB ports are used to move data between the Raspberry Pi or onboard processing hardware and the laser/camera modules. Underneath the USB ports is the Power IN port, into which 12V DC is supplied by the vehicle's accessory power system. This port connects to a voltage regulator that supplies power to all the devices downstream, including sensors and the main microcontroller, in a safe manner.

Control Interfaces and Functional Design

On the top of the unit, there is a Bluetooth pairing button molded into the housing. The button offers remote connectivity for system setup, diagnostics, or firmware updates. It is paired on the initial deployment to facilitate wireless data transmission to our companion apps and mobile devices.

Adjacent to the Bluetooth button is the main Power switch. This switch cuts power to the system entirely when in the off position, providing an easy way of disabling the device without the need to unplug any cables.

Internally, the box houses the Raspberry Pi or equivalent controller, a power distribution board, and breakout cables for external sensors. The simplicity of ports was designed to make the box user-serviceable if, in the future, changes or diagnostics have to be performed. Each port is positioned to align with the present Raspberry Pi GPIO and USB layout for direct compatibility.

Finally, all the side surfaces of the unit are featureless and closed for easy sealing and cable management. This allows wires to leave in a clean bundle from one surface, preventing potential cross interference and addressing the long term modularity goals of the project.

6.4.3 IR/RGB Dual-Camera Housing Unit

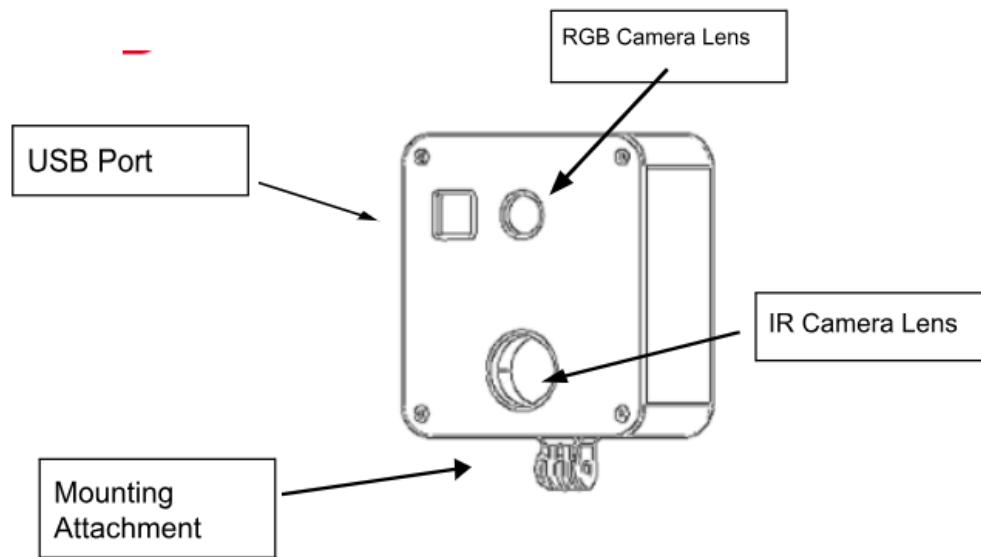


Figure 6.16: *Schematic of the Dual Camera Housing Unit*

The schematic above represents the final camera housing design of the dual camera unit that contains both infrared (IR) and RGB cameras within a vertically mounted housing unit. The camera unit is mounted externally at the back of the vehicle, pointing down towards the road surface. Its vertical mounting was chosen particularly to orient the field of view of the two cameras so that pothole detection and road irregularity detection are optimized. Unlike past horizontally arranged iterations of this design, the vertical layout in this case maintains both cameras in constant line of sight to the target area, reducing calibration errors and providing them with more synchronized data input.

The IR camera is positioned at the top of the housing to capture thermal data for pothole verification, and the RGB camera below it, takes visual data for presenting evidence to image based algorithms. This alignment mimics the traditional point of view of the pothole so that any surface deformation would be simultaneously recorded by both spectral bands. Every lens is also mounted flush within the housing faceplate to minimize the exposure of the lens and reduce the risk of environmental interference.

Routing the cables at the back isolates them from wind resistance, road debris, and direct rainfall, which are conditions typically found at the rear of a traveling vehicle. By adhering to common industry tested routing practices, the

design contributes towards system reliability over the long term and facilitates integration with other vehicle mounted electronics.

6.4.4 GoPro Compatible Camera Housing Unit Mount

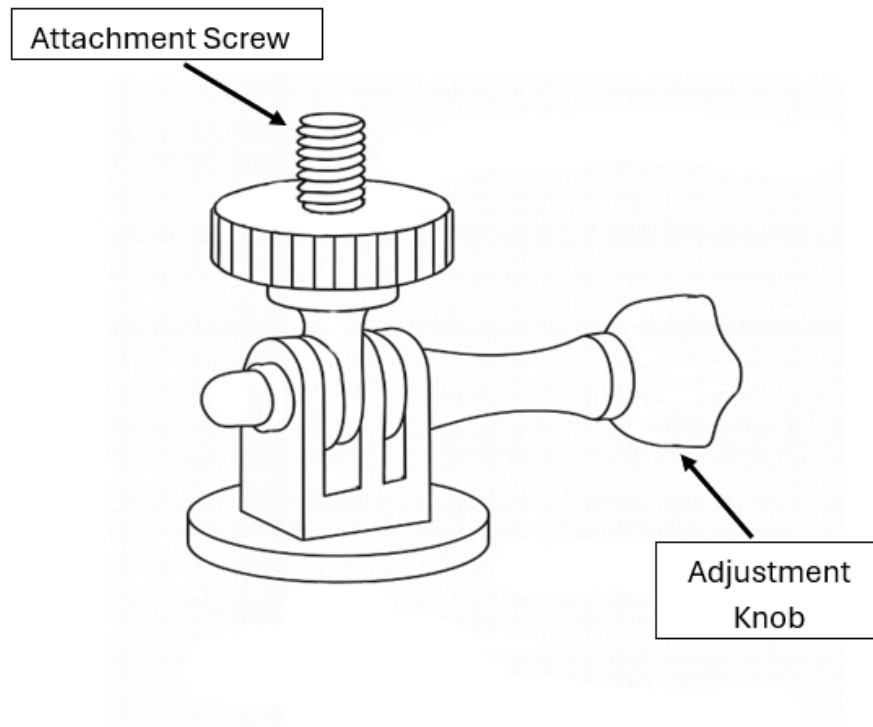


Figure 6.17: *Schematic of the Camera Housing Unit Mount*

The GoPro-compatible mount adapter, shown above, is a critical component between the sensing system and mounting surface. The mechanical assembly is designed to provide a standardized interface for camera based or sensor systems that utilize GoPro style connectors. The adapter has a base with a flat circular mounting foot, which offers broad surface contact for adhesion or mechanical fastening. A vertically mounted dual-pronged housing runs from the base to stabilize the horizontal pivot pin, making the mount adjustable over a broad range of angles. This is required for applications requiring alignment calibration, such as vision sensors or laser modules that must be precisely pointed. The included thumb screw sets the pivot in place and offers rapid angle locking in field applications without the need for specialized tools.

On top of the mount is a spinning thread platform to hold a 1/4-inch industry standard camera thread. The top platform accepts payloads like light housings or cameras, using standard threading practices of the industry for wide compatibility. The ridged ring that goes around the threaded disc allows for improved grip and tightening. The thumb screw GoPro style mount is a key element to our modularity in the project because it allows the different housing systems to be easily mounted or removed based on requirements. The rapid release of the thumbscrew and the universal thread make fast prototyping and field changes possible. The design of the mount ensures consistent alignment even in mobile or outdoor environments where vibration or motion could otherwise interfere with system performance.

6.4.5 Laser Housing Unit

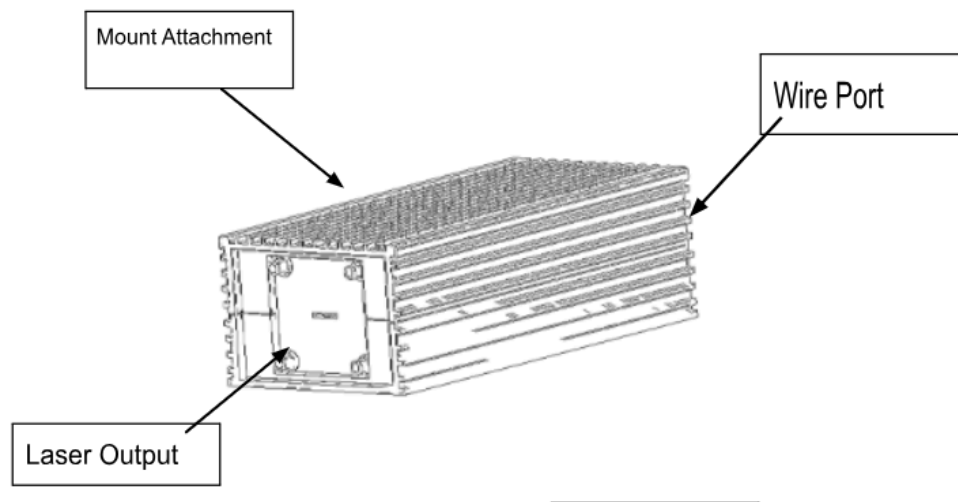


Figure 6.18: *Schematic of the Laser Housing Unit*

The schematic above displays the design of the laser housing unit, which plays a critical role in enhancing the sensing and detection characteristics of the entire pothole detection system. The device is responsible for projecting an accurate IR beam over the field of view of the camera to be used in 3D geometry estimation via triangulation and stereo image analysis. Within the enclosure are clamps to securely hold the laser and lenses in place, ensuring accurate measurement. The enclosure is cast in the form of a rectangular prism and possesses a number of key features: thermal control via exterior fins, waterproof electrical cable management, and a mechanical mounting clamp.

The most defining aspect of this design may be the existence of a heatsink structure on the side of the enclosure. This array of parallel fins is machined aluminum and has the function of passive cooling during usage. The laser diode within the case produces high-frequency IR light, which translates into heat produced by long term use. Without active cooling, the performance of the diode would begin to degrade, resulting in less reduced accuracy and longevity. By attaching aluminum fins along the side, the module offers extensive surface area for ambient air passing over it to dissipate excess heat. These fins are dimensioned to allow for air convection without sacrificing structural rigidity.

Laser Placement, Electrical Routing, and Mounting Strategy

On the face of the laser housing, a laser aperture is centered and set back in a sealed lens port for accurate beam projection and environmental sealing. The laser is mounted beneath the vertically mounted camera housing. This orientation enables the outgoing beam to travel in the same view axis as the camera, ensuring precise synchronization between visual imagery and laser depth data. With the same sensing direction, the system is able to detect surface anomalies such as potholes through both image and distance based validation.

The rear of the enclosure contains the wire routing output, where small gauge wires run through. These wires deliver power and control signals from the onboard PCB board within the vehicle cabin. The wires are routed neatly through the mounting structure and directed towards the interior of the vehicle, securely fastened to avoid slackness, tangling, or interference during movement.

Mounted to the side of the laser housing is an adjustable clamp designed to mate with the main vertical support pole extending from the trunk mount. The clamp allows for two degrees of freedom, giving installers the flexibility to calibrate the laser orientation independent from the camera unit above. The hinge system, when locked in place, resists movement under conditions of high frequency vibration caused by rough road surfaces. This mechanical isolation ensures the laser remains accurately aligned even after extended use in dynamic driving environments.

7. Software Design

7.1 Overall Software Design

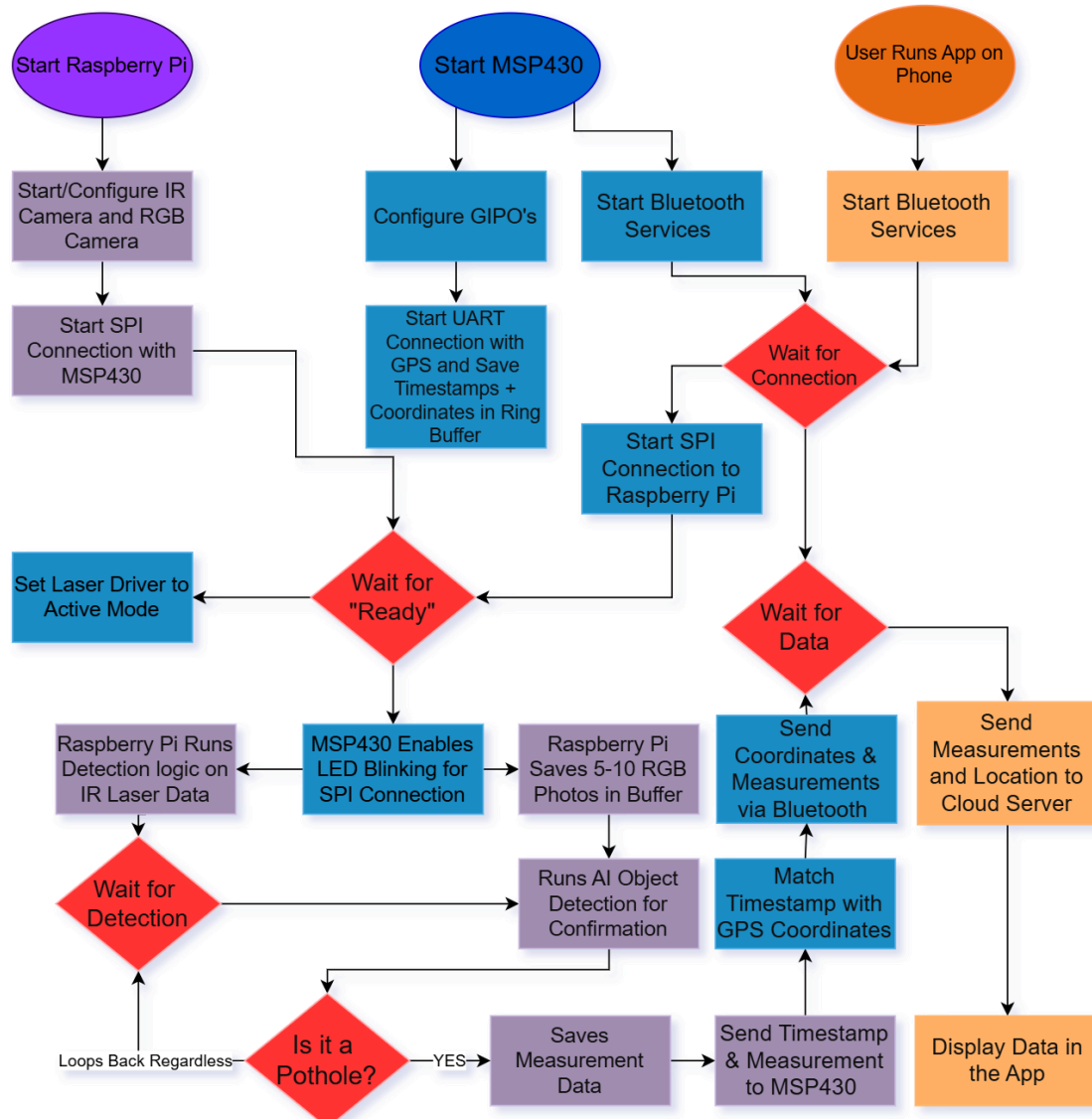


Figure 7.1: Overall Software Flowchart

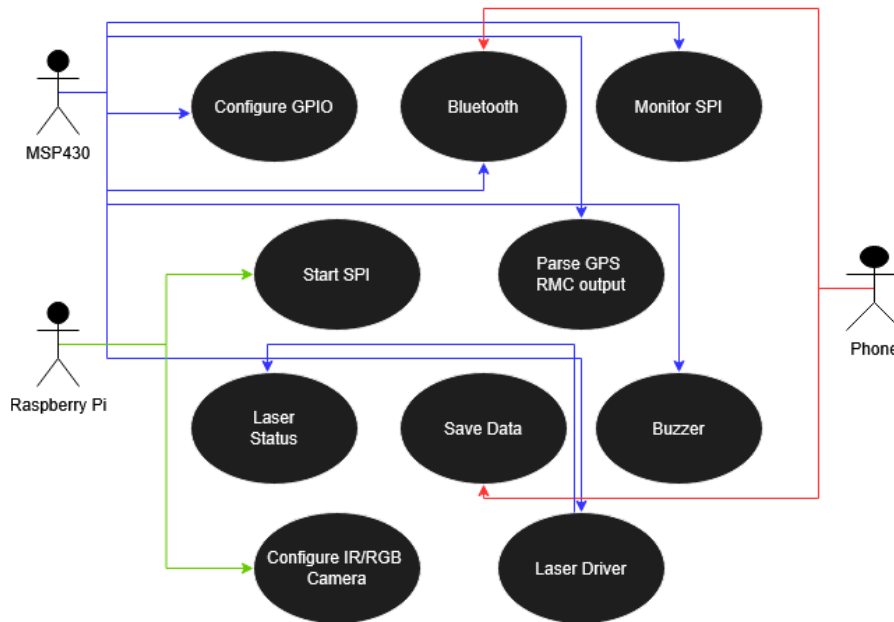


Figure 7.2: Overall Software Use Case Diagram

The use case diagram above outlines the major software functions of the pothole detection system by mapping interactions between three key components: the MSP430 microcontroller, the Raspberry Pi, and the user's phone. Each actor interfaces with distinct elements of the system and contributes to the overall functionality of detection, analysis, timestamp alignment, and wireless communication.

At a high level, the MSP430 serves as the embedded coordination layer for the system. It initializes essential peripherals, manages GPS acquisition, performs the READY-packet synchronization with the Raspberry Pi, activates the laser subsystem, and prepares formatted data packets for wireless transmission. The MSP continuously receives pothole metadata from the Raspberry Pi, aligns it with the appropriate GPS reading, and relays the finalized detection to the mobile application through the BLE module. The detailed workflow of GPS processing, timestamp handling, and packet formation is described separately in the MSP430 Software Design section.

Meanwhile, the Raspberry Pi executes its own initialization sequence. It brings the SPI interface online, configures the IR and RGB cameras, and begins filling ring buffers with incoming frames. Once the structured-light laser is confirmed active through successful SPI communication, the Pi launches the IR-processing thread to perform sheet-deformation analysis. When deformation exceeding a threshold is detected, the Pi sends the corresponding RGB frames

through a YOLOv5 inference model to confirm the presence of a pothole. If the detection is validated, the Pi computes area and depth, selects the best RGB frame, compresses it as a JPEG, packages the results into a \$PH metadata frame, and transmits both the metadata and JPEG image to the MSP430 via SPI.

After the MSP430 receives the \$PH metadata and performs its alignment operations, it constructs a unified \$PD packet and forwards the event to the paired smartphone via BLE. The MSP does not process or forward JPEG images; these remain on the Pi side. Once the \$PD packet reaches the phone, the mobile application parses and logs the detection, synchronizes it with the cloud database when connectivity is available, and renders the pothole as a pin on the map interface for drivers, technicians, and administrators to review.

7.2 MSP430 Software Design

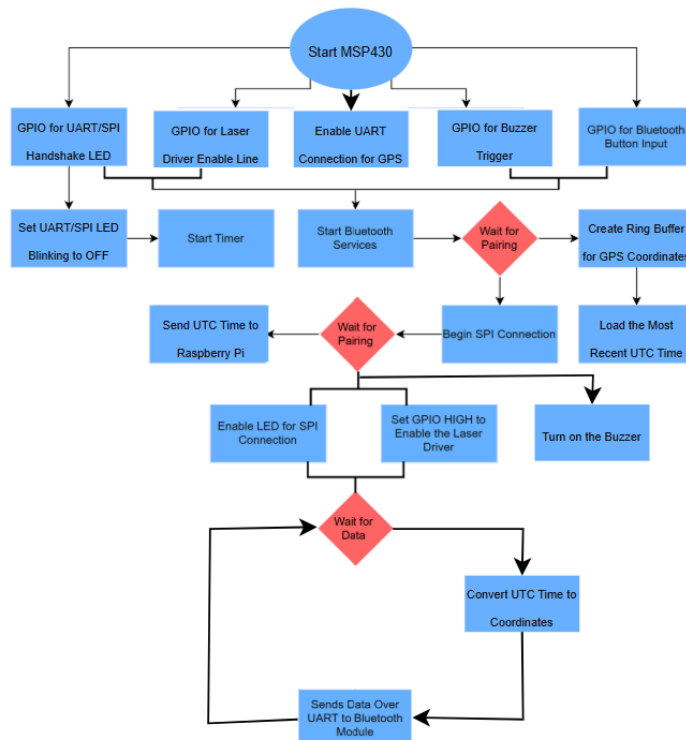


Figure 7.3: MSP430 MCU Software Flowchart

The MSP430FR2355 microcontroller acts as the central controller for system-level tasks, including clock configuration, SPI synchronization with the Raspberry Pi, GPS acquisition, BLE communication, and GPIO coordination. It manages peripheral activation, performs the READY-packet negotiation, prepares GPS-aligned pothole data for the mobile application, and maintains a

reliable communication bridge between the detection hardware and the phone interface.

When the MSP430 powers on, it first configures its system clock to operate at 8 MHz, ensuring stable timing for all serial interfaces and internal counters. Immediately after clock setup, the firmware initializes the SPI pins and enters the READY-handshake routine to establish communication with the Raspberry Pi. During this phase, the MSP430 monitors the incoming SPI stream until the Pi sends its READY packet. Once this occurs, the microcontroller responds with a timestamp frame so the Pi can align its internal clock for consistent event timing. Only after this handshake is complete does the MSP430 proceed with the remainder of its startup tasks, including loading timers, enabling GPIO lines, and preparing peripheral subsystems.

Early in the initialization sequence, the MSP430 configures the MTK3339 GPS receiver by transmitting the appropriate PMTK commands over UART. These commands disable unused NMEA sentences, switch the module to RMC-only output, and configure the GPS to operate at a 4 Hz update rate. As GPRMC sentences arrive, the MSP430 parses the latitude, longitude, and UTC fields, converts the coordinates into signed integer microdegrees, and stores each reading in a circular buffer. This buffer is used exclusively on the MSP430 to match incoming pothole timestamps from the Raspberry Pi with the nearest GPS sample, allowing the microcontroller to produce accurate, synchronized \$PD packets for the mobile application.

Once GPS acquisition and BLE module initialization are complete, the MSP430 enables the laser-driver GPIO and the SPI-status LED, indicating that the structured-light subsystem and communication link are active. The microcontroller then enters its main data-processing loop, continuously listening for \$PH packets from the Raspberry Pi. When a new pothole detection arrives, the MSP430 validates the packet contents, retrieves the corresponding GPS entry from its circular buffer, extracts depth and area information, and constructs a properly formatted \$PD frame for transmission. The resulting packet adheres to a fixed structure:

```
$PD,<area>,<depth>,<lat_uDeg>,<lon_uDeg>#
```

where latitude and longitude are expressed in signed integer microdegree format. This standardized format ensures that the mobile application receives a complete, self-contained representation of each pothole event.

Through this design, the MSP430 serves as the embedded backbone of the system, handling timing, GPS processing, hardware control, and data

packaging for BLE transmission. Its ability to maintain SPI communication with the Raspberry Pi while independently managing peripheral functions ensures consistent operation across the entire detection pipeline without relying on the Pi's processing resources.

7.3 Raspberry Pi 5 Software Design

7.3.1 High-Level Module Interaction Diagram

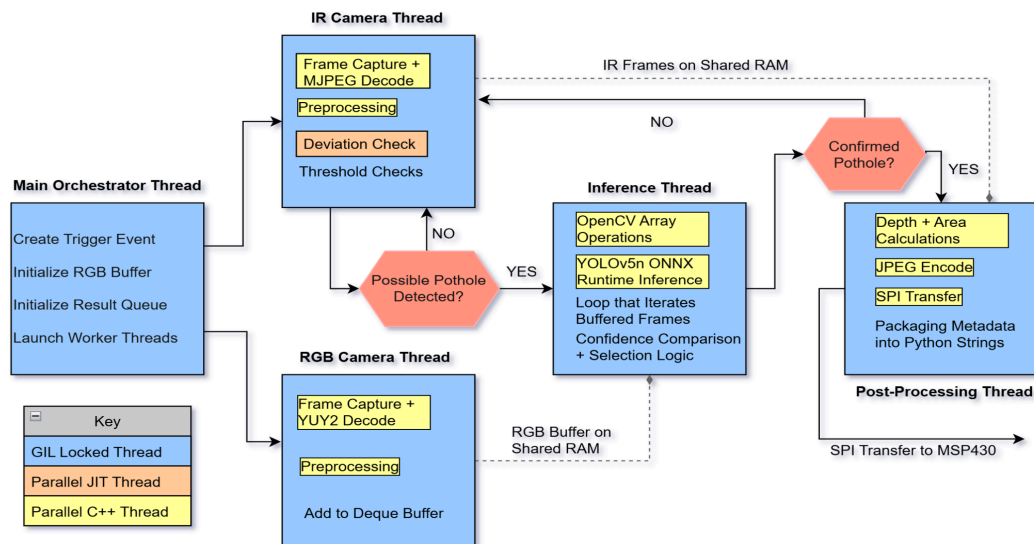


Figure 7.4: High-Level Computer Vision Block Diagram

The program is designed to run on five main threads which include: the orchestrator, infrared camera, RGB camera, inference, and post-processing thread. Each of these threads create a group of sub-threads based on their relation and time sensitivity. The five main threads all use the Global Interpreter Lock (GIL) which prevents two CPU-bound threads from running at the same time and prevents true parallelism between the main threads. For the sub-threads grouped inside the main threads, a select number of them drop the GIL allowing true parallel multi-threading. Sub-threads involving OpenCV or NumPy, for example, natively drop the GIL using C++, while the deviation check uses the Just-in-time (JIT) accelerator to achieve this. The organization of the program was carefully designed to optimize the limited computing power of the Raspberry Pi 5.

The orchestrator thread is in charge of initializing the program. It runs once at startup to create and configure all shared data structures, including a

threading event for the infrared trigger signaling, a collections deque as the RGB frame ring buffer, and a queue for passing inference results to the post-processing thread. Once an SPI connection is established in the post-processing thread, it then spins up the remaining three worker threads, remaining idle until an SPI connection is lost, where it will cleanly terminate all child threads. Since it performs only light Python logic, with no heavy loops or blocking calls, it remains entirely GIL-bound.

Dedicated to real-time, laser-sheet detection, this thread continuously grabs 1280 x 720 MJPEG frames from the IR camera and each raw frame is immediately converted to grayscale during pre-processing. All three of these calls release the GIL inside OpenCV's C++. For each frame captured, the deviation check is called to quickly look for signs of a pothole without calculating the measurements. It achieves this by thresholding each row, computes subpixel centroids in parallel across all cores, and then calculates an RMS deviation from a flat baseline. This CPU-bound task uses a JIT function, allowing it to drop the GIL. All remaining python code inside the infrared camera thread remains GIL-bound.

The RGB camera thread keeps the inference thread fresh with images to be used if the infrared camera thread flags a possible pothole. It loops at 30 FPS, calling to grab YUY2 frames, and appends each frame into the shared deque. The frame capture, YUY2 decode, and pre-processing all drop the GIL. Since the deque append is a quick Python operation, it momentarily holds the GIL, but it's lightweight enough that frame capture and decoding run at full camera speed.

The inference thread remains idle until the infrared thread flags a possible pothole. When this occurs, the RGB frames currently in the ring buffer are quickly copied to be used. For all frames, a lightweight pre-processing is done before running each frame through the YOLOv5 ONNX inference. If the inference detects potholes in the frames, it compares the confidence scores and bounding boxes dimensions to determine which image it packages into the result queue, then returns to waiting on the next trigger. The small Python loop that does the selection remains GIL-bound, but the heavy inference work executes in parallel in native code. If a pothole is not detected, it disregards the infrared camera's detection.

Whenever the inference thread enqueues a result, the post-processing thread pulls it off the results queue. First, it uses NumPy to compute the pothole's depth and area from the infrared data image. The RGB image is then compressed to convert the image to a byte array. Finally, it writes the metadata and image bytes out over SPI. All three of these calls use C++, releasing the GIL.

Any Python-only steps, like formatting the metadata string, happen under the GIL with little to no overhead relative to that of the native operations.

7.3.2 IR Camera Thread Flowchart

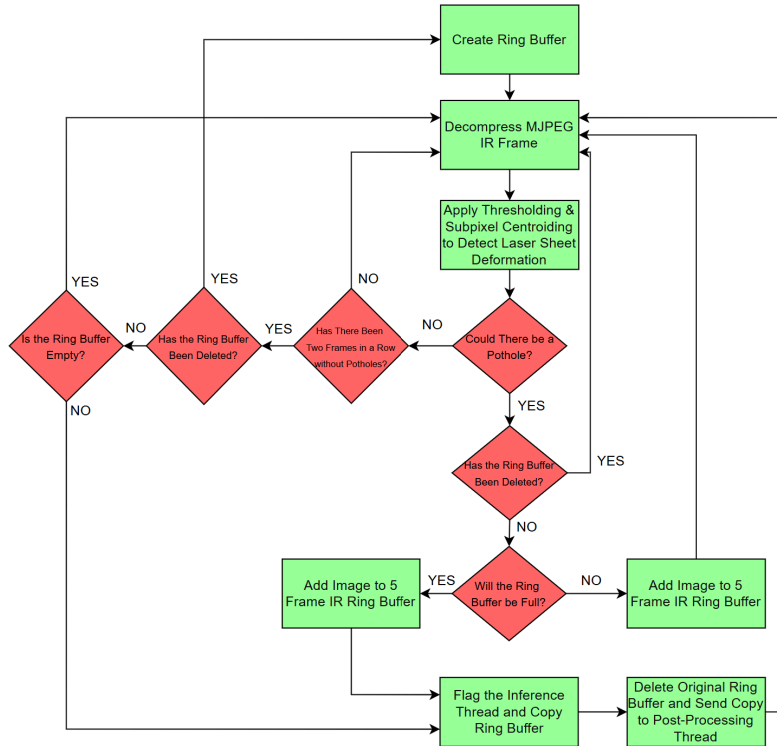


Figure 7.5: IR Camera Thread Flowchart

The IR camera thread is the one that locates potholes detection pipeline it change the real-time analysis of the incoming IR camera frames with a lightweight manner. A laser sheet detection method is used for this, where deformation of the projected laser line is utilized to find parts of the image that show potholes. The flowchart above represents the thread's logic and decision structure. The process is started by creating an IR frame buffer with a fixed 5-frame capacity, but alongside detection, the system first needs to establish a baseline profile for the laser sheet. The thread with these 20 first IR frames calculates the subpixel centroid of the laser reflection in each row and averages these centroid positions across frames to build a baseline laser profile $x_{base}(y)$. This baseline represents the expected shape of the laser line on flat pavement and serves as the reference for all future detection logic.

Once the baseline has been set up, every new IR frame is first converted from MJPEG format and then thresholded, subpixel centroided, and parabolic peak refined. The three-point parabolic interpolation of Eq. 6.3 is used to calculate subpixel peak refinement, where intensity samples (y_1, y_2, y_3) around the peak are used to compute the refined position:

$$offset = \frac{y_1 - y_3}{2(y_1 - 2y_2 + y_3)}, x_{refined} = x_{peak} + offset \quad \text{Eq. 6.3: Parabolic Peak Refinement}$$

This refined peak detection complements the centroid algorithm, which is shown in Eq. 6.4 and computes the subpixel center of mass of each laser line segment:

$$x_c = \frac{\sum_{x=a}^b x \cdot I(x)}{\sum_{x=a}^b I(x)} \quad \text{Eq. 6.4: Center of Mass [87]}$$

To determine whether a pothole may exist, the system evaluates the deviation between the current laser line and the baseline profile using the RMS deviation formula in Eq. 6.5:

$$RMS \text{ Deviation} = \sqrt{\frac{1}{H} \sum_{y=0}^{H-1} (x_y - \bar{x})^2} \quad \text{Eq. 6.5: RMS Deviation [87]}$$

where H is the number of rows evaluated. In addition to RMS deviation, the thread checks how many rows exceed a depth threshold and whether those deviations span enough rows to meet the width threshold. These conditions ensure that both the depth and width of the deformation are consistent with a pothole rather than noise. All of these calculations are JIT-accelerated using Numba to stay within the real-time 30 FPS budget.

After an algorithm detection of a potential pothole, the IR Camera Thread is the first to see if a current ring buffer is available. In case a buffer is active and it will be full after adding the frame, the thread makes a copy of the buffer, sends it to the post-processing thread, and removes the original buffer so that the same pothole is not counted twice. If the buffer is still not full, the detected frame is inserted and the thread is continuously checking other incoming IR frames. When detection happens but the previous buffer has already been removed, the system considers that the pothole has already been recorded. It does not react to new detections until two consecutive frames without potholes are detected. Such a cooldown avoids the repeated detection of the same pothole if the vehicle is slowing down or stopping over it.

The IR thread will be waiting for two non-detection frames after the buffer has been cleared. When this is the case, the system will create a new ring buffer and allow a new detection cycle to be started. The system considers this as the completion of a full event if additional pothole detections are recognized while a buffer is active. Therefore, it hands over the buffer to the post-processing thread to select the clearest frame and calculate the area and depth of the pothole. The buffer is deleted again. Such a method guarantees that every pothole is recorded with high precision, avoids repeated detections of the same pothole, and keeps the IR thread engaged with only fast, baseline-based deviation analysis while the heavier calculations are running in another place.

7.3.3 RGB Camera Thread Data Structure

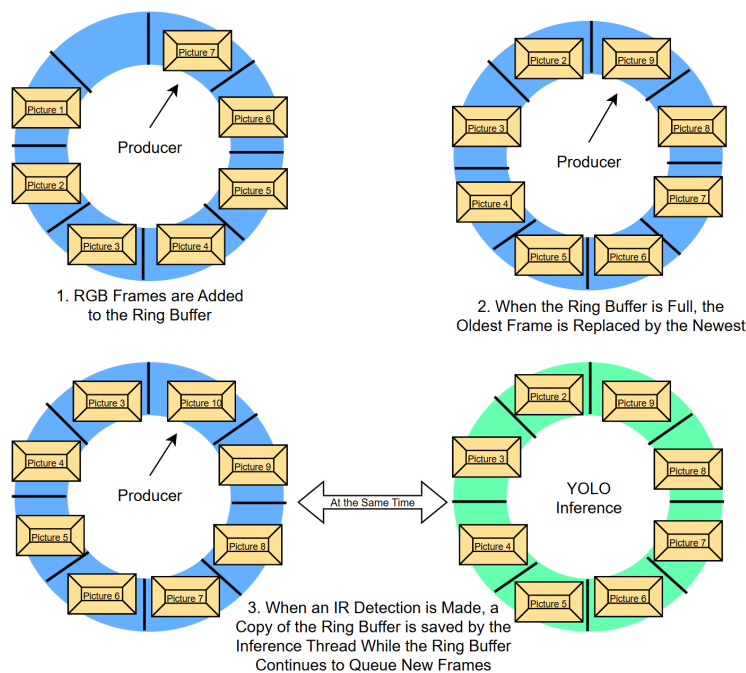


Figure 7.6: Ring Buffer Workflow Diagram

In our system, an RGB camera continuously captures frames in real time, which are temporarily stored in a ring buffer. This structure plays a critical role since the producer (RGB thread) runs at a much faster speed than the consumer (inference thread). This enables the system to maintain real-time performance while making sure that the YOLO inference model always uses the most recent image data. The figure above illustrates the ring buffer mechanism across three stages, which include: initial frame population, buffer addition and replacement, and inference triggered by IR detection.

The RGB thread acts as a producer, capturing video frames at around 30 frames per second. These frames are immediately added to the ring buffer, which has a fixed maximum length of 8 frames in this example. The buffer behaves as a circular queue; once it reaches capacity, each new frame replaces the oldest frame. This makes sure that the buffer always contains the most recent 8 frames without growing continuously or requiring manual memory management. Using this method provides a time constant of $O(1)$ for both add and pop actions. This data structure is perfect for situations where a set of historical data is needed while discarding anything beyond a set threshold.

Parallel to the RGB thread, an infrared thread monitors for visual deviations that may indicate the presence of a pothole or road anomaly. Upon detection, it signals the inference thread to begin processing. The inference thread operates at a slower speed than the RGB thread, making real-time inferences unreliable. Real-time inferences would likely cause most of the buffered frames to get skipped, making it much less likely for a pothole to get detected. To address this, a thread-safe copy operation is used to very briefly lock the buffer while a copy is made. The ring buffer will then continue its normal operation as the inference thread looks for potholes in the copied ring buffer. This allows the ring buffer to keep a continuous ring buffer in operation while allowing the inference thread to work at slower speeds without losing frames.

7.3.4 Inference Thread Flowchart

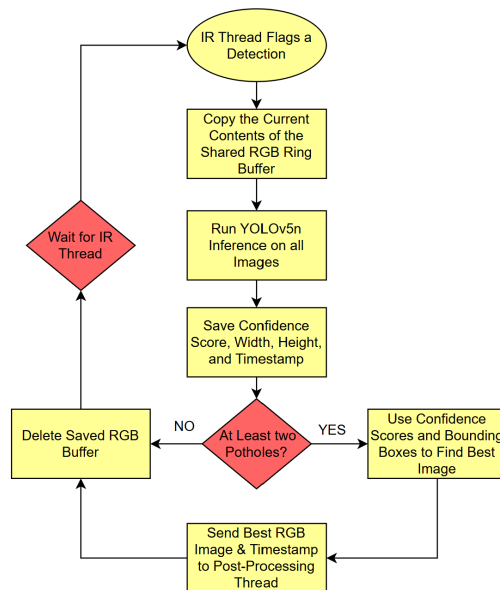


Figure 7.7: *Inference Thread Flowchart*

The Inference Thread waits until it is triggered by the IR Camera Thread, which flags a possible pothole event. Once flagged, the Inference Thread then quickly copies the current contents of the shared RGB ring buffer. This buffer contains a series of recent frames captured by the RGB Camera Thread, allowing the system to look at the visual context surrounding the IR-detected deformation. After copying the buffer, the Inference Thread runs YOLOv5n object detection on each of the saved RGB frames. Each inference result includes a confidence score, width, height, and timestamp for any detected pothole objects. These metrics are saved to be used in a selection routine.

Once all frames have been processed, the thread checks whether at least two potholes have been detected in the set of buffered images. This makes sure that general noise or minor objects do not lead to false positives. If fewer than two potholes are found, the event is discarded, and the system resets to wait for the next IR trigger. If at least two potholes are detected, a weighted algorithm is used for each frame by normalizing the area of the bounding box, multiplying it by two, and then adding it to the confidence score. This prioritizes images that not only have a high confidence but also capture a large visible portion of the pothole, indicating better framing and detail. The frame with the highest score is selected as the best image. The best RGB image, along with its attached timestamp, is then sent to the Post-Processing Thread. After this transfer, the RGB buffer used for the inference is deleted to free memory, and the thread returns to its waiting state for the next detection from the IR Thread. This strategy makes sure that efficient use of resources are met while maintaining high accuracy, filtering out noise, and prioritizing clear images to be sent over Bluetooth.

7.3.5 Post-Processing Thread Flowchart

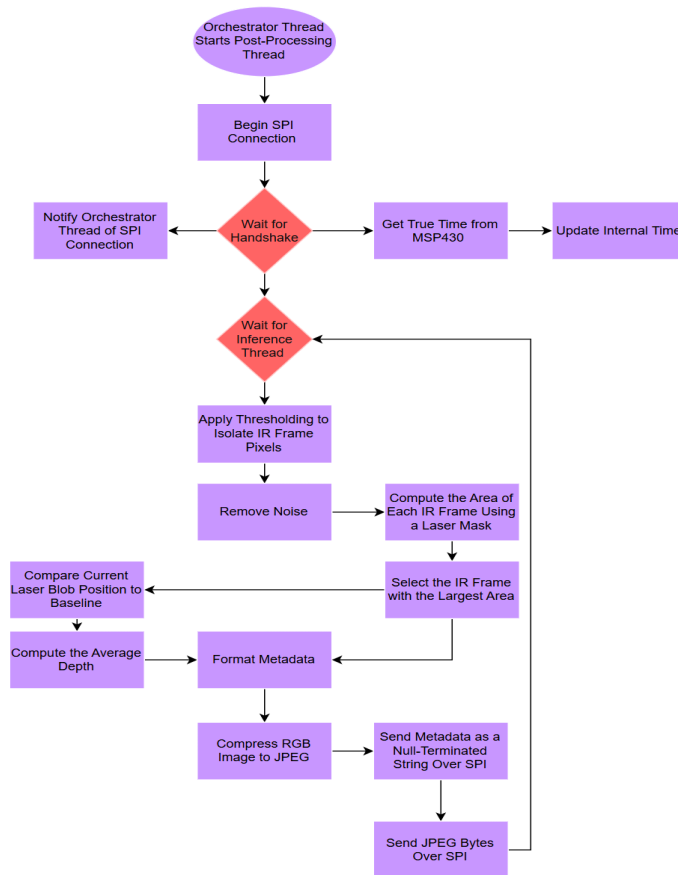


Figure 7.8: Post-Processing Thread Flowchart

The post-processing thread is responsible for handling the last stage of the pothole detection code. It performs many important tasks, including SPI communication, infrared area and depth calculations, image compression, and data transmission. The thread is first launched by the orchestrator thread and is the only thread allowed to communicate with the MSP430 microcontroller. This makes sure that timing and metadata remain centralized and consistent between the Raspberry Pi and the user's phone. Upon initialization, the post-processing thread begins by establishing an SPI connection with the MSP430. It waits for a successful connection to complete before continuing. Once connected, it receives the real-world time from the MSP430, which gets accurate time data from the user's smartphone. This timestamp is used to line up all pothole detections with real-world GPS time for backtracking. The internal clock on the Raspberry Pi is updated to this time, and the orchestrator thread is then notified to start the remaining processing threads.

Once the post-processing thread receives an RGB frame and its matching set of IR frames from the inference thread, it begins looking through the images. Each IR frame is first thresholded to isolate the laser line pixels along a single fixed horizontal line where the laser is expected to appear. To clean up the result, small blobs and noise are removed using basic image processing steps like erosion or a minimum pixel count filter. The cleaned IR ring buffer is then used to calculate the pothole's width in frame. The frame with the largest measured width is then chosen for the depth calculation, which saves time and ensures only the clearest result is used. For width, the number of bright pixels is counted and converted to inches using a scale factor from calibration, as shown in equation 6.5 shown below.

$$Area_i = \sum_x \sum_y L_i(x, y)$$

Eq. 6.6: Area of Region [88]

The frame with the largest laser-deformed area is selected for further depth analysis. The system compares the vertical position of the laser blob in this frame to a pre-established baseline laser sheet image recorded over flat pavement. By computing the average vertical offset between the baseline and the current deformation, the thread estimates the pothole depth. Both formulas are shown below as equations 6.6 and 6.7. This approach allows for rough but fast estimation of both pothole dimensions using only passive IR frame analysis.

$$\bar{y} = \frac{\sum_x \sum_y y * I(x, y)}{\sum_x \sum_y I(x, y)}$$

Eq. 6.7: Vertical Centroid Profile [88]

$$\Delta h = \frac{1}{W} \sum_{x=1}^W (B(x) - D(x))$$

Eq. 6.8: Average Depth Offset [88],[89]

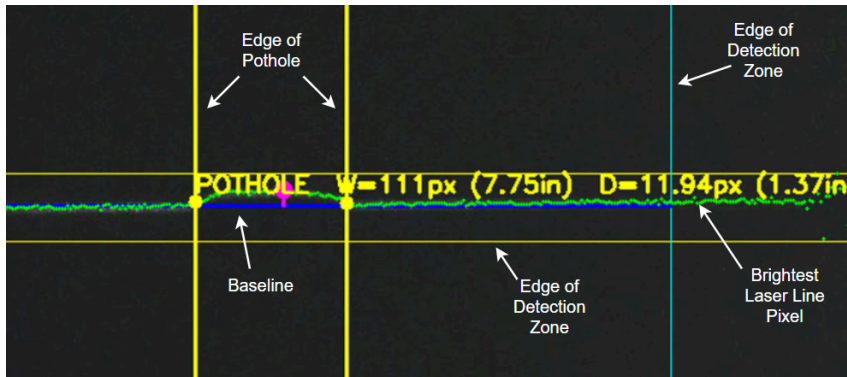


Figure 7.9: Example of Pothole Detection and Measurement

Once calculations are complete, the post-processing thread formats a metadata string including the timestamp, depth, and area. This metadata is sent as a null-terminated string over SPI to the MSP430. Afterward, the RGB frame is compressed into a JPEG image and transmitted in raw byte format over the SPI connection to the MSP430. This thread ensures all detection events are timestamped, measured, and packaged efficiently for transmission with minimal CPU overhead.

7.4 App Software Design

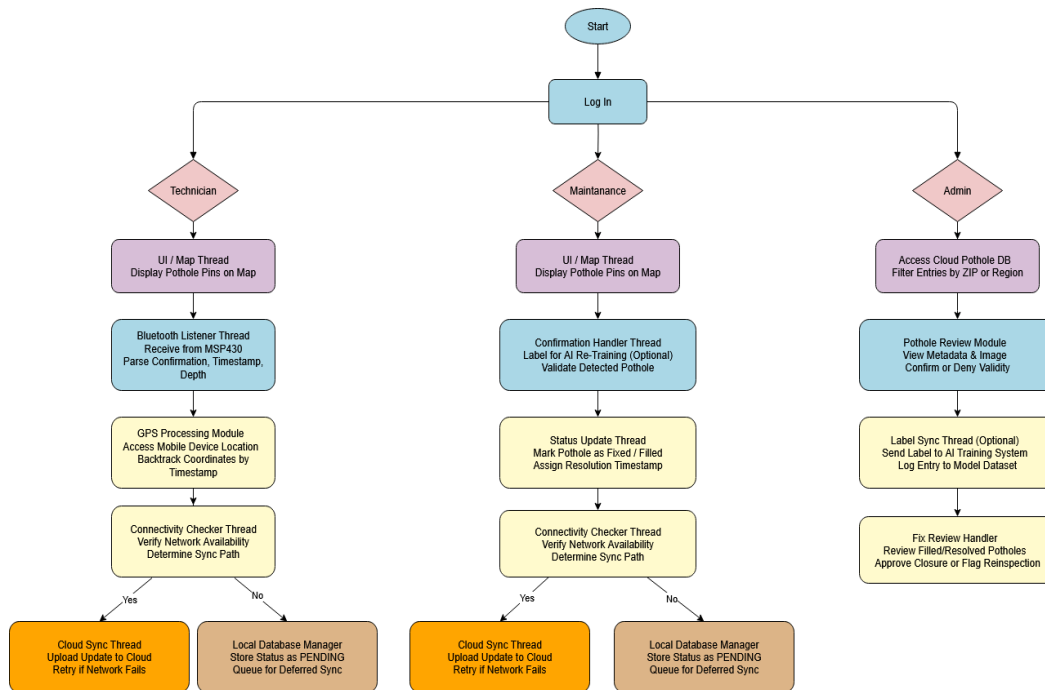


Figure 7.10: Mobile App Block Diagram

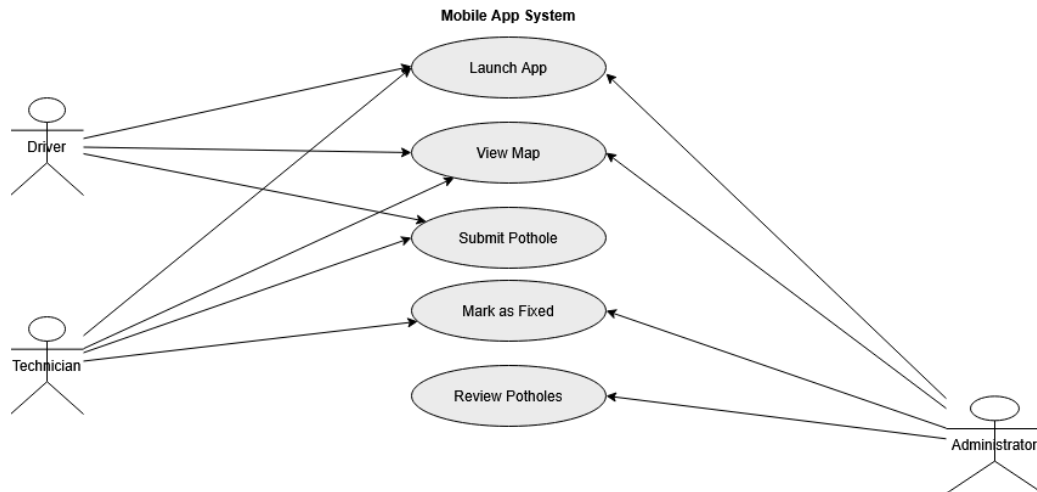


Figure 7.11: Mobile App Use Case Diagram

The mobile application is built around a multi-role architecture that separates functionality between three user types Driver, Technician, and Administrator using role-based access control implemented through JWT tokens stored securely on the device via AsyncStorage. When users launch the application, they are presented with a unified login screen that authenticates credentials against an Express.js backend, decodes the JWT to determine the assigned role, and routes them to the appropriate interface using React Navigation’s Native Stack Navigator. Each role maintains its own screen stack and state context to ensure clean separation of privileges and data flows across the application.

For Drivers and Technicians, the application centers on handling hardware-originated pothole detections and visualizing them on an embedded map view. The system establishes Bluetooth Low Energy communication with the MSP430 subsystem using the react-native-ble-plx library, enabling the app to subscribe to live \$PD packets generated by the Raspberry Pi’s AI detection pipeline. The map interface is rendered through React Native Maps, leveraging Apple Maps on iOS to display real-time pothole markers derived from MCU-provided GPS coordinates. Coordinate conversion and metadata extraction are handled through JavaScript utilities that parse the incoming BLE packets and update application state accordingly. Network connectivity is monitored using the @react-native-community/netinfo module, which determines whether detections should be posted immediately to the Express REST API via Axios or cached locally for later upload. In offline environments, detections are serialized to JSON and queued in AsyncStorage as part of an offline-first buffer, ensuring that all events are preserved until connectivity is restored.

The Technician interface additionally supports workflows for reviewing and updating pothole entries. Tapping on a marker or list item triggers an edit screen where depth, area, and repair status can be modified. Status updates such as marking a pothole as resolved or requiring further verification are committed through asynchronous Axios requests that attempt immediate synchronization with the server. When connectivity is unavailable, updates are added to the same offline queue for later processing. Each action is timestamped using JavaScript's new `Date().toISOString()` to maintain consistent temporal metadata across clients and backend systems.

Administrators are provided with a more comprehensive management interface with access to aggregated, cloud-stored pothole data. The backend exposes filtered collections through Express middleware, utilizing MongoDB's geospatial querying capabilities including `$geoWithin`, `$near`, and 2dsphere-indexed coordinate fields to identify potholes within specific ZIP code regions or custom boundaries. The React Native frontend renders these results using `FlatList` for efficient virtualization of large datasets, while detail screens display extended metadata retrieved directly from MongoDB documents. Administrative actions, such as confirming or denying detections or reviewing technician-marked repairs, are processed through atomic update routes on the backend. Consistency between locally cached data and cloud-stored records is maintained through timestamp-based conflict resolution and incremental synchronization logic.

The application adopts a robust offline-first architecture where Axios attempts all network requests with configurable timeouts, falling back to local persistence when a request fails or when `NetInfo` reports a disconnected state. The offline queue is implemented as an array of JSON-encoded operations stored in `AsyncStorage`. A background sync routine invoked whenever connectivity is restored iterates through queued entries, deserializes the operations, and replays them against the Express API endpoints. The backend uses the native MongoDB Node.js driver for database operations, with geospatial indexes enabling efficient lookup and update operations for location-based data. React Native Maps renders location markers dynamically using the latest dataset retrieved from the backend, and the entire interface features comprehensive error handling to ensure graceful degradation during intermittent connectivity. This design ensures that the application remains fully functional across real-world mobile conditions while maintaining data consistency and reliability across all user roles.

8. System Fabrication

8.1 PCB Design

In this project, PCB design is one of the most important aspects of the development of the pothole detection system. It serves as the physical foundation that brings all of the electronic components together onto a functional platform. A well-designed PCB not only ensures the correct electrical connectivity between components, but it is also a big part of signal integrity, power distribution, and overall system reliability. It also helps reduce the overall size of the system, making it easier to mount and protect within the vehicle.

This project includes multiple subsystems, such as power regulation, microcontroller logic, communication modules, and devices drawing a large amount of current, and the PCB orchestrates their interaction in a compact format. Careful planning of the layout also helps reduce potential issues like overheating or signal interference. By keeping everything organized and efficient, the PCB plays a key role in making sure the system runs smoothly in real-world conditions.

8.1.1 Main PCB Design

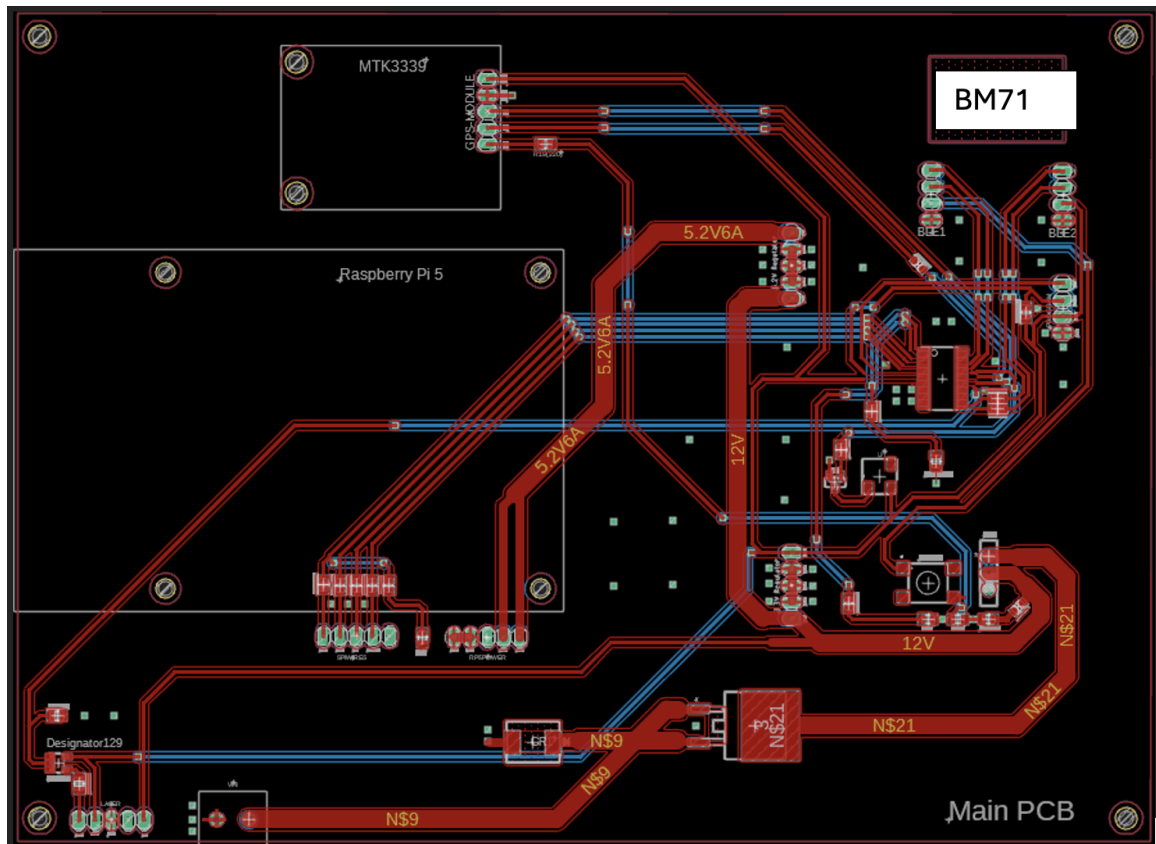


Figure 8.1: Main PCB Layout

This image shows the layout of the main PCB for the pothole detection system, which is responsible for integrating all critical, low-voltage control and communication components. The red traces represent the connections on the top layer of the PCB, while the blue traces show the connections on the bottom layer of the PCB, which demonstrates a two-layer board design.

The main components of this PCB, such as the MSP430FR2355 microcontroller, RN4871 Bluetooth module, 3.3V regulator, and 5V regulators, are placed in a way to optimize signal routing and minimize interference. The trace width in most places is 16 mm, with the largest width being from the 12V DC input in the upper right hand corner. These widths are subject to change as the project progresses into further stages.

The design of the main PCB prioritizes functionality. The design team believes the power traces are routed with sufficient width to support the expected current draw from subsystems like the laser driver, and signal traces are routed

to minimize cross-talk. Ground planes and decoupling capacitors have been placed near sensitive integrated circuits such as the microcontroller and communication modules, in order to decrease the amount of electrical noise to create stable operation. Multiple pin headers are used for programming, laser driver connections, and interfacing with the Raspberry Pi 5 via SPI. Overall, this PCB design will be the pillar of the system by supporting the sensing, processing, and communication requirements that are required by the user.

8.1.2 Laser PCB Design

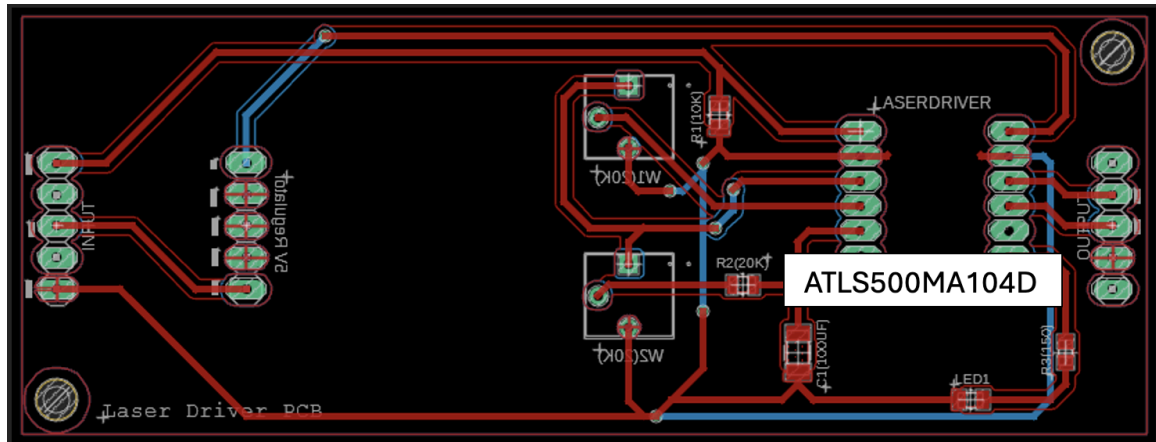


Figure 8.2: Laser Driver PCB Layout

The image shown above is the PCB design for the secondary board, which hosts the laser driver. This board is responsible for precision control of the laser diode by using the ATLS500MA104D analog laser driver. Power distribution on the board is marked with 5V traces and ground traces, which are routed with the intention to prevent thermal and electrical noise. Wide power traces help accommodate the high current demands of the laser diode, while decoupling capacitors and inductors are positioned to reduce ripple and stabilize the voltage across the board.

8.1.3 5.2V Regulator PCB Design

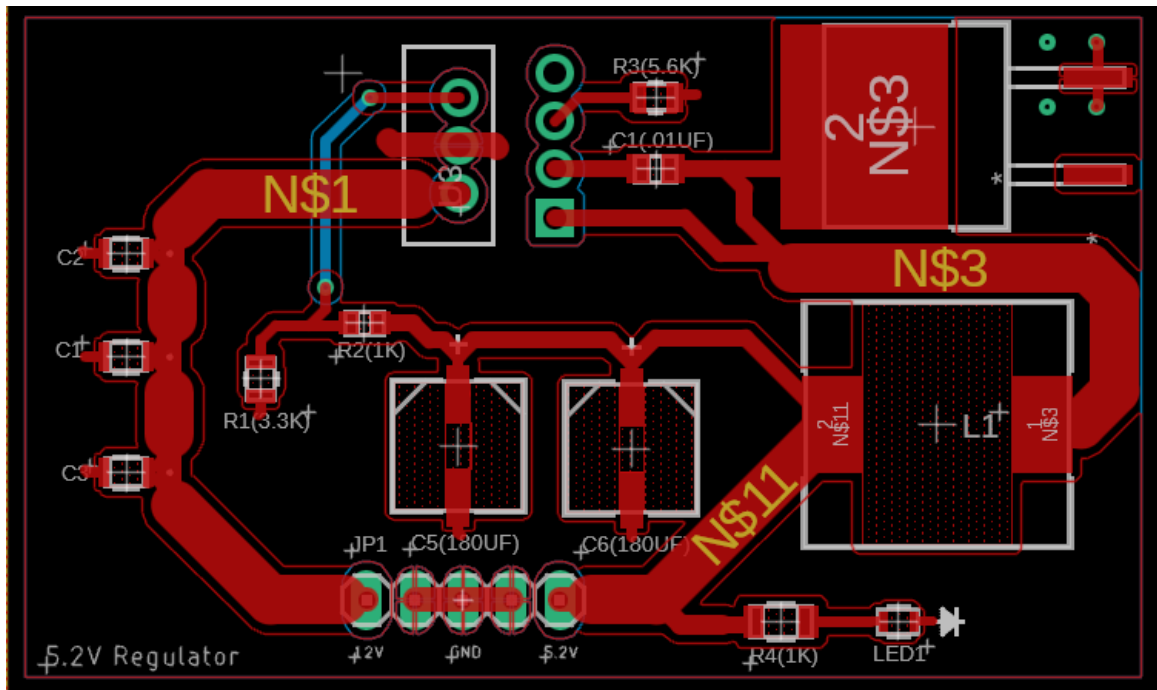


Figure 8.3: 5.2V Regulator PCB Layout

Shown in Figure 8.3 is the 5.2V regulator which was designed specifically for the Raspberry Pi 5, which needed about 5.2V as well as up to 5A in some cases. This regulator board features an inductor, capacitor, resistors, an LM2679-ADJ integrated circuit, and headers for connection.

8.1.4 5V Laser Regulator PCB Design

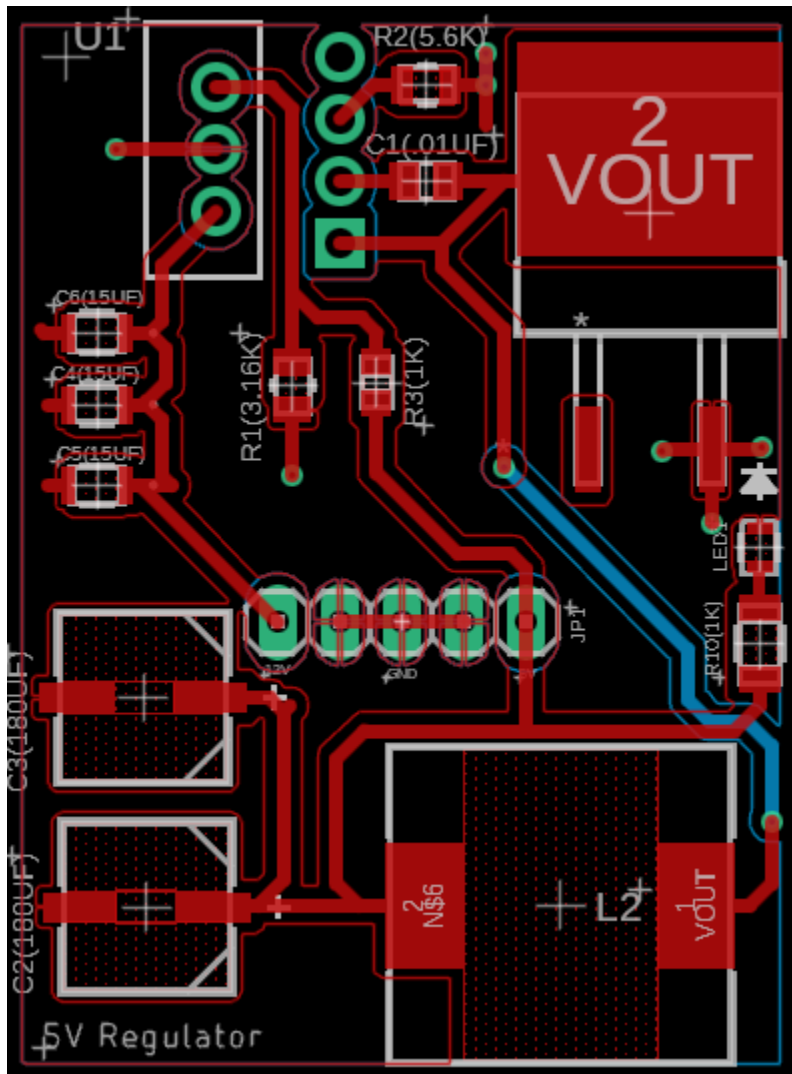


Figure 8.4: 5.2V Laser Regulator PCB Layout

Shown in Figure 8.4 is the project's 5.2 V regulator specifically designed to supply the laser driver with the correct amount of power. This regulator features an inductor, capacitors, resistors, headers for connection, and an LM2679-ADJ integrated circuit.

8.1.5 3.3V Regulator PCB Design

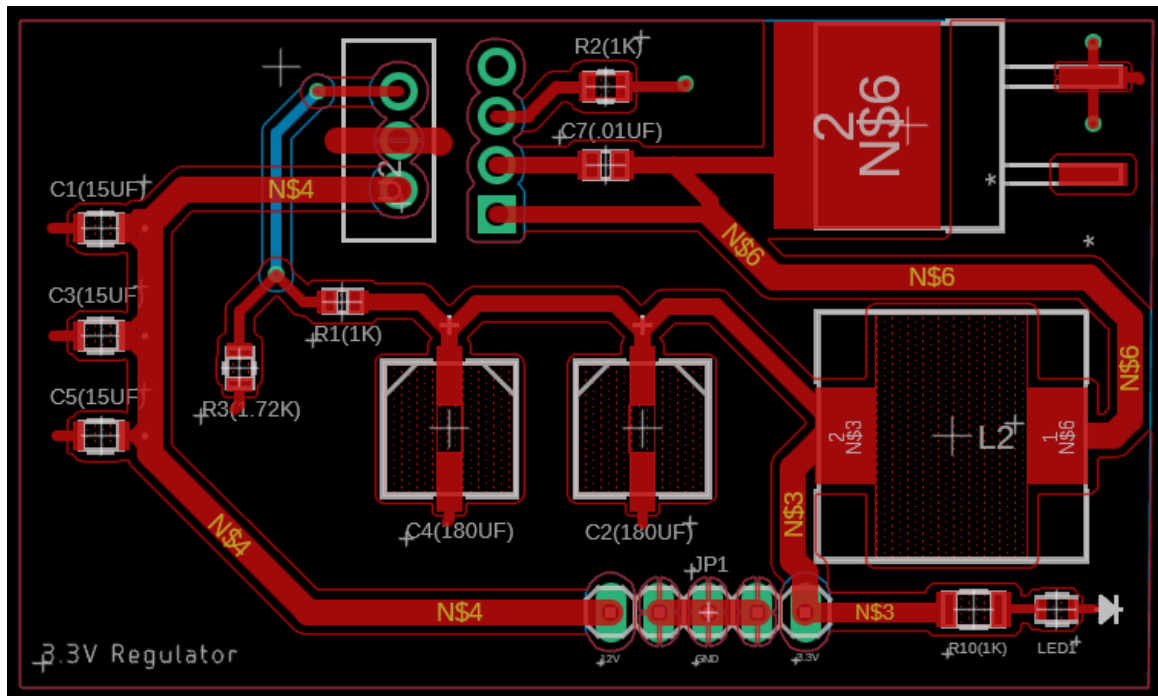


Figure 8.5: 3.3V Regulator PCB Layout

Shown in Figure 8.5 is the project's 3.3V regulator design which features an inductor, capacitors, the LM2679-ADJ regulator integrated circuit, resistors, and male headers to connect it to the main pcb, to allow the correct amount of power to the corresponding peripherals.

8.1.6 Bluetooth PCB Design

Shown in Figure 8.6 is the Bluetooth pcb design. This board features the BM71 Bluetooth module, a switch, two sets of headers to connect to the rest of the system, as well as multiple capacitors and resistors, and a supervisor.

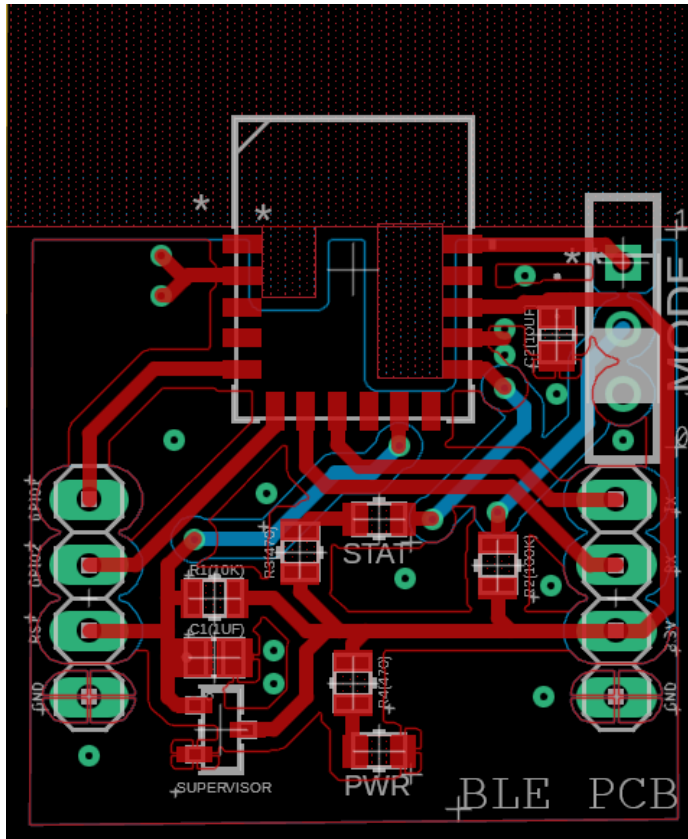


Figure 8.6: Bluetooth PCB Layout

8.2 Assembled PCB

8.2.1 Main PCB

Shown in Figure 8.7 is the main pcb of the project. Displayed here are two voltage regulators (3.3V and 5.2V). Also shown on this board is the GPS module, Bluetooth module, DC buzzer, reset button, MSP430 MCU, and the Raspberry Pi 5.

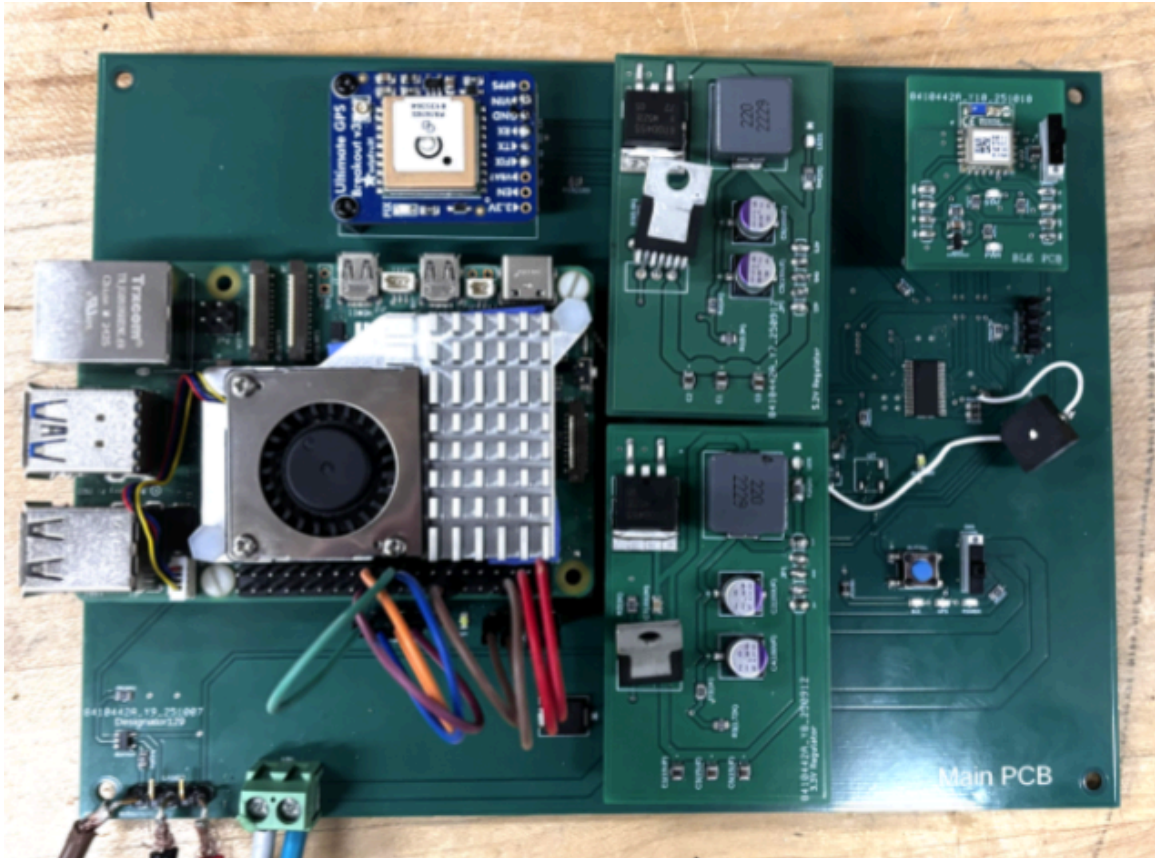


Figure 8.7: Assembled Main PCB

8.2.2 Laser Driver PCB

The laser driver pcb, shown in Figure 8.8 features a 5 V regulator, the laser driver, two potentiometers (one for adjusting current limit, one for adjusting real-time current), and male headers which lead to the laser diode.

The laser driver we selected, the ATLS500MA104D could handle 5 V of input voltage, and 500 mA of current. Which is why the design team chose to use a 5 V regulator. The potentiometers are there to adjust the current for the laser diode, which the design team found the limit to be around 340 mA of current. It was very important to keep the current under 340 mA, which is where the potentiometers were very important in this design.

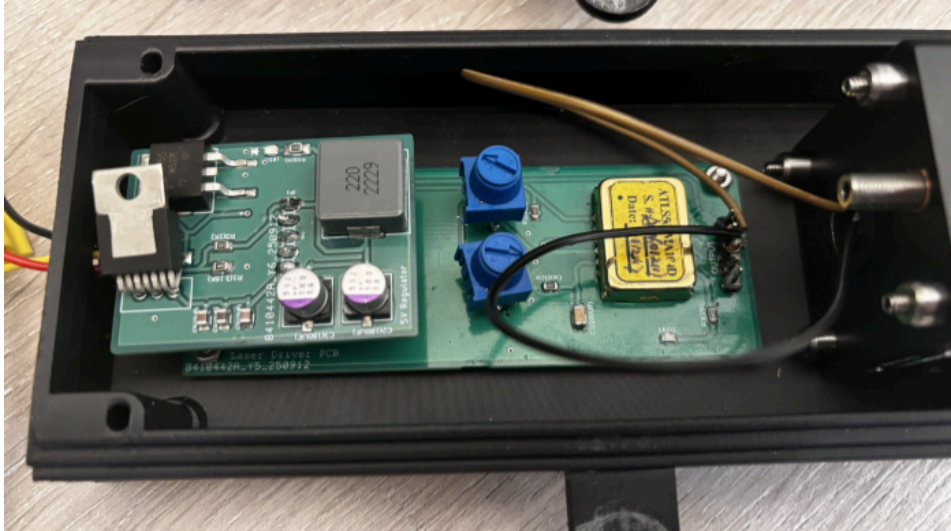


Figure 8.8: Assembled Laser PCB

8.3 Mechanical Systems Fabrication

8.3.1. Mounting System



Figure 8.9: Full Assembled Mounting Arm

The main structural support for the whole pothole detection assembly is the mount system, which secures all sensing components to the rear of the vehicle. It achieves both stability and the required standoff distance behind the car by using a multi suction cup design and a rigid extension pole. This mounting minimizes interference from the vehicle body while providing an unobstructed view of the road surface for scanning. With adjustable joints that allow the fine tuning of camera and laser orientation, the added elements like silicone tape, foam insulation,

and tuned mass weights will reduce the mechanical oscillations through the pole from motion. It provides a very stable platform necessary for the collection of data on the roads.

8.3.2 Camera Housing



Figure 8.10: Printed Camera Housing

The camera housing shields the cameras from environmental exposure, while maintaining precise alignment of the optical system. It is 3D printed from ABSr to provide a rigid, lightweight enclosure with dedicated openings for the RGB and IR cameras and alignment optics. The interior has mounting posts and alignment tabs that position the camera in a fixed location, resistant to rotational or lateral movement under operation. The housing is fitted to the extension pole via a GoPro style mount to facilitate quick installation while providing robust structural support.

8.3.3. Laser Housing

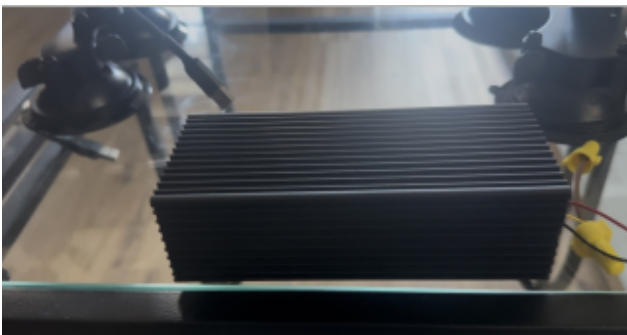


Figure 8.11: Printed Laser Housing

The laser housing is designed to isolate and secure the laser module used for road surface scanning and triangulation. Its geometry locates the laser at a controlled angle relative to the camera system, allowing consistent projection of the laser line onto the pavement. Internal mounting standoffs keep the laser fixed in orientation to maintain calibration accuracy during vehicle motion. The enclosure shields the optical components from dust, wind, and vibration, ensuring the emitted beam remains stable.. Protective clear material will cover the output aperture to prevent contaminants from affecting beam quality. The laser housing's structural rigidity and controlled optical alignment are vital for producing reliable depth measurements during system operation.

8.3.4 Main PCB Housing



Figure 8.12: Printed Main PCB Housing

The main PCB housing protects the central electronics, which are responsible for powering and coordinating all sensing components. The enclosure isolates the circuit board from weather exposure, vibration, and accidental contact through the utilization of rigid 3D-printed walls with internal mounting posts that elevate the PCB off the enclosure floor. This prevents electrical shorts, shields against debris, and allows airflow paths for heat dissipation.

9. System Testing and Evaluation

9.1 Computer Vision Training and Testing

The object detection model was trained using a custom YOLOv5n architecture optimized for edge deployment. A total of 1,660 labeled images were collected and labeled for pothole detection. 1,325 images were used for training and 335 images were used for validation to track performance during training.

This training process took a total of 7 hours, with 50 epochs being performed and without the use of a GPU. After training, the resulting PyTorch model was converted to ONNX format to support broader hardware compatibility and deployment. To reduce memory usage and improve inference speed on the Raspberry Pi 5, the ONNX model was optimized using dynamic INT8 quantization. This process compresses model weights into 8-bit integers while maintaining high accuracy, allowing for faster inference and reduced RAM consumption at runtime.

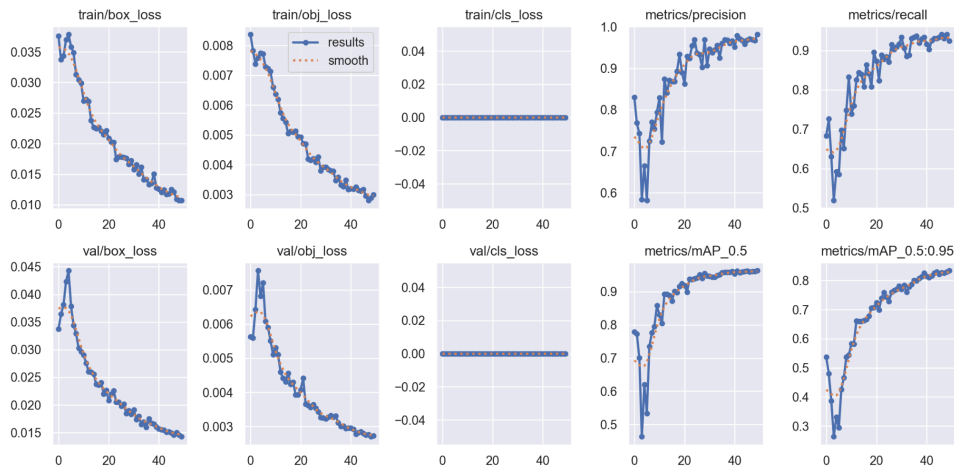


Figure 9.1: Overall Model Training Results

The figure above shows the performance metrics over 50 epochs for the YOLOv5n model trained on a binary pothole detection dataset. Training box loss, training object loss, validation box loss, and validation object loss all decrease steadily over time, which indicates effective learning. The classification loss remains zero, which is expected, since there is only one class for this model. The precision, recall, and mAP (mean average precision) metrics all show significant improvement, with mAP@0.5 reaching around 0.95 and mAP@0.5:0.95 climbing just over 0.80, suggesting strong localization and detection performance. A strength of this model is the high accuracy and stable validation curve, meaning it generalizes well without overfitting. One minor downside is a slight change in early epochs, but it quickly rebounds overall. The current mAP@0.5:0.95 of about 0.8 is very good; however, adding more data could improve this metric into the 0.9 range, resulting in even tighter and more accurate bounding boxes. Overall, this is a very effective training run for a binary detection task.

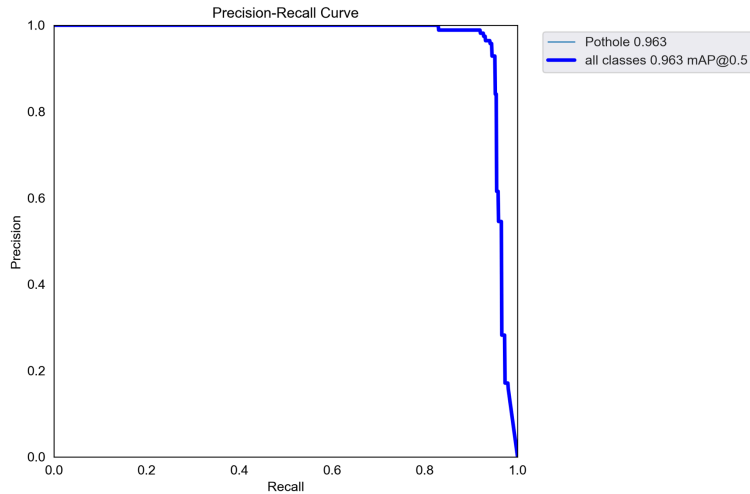


Figure 9.2: Precision-Recall Curve

The figure above shows the precision-recall curve, which shows the model's ability to balance precision and recall for different confidence thresholds. Precision measures how many of the predicted potholes are actually correct, while recall measures how many of the actual potholes the model was able to detect. The difference between these two metrics is much more noticeable in multiclass models where it may correctly guess an object exists but incorrectly classify it. In this graph, the sharp, elbow shape and the curve's tight proximity to the top-right corner shows strong performance in both areas. The mean average percentage of 0.963 shows that the model keeps excellent precision even as recall increases. This means it detects most potholes while keeping false positives low. The dip at the end is normal at low confidence thresholds. Overall, this precision-recall curve confirms the model is both accurate and reliable in detecting potholes.

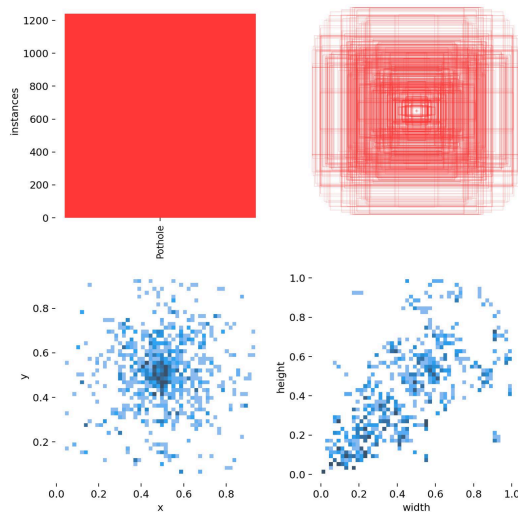


Figure 9.3: Bounding Box Visualizations

The figure above shows bounding box visualizations, which provides insight into the training dataset's distribution. The top-left chart shows that all 1,200+ labeled instances belong to the Pothole class with the entire graph being one color, since this is a binary model. The top-right chart shows the differences of bounding box sizes and locations, showing a dense concentration of annotations around the image center. The bottom-left graph reveals the spatial distribution of potholes, with a strong central bias. The bottom-right scatter plot, width vs. height, shows a clear linear relationship, suggesting that larger potholes are consistently annotated with proportionally larger boxes. Overall, these graphs indicate there is a reasonable spread of bounding box sizes and locations; however, further training should involve smaller potholes located near the edges of the images, since real world detections will vary.

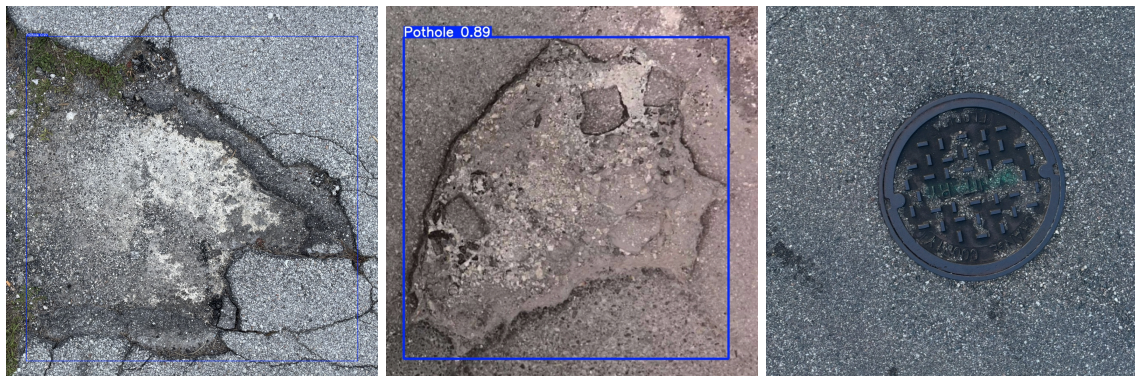


Figure 9.4: Example Detection Results

The figure above shows three example images that were run through the object detection model after optimization was applied. The image on the left,

although it's hard to read, shows a confidence score of 0.97. This means that the model was 97% confident that the object inside the bound box belongs to the pothole class. The middle image shows a confidence score of 0.89. The lower confidence score is likely due to the lighting conditions and the lack of contrast in the image. This photo was taken at night using the phone's flash compared to the natural light of the first image. The image on the right is a manhole, which was used to test the model on whether it will cause a false positive. In this case, it properly identified this as background and didn't detect a pothole. This model was trained using manholes, puddles, oil stains, and normal roadway conditions to reduce the risk of false positives.

9.2 Optical Testing

To confirm the functionality of the optical design, we conducted an experiment in the CREOL undergraduate lab that simulated the environment in which our device would be operating. We did this using foam pads designed to be inverted potholes that would show up like a pothole when the image was flipped, most of these were made to our smallest specifications while some were much larger to simulate a variety of sizes. A wall was used as our target area with our laser flipped so that the line is projected vertically rather than horizontally. The IR camera was flipped 270° to capture the target area vertically and backwards to account for the circumstances of the testing. For this example our test object had a width of 7.8 in and a depth of 1.4 in. Figure 9.5 shows the object as viewed by the IR camera while the laser line was being projected over the object at a 30° angle.



Figure 9.5: Test surface illuminated by IR laser line at an angle.

We then used the machine vision software to calculate the depth and width of the pothole. This was accomplished using equation 9.1 and the known incidence angle of the laser line. The output from the machine vision algorithm is shown in figure 9.6 below.

$$\tan(\theta) = \frac{o}{a}$$

Eq. 9.1: Tangent of an angle on a right triangle

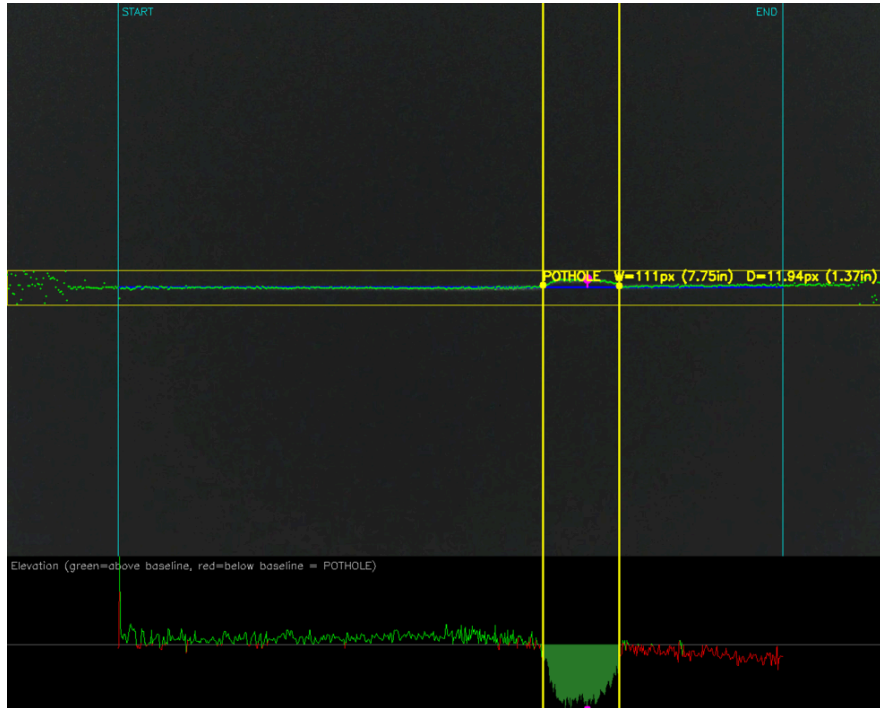


Figure 9.6: Machine vision algorithm output and measurements.

The machine vision algorithm estimated the target pothole to be 7.75 in wide and 1.37 in deep. While these are not the actual measurements of the pothole they are extremely close especially the depth calculation which is more important as the computer vision also makes an estimate for the width of the pothole. Due to the nature of the testing setup there is a possibility that the angle is not exactly 30° and thus the measurements are a little off. However, this does not concern us as the intended purpose for our design is for governments to use it to aid in the tracking and filling of potholes and it is extremely unlikely that they would only bring the exact amount of asphalt needed to fill a pothole.

The other half of our optical design was to modify a visible light camera to capture NIR light in daylight while achieving a high SNR between the laser line and sunlight in the target area. Figure 9.7 shows the laser operating in daylight on the road and the camera receiving a clear image of the line.



Figure 9.7: Laser operation on fresh asphalt in daytime, Camera output in top left.

While there is some glare on the image due to the sunlight it is clear that the laser line shows up excellently on even freshly paved asphalt. It is also apparent that the filters successfully block out enough of the noise from the sun to get a clear image of the laser line. The results of our testing suggest that the optical design is working as intended.

9.3 SPI Interface Validation

The development of SPI communication between the Raspberry Pi 5 and MSP430FR6989 microcontroller encountered significant hardware-related challenges that initially appeared to be software issues, requiring extensive systematic testing to identify and resolve pin-specific functionality problems. The initial implementation followed standard datasheet specifications for UCA0 SPI configuration, connecting Pi Pin 23 (SCLK) to MSP430 P1.5 (UCA0CLK), Pi Pin 19 (MOSI) to MSP430 P2.0 (UCA0SIMO), Pi Pin 21 (MISO) to MSP430 P2.1 (UCA0SOMI), and establishing a common ground reference between the devices. Despite implementing what appeared to be correct software configuration on both devices, with the MSP430 programmed as an SPI slave featuring interrupt-driven data reception and visual LED feedback, no communication was established between the systems, leading to initial suspicions of software-related configuration errors.

The debugging process began with extensive software troubleshooting under the assumption that the issue was due to incorrect protocol implementation or timing problems. Various SPI modes and clock frequencies were tested while implementing diagnostic features in the MSP430 code, including startup LED sequences to confirm code execution, continuous status indicators to verify main loop operation, and interrupt counters to detect SPI activity. Additionally, potential LaunchPad configuration issues were investigated by verifying all jumper settings, including the J101 isolation jumpers, J7 and J8 power jumpers, and voltage selection jumpers, all of which were confirmed to be in their correct

default positions. The persistence of communication failure despite these software modifications indicated that the root cause was likely hardware-related rather than stemming from code implementation errors.

Recognizing that conventional software debugging approaches were insufficient, a systematic hardware testing methodology was developed that bypassed the SPI protocol entirely to isolate potential pin functionality issues. Due to the absence of multimeter access for traditional continuity and voltage testing, alternative verification methods were necessary to diagnose the hardware interface problems. This approach involved creating a GPIO-based testing framework where MSP430 pins were configured as simple digital inputs with pull-up resistors, monitored through polling rather than interrupt-driven SPI communication. The test methodology began with manual verification, where each MSP430 pin was physically connected to ground using jumper wires, confirming that all pins (P2.0, P2.1, and P1.5) appeared to function correctly and triggered the expected LED responses. Building on this foundation, Python scripts were developed to control individual Raspberry Pi GPIO pins as digital outputs, systematically testing each connection by toggling pins between HIGH and LOW states while monitoring the MSP430 response.

The systematic GPIO testing revealed the critical hardware issue that had been masquerading as a software problem: while Pi Pin 19 successfully controlled MSP430 P2.0, and Pi Pin 21 effectively controlled MSP430 P2.1, Pi Pin 23 failed to produce any detectable response from MSP430 P1.5, despite P1.5 responding correctly to manual ground connections. This discovery was particularly significant because P1.5 served as the SPI clock input (UCA0CLK), meaning that, while data transmission paths were functional, the synchronization signal essential for SPI protocol operation was not being properly received by the microcontroller. The pin-specific nature of this failure explained why the software appeared correct but communication remained impossible, since the hardware foundation for the protocol was compromised at the most critical connection point.

Further investigation into alternative pin configurations led to examination of the MSP430FR6989 datasheet for additional UCA0CLK options, identifying P2.2 as an alternative clock input pin. However, initial testing of P2.2 also appeared to fail, which was later discovered to be due to a coding oversight where the MSP430 monitoring code had not been updated to include P2.2 in the pin state checking routine. After correcting this configuration error and properly including P2.2 in the test code, the connection between Pi Pin 23 and MSP430 P2.2 functioned as expected, establishing a complete working pin configuration that utilized P2.0, P2.1, and P2.2 all on Port 2 of the microcontroller. This hardware solution allowed for the use of the original SPI software implementation

without any protocol modifications, confirming that the initial code was correct and that the communication failure had been entirely due to the non-functional P1.5 pin connection.

With the working pin configuration established through systematic individual testing, the complete SPI communication system was implemented using the validated hardware connections. The final configuration connected Pi Pin 23 (SCLK) to MSP430 P2.2 (UCA0CLK), Pi Pin 19 (MOSI) to MSP430 P2.0 (UCA0SIMO), Pi Pin 21 (MISO) to MSP430 P2.1 (UCA0SOMI), and maintained the ground reference connection, as shown in the figure below, which displays the finalized physical setup between the Raspberry Pi and MSP430 development boards. This arrangement successfully established bidirectional SPI communication at 50kHz, with the MSP430 reliably receiving data from the Raspberry Pi and providing visual confirmation through LED responses for each transmitted byte, validating that the original software implementation had been correct throughout the debugging process.

The debugging experience demonstrated that hardware-related issues can often present symptoms identical to software configuration problems, particularly in embedded systems where physical pin functionality directly impacts protocol operation. The systematic GPIO testing methodology, necessitated by the lack of traditional test equipment such as multimeters, proved essential for distinguishing between software and hardware failures, providing a reliable foundation for identifying the specific, non-functional pin that had prevented communication. This experience emphasized the importance of comprehensive hardware validation before assuming software-related issues, as the resolution ultimately required no changes to the original SPI protocol implementation once the correct pin configuration was identified. The successful establishment of this communication interface created a robust foundation for the pothole detection system's real-time data transmission requirements between the AI processing unit and the wireless communication controller.

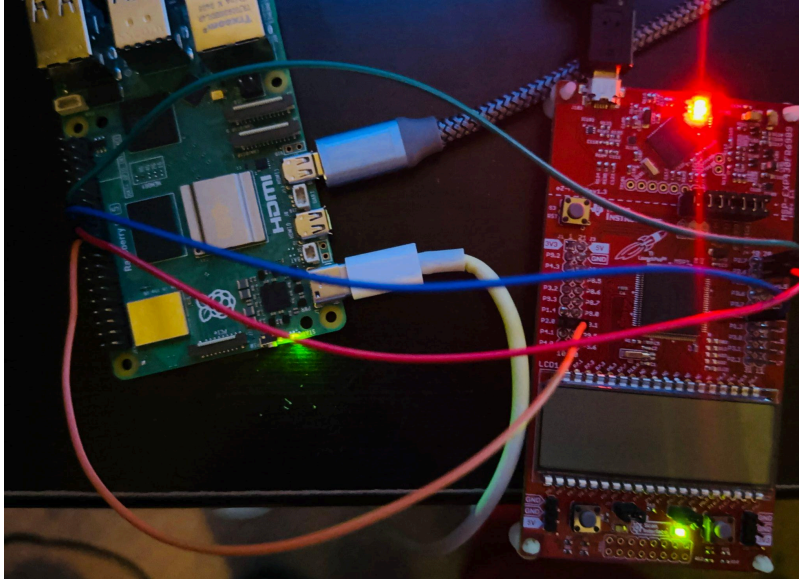


Figure 9.8: Working SPI connection between Raspberry PI 5 and MSP430FR6989

9.4 Full System Testing

To test our system we integrated the device into one of our own vehicles. This was done by attaching the device to the rear of the vehicle, using the mounting arm to adjust the height, angle and alignment of the laser and cameras.



Figure 9.9: Device mounted to a vehicle.

We then fed the wires powering and connecting the laser and cameras through the car to the area below the driver's seat where the PCB is attached the vehicle. Using fuse taps we supplied power to the system from the car's own

battery. After the system was completely implemented we drove the car over a pothole and recorded the data captured by our app as seen in figure 9.10.

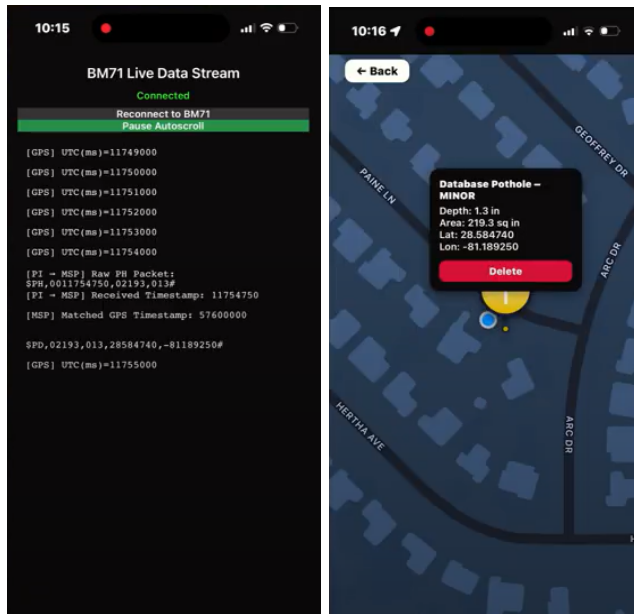


Figure 9.10: Data collection in app (Right) Pothole location and size data on map (Left)

After measuring the pothole we found it to have an actual area of 223 sq. in and a depth of 1.5 in. These results show that our fully integrated system does indeed work with the only issue being that our calculations are a bit off from the real size of the pothole. As mentioned earlier this is not a big concern due to the nature of how governments approach the filling of potholes.

10. Administrative Content

10.1 Budget

The budget for this project is limited to \$1600. We set this limit as we will be paying for this project out of pocket and would like to keep costs low. Additionally, a large part of our design is to make the device easy to use and accessible; if the device were too expensive, it would limit its accessibility. Even with room for additional costs we ended up hitting our budget. This was due to some things needing to be reiterated and additional PCBs being bought to ensure we had a working system come demo day.

10.2 Bill of Materials

Table 10.2.1: Bill of Materials

Component	Name	Quantity	Price
IR Sensitive Camera	Arducam B0332	x1	\$49.99
RGB Camera USB 3.0	ASIN:B0DBV8VFJZ	x1	\$152.28
IR Camera Lens	Arducam 1/4" M12 Mount 2.1mm Focal Length Low Distortion Camera Lens	x1	\$14.51
Housing Unit Materials	TBD	TBD	~\$50.00
12V Car Fuse Taps	Nilight - 50038R	x1	\$8.09
PCB	JLCPCB	x1	~\$371.00
NIR Laser Diode	L820P200 820nm Laser Diode	x1	\$106.47
Bandpass Filter	Filter BP Col Hoya RT-830 25.4x2.5mm	x1	\$69.96
Longpass Filter	Filter LP Col Hoya IR-80 50.8sqx2.5mm	x1	\$100.28
Aspherical Collimating Lens	Thorlabs C330TMD Molded Aspheric Lens	x1	\$86.65
90° Fan Powell Lens	B0C68P8M56	x1	\$42.57
Laser Collimation Package	LDH56-P2	x1	\$141.10

Raspberry Pi 5	B0CK2FCG1K	x1	\$95.99
16-bit MCU	MSP430fr2355TDBTR	x1	\$2.69
15 foot USB 2.0 extender and repeater Cable	ASIN: B0B24CPX5C	x2	\$37.00
6 foot USB 3.0 cable	ASIN:B09YYV8M83	x2	\$37.00
Bluetooth v5.0 Transceiver Module 2.402 ~ 2.48GHz	BM71BLES1FC2-0B04 AA	x1	\$9.24
DC Buzzer	458-ASI09N27MFD-03T RQCT-ND	x1	\$3.67
Laser Driver	ATLS500MA104D	x1	\$79.00
GPS Module	1528-1153-ND	x1	\$29.95
GPS Antenna	ASIN: B0769FRT6X	x1	\$15.45
360 camera mount	PellKing Suction Car Mount for Insta360	x1	109.99
ABS Filament	PE01001	x1	22.99
Total			\$1597.81

10.3 Milestones and Schedule

10.3.1 Senior Design 1

Table 10.3.1: Senior Design Schedule

Week	Description and Dates
1-3	Group formation, Research designs, Write D&C due 5/30
4-5	D&C meeting (6/3), Upload revision to website (6/6), More Research, Start ordering parts, Begin midterm report, Start building optics
6-8	Continue work on midterm report, Start developing code and building CAD designs, Optics Demo(6/26), Test camera framerate, Begin designing PCB
9-11	Finish Midterm (7/7), Test Code with Dev Board + Optics, Midterm meeting (7/9-7/11), Upload midterm revision (7/18), Work on final report
12-13	Work on the final report, mini demo video, and building the optical design. Finish up PCB design and order
13	Finish Final Report, Mini Demo Video, and Optical Design (7/29), Begin prototyping

10.3.2 Senior Design 2

Table 10.3.2: Senior Design 2 Schedule

Week	Description and Dates
14-15	School Break
16	Edit a 150-page paper as needed
17-19	Begin building device housing and mounting optics, implement laser diode with TEC and Laser Driver for on-board testing, and begin implementing baseline code
20-22	Connect everything to the PCB and finish building the device housing. Assemble the device
23-24	Begin full system testing, identify problems, and begin solving them

25-29	Lots of testing, make a final demo video, make final changes to the design, and hope for the best
30	Final Committee Presentation

10.4 Table of Work

Table 10.4.1: *Table of Work*

Role	Primary	Secondary	Tertiary
PCB Design	Samuel Welch	Travis Grant	Jose Kostyun
Optics	John Billeci	Travis Grant	Brandon Skervin
Computer Vision	Travis Grant	Jose Kostyun	John Billeci
MCU Programming	Jose Kostyun	Travis Grant	Samuel Welch
Application	Jose Kostyun	Travis Grant	Samuel Welch
Housing Design	Brandon Skervin	Travis Grant	John Billeci
Mounting Design	Brandon Skervin	Travis Grant	Samuel Welch

11. Conclusion

In conclusion, the Pothole Detection and Tracking System represents an important step toward modernizing how cities detect and respond to road surface damage. As described in the executive summary, the goal of this project is to build a cost-effective and automated solution that addresses the shortcomings of traditional road inspection methods, which often depend on slow manual surveys, inconsistent user reporting, or expensive specialized vehicles.

This project focuses on creating a system that is more efficient and significantly less expensive than existing commercial options. Current systems, such as LiDAR-equipped inspection trucks, high-end integrated vehicle scanners, and post-processed video services, can be accurate, but remain impractical for many municipalities due to cost and complexity. In contrast, the Pothole Detection and Tracking System uses readily available components, lightweight embedded processing, and machine learning models to provide cost-efficient, real-time results.

The system is organized into three major components that work together as an integrated platform. The first is the camera and infrared scanning system. An infrared laser projects a thin sheet of light across the road behind a vehicle. An infrared camera records this projection and continuously monitors distortions caused by dips or holes in the surface. When a distortion is found, the Raspberry Pi triggers the RGB camera to capture a high-resolution image of that distortion area. This method ensures that only areas with likely hazards are analyzed, which makes the system more efficient and accurate.

The second component is the processing hardware, which consists of a Raspberry Pi and a custom MSP430-based printed circuit board. The Raspberry Pi runs real-time image analysis software that identifies potential potholes and passes those images through a lightweight YOLOv5n machine learning model for confirmation. The MSP430 handles essential low-level control tasks such as laser driver management, triggering the cameras, managing power, and communicating with the Pi over SPI. To ensure safe and reliable operation, the device draws power through the vehicle's fuse box.

The third component is the mobile application and cloud system. When a pothole is confirmed by the onboard AI, the data, including the image, time of detection, and system measurements, is sent via Bluetooth to a smartphone. The app records GPS coordinates using the phone's location services, stores the event locally, and uploads it to a secure cloud database if a network is available. Maintenance teams, administrators, and users can view detections through an interactive map, confirm or reject entries, and keep track of repair progress.

The design incorporates a custom, weather-resistant, rear-mounted bracket so the system can be attached to a wide variety of vehicles. By mounting on ordinary cars, fleets, or even contractor vehicles, the system can collect a constant stream of road condition data with little added cost. This portable approach differs in comparison with specialized scanning trucks, which are expensive and limited in deployment.

With this integration of hardware, software, and cloud services, the project has demonstrated that low-cost sensors combined with embedded processing and AI can be used to build a capable and reliable pothole detection tool. It not only improves the accuracy and speed of pothole reporting but also makes these technologies accessible to cities and contractors that do not have the resources to invest in traditional, high-end systems.

Future work for the Pothole Detection and Tracking System will focus on improving detection under poor weather and low-light conditions, refining the mobile application to support multiple mobile OS's and additional features, and

expanding the system's ability to estimate the size and depth of potholes. Longer-term goals include deploying these units across multiple vehicles to provide city-wide coverage and analytics.

This document serves as a comprehensive record of the project, including the motivation, background, technology selection, design, testing, and lessons learned. The result of this work is a low-cost, automated pothole detection system that addresses a common infrastructure problem using accessible, scalable technology. The planning and teamwork of the University of Central Florida engineering students demonstrate how practical innovation can lead to smarter, safer, and more efficient road maintenance systems.

Appendix A | References

- [1] A. Anaissi, K. Nguyen, T. Rakotoarivelo, and M. Alamdari, "Smart pothole detection system using vehicle-mounted sensors and machine learning," *ResearchGate*, Jan. 2019. [Online]. Available: https://www.researchgate.net/publication/330301489_Smart_pothole_detection_system_using_vehicle-mounted_sensors_and_machine_learning
- [2] C. Ruseruka, J. Mwakalange, G. Comert, S. Siuhi, F. Ngeni, and Quincy Anderson, "Pothole detection and dimension estimation using in-vehicle sensors," *Transportation Engineering*, vol. 10, Mar. 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2666827024000239>
- [3] R. Reidy, "Michelin acquires computer vision start-up RoadBotics," *Traffic Technology Today*, Jul. 12, 2022. [Online]. Available: <https://www.traffictotechnologytoday.com/news/acquisitions-mergers/michelin-acquires-computer-vision-start-up-roadbotics.html>
- [4] F. Lambert, "Tesla vehicles are now scanning for potholes and rough roads to help avoid them," *Electrek*, Jul. 4, 2022. [Online]. Available: <https://electrek.co/2022/07/04/tesla-vehicles-scanning-for-potholes-and-rough-roads-help-avoid-them/>
- [5] Federal Highway Administration, "Road/Intelligent Transportation System (ITS) Maintenance," *U.S. Department of Transportation*, [Online]. Available: <https://ops.fhwa.dot.gov/crowdsourcing/examples/maintenance.cfm>
- [6] ImageVision, "Pothole Detection Using Computer Vision for Road Quality Management," *ImageVision*, Feb. 2025. [Online]. Available: <https://imagevision.ai/blog/pothole-detection-using-computer-vision-for-road-quality-management/>
- [7] K. Quinlan, "Memphis piloting AI video tech equipped on city trucks to find, fix infrastructure issues," *StateScoop*, Dec. 24, 2024. [Online]. Available: <https://statescoop.com/memphis-ai-video-tech-equipped-city-trucks-fix-infrastructure-issues/>
- [8] A. Rubin, "City of San Jose testing use of AI to identify potholes, graffiti and homeless encampments," *KTVU*, Mar. 26, 2024. [Online]. Available: <https://www.ktvu.com/news/city-of-san-jose-testing-use-of-ai-to-identify-potholes-graffiti-and-homeless-encampments>
- [9] XenomatiX, "XenoTrack | Road Profile LiDAR," *XenomatiX*, [Online]. Available: <https://xenomatix.com/lidar/xenotrack/>

- [10] G. Gavigan, "LiDAR and AI Used to Detect Potholes," *LiDAR News*, Mar. 28, 2023. [Online]. Available: <https://blog.lidarnews.com/lidar-and-ai-used-to-detect-potholes/>
- [11] A. Jain, "Single-stage detector vs 2-stage detector," Medium, Feb. 10, 2025. [Online]. Available: <https://medium.com/@abhishekjainindore24/single-stage-detector-vs-2-stage-detector-3e540ea81213>
- [12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2015, pp. 91–99.
- [13] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201
- [14] U. Nepal and H. Eslamiat, "Comparing YOLOv3, YOLOv4 and YOLOv5 for autonomous landing spot detection in faulty UAVs," *Sensors*, vol. 22, no. 2, p. 464, 2022, doi: 10.3390/s22020464.
- [15] O. E. Olorunshola, M. E. Irhebhide, and A. E. Ewiekpaefe, "A comparative study of YOLOv5 and YOLOv7 object detection algorithms," *Journal of Computing and Social Informatics*, vol. 2, no. 1, pp. 1–9, 2023.
- [16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Amsterdam, The Netherlands, 2016, pp. 21–37.
- [17] M. R. Islam, M. A. Rahman, and Y. M. Jang, "Lightweight deep learning models for real-time object detection on edge devices," *J. Real-Time Image Process.*, vol. 21, pp. 1–12, 2024.
- [18] Ultralytics, "YOLOv5 performance benchmarks," *Ultralytics Docs*, 2023. [Online]. Available: <https://docs.ultralytics.com>
- [19] Cytron Technologies, "YOLO vs SSD: A detailed comparison for object detection," *Cytron.io*, Apr. 19, 2022. [Online]. Available: <https://www.cytron.io/tutorial/yolo-vs-ssd-a-detailed-comparison-for-object-detection>

- [20] Dong, Pinliang, and Qi Chen. *Lidar Remote Sensing and applications*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2018.
- [21] Raj, Thinal, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. "A survey on LiDAR scanning mechanisms." *Electronics* 9, no. 5 (2020): 741.
- [22] OpenAI, "ChatGPT", OpenAI, [Online]. Available: <https://www.openai.com>. [Accessed 15 June 2025]
- [23] S. Ashfaq, M. H. Askari Hemmat, S. Sah, E. Saboori, O. Mastropietro, and A. Hoffman, "Accelerating deep learning model inference on Arm CPUs with ultra-low bit quantization and runtime," in *Proc. Embedded World Conf.*, Nuremberg, Germany, 2022.
- [24] X. Shen, P. Dong, L. Lu, Z. Kong, Z. Li, M. Lin, C. Wu, and Y. Wang, "Agile-Quant: Activation-Guided Quantization for Faster Inference of LLMs on the Edge," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, 2024.
- [25] N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications," *Micromachines*, vol. 13, no. 6, p. 851, May 2022.
- [26] R. Lopez Mendez, "Neural network model quantization on mobile," *Arm Community AI Blog*, Oct. 17, 2023
- [27] Szeliski, Richard. *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [28] Texas Instruments, *MSP430FR2355 Mixed-Signal Microcontroller Datasheet*, Rev. E, Apr. 2024. [Online]. Available: <https://www.ti.com/lit/ds/symlink/msp430fr2355.pdf>
- [29] Espressif Systems, *ESP32-WROOM-32E & ESP32-WROOM-32UE Datasheet*, v1.6, Apr. 2024. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf
- [30] Abdullwahed, S., & Tran, X. D. (2023). Evaluating the Performance of React Native and .NET MAUI for Cross-Platform Mobile Development. Kristianstad University Thesis Repository.
- [31] Abu Zahra, H., & Zein, S. (2022). A Systematic Comparison Between Flutter and React Native from an Automation Testing Perspective. Birzeit University.

- [32] Biørn-Hansen, A., Grønli, T. M., Ghinea, G., & Yigitbas, E. (2019). An Empirical Study of Cross-Platform Mobile Development in Industry. *Advances in Human-Computer Interaction*. <https://doi.org/10.1155/2019/5743892>
- [33] Ivanović, Z., Kravari, K., & Korošec, P. (2023). A large-scale empirical study on mobile performance: energy, runtime, and code quality in Android apps. *Empirical Software Engineering*. <https://doi.org/10.1007/s10664-023-10391-y>
- [34] Lee, M.-S., Kim, S., & Park, J. (2022). Kotlin, React Native and Flutter: an empirical comparison. In *Proceedings of the 2022 ACM Symposium on Applied Computing*. <https://doi.org/10.1145/3590837.3590897>
- [35] Müller, L.-M., et al. (2022). Understanding the quality and evolution of Android app build systems. *Journal of Software: Evolution and Process*, 27(1), 1–35. <https://doi.org/10.1002/smr.2602>
- [36] Lovrić, L., Fischer, M., Röderer, N., & Wunsch, A. (2023). Evaluation of the Cross-Platform Framework Flutter Using the Example of a Cancer Counselling App. In *Proceedings of the 9th International Conference on Information and Communication Technologies for Ageing Well and e-Health (ICT4AWE)* (pp. 135–142). SCITEPRESS. <https://doi.org/10.5220/0011824500003476>
- [37] Tan, C. W., Lin, C. Y., & Choo, Y. H. (2021). Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-Based LLMs. Presented at Nanyang Technological University.
- [38] Uddin, M., Chen, M., & Lee, S. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering*. <https://doi.org/10.1007/s10664-020-09827-6>
- [39] Texas Instruments, *MSP430FR5xx and MSP430FR6xx Family User's Guide*, SLAU367P, May 2013 [Revised June 2024]. [Online]. Available: <https://www.ti.com/lit/ug/slau367p/slau367p.pdf>
- [40] Raytac Corporation, “Downloads – MDBT50Q Series,” [Online]. Available: https://www.raytac.com/download/index.php?index_id=43. [Accessed: June 22, 2025].
- [41] Microchip Technology Inc., *BM70/71 Bluetooth® Low Energy (BLE) Module Data Sheet* (DS60001372H), July 2018. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/BM70-71-Bluetooth-Low-Energy-BLE-Module-Data-Sheet-DS60001372H.pdf>
- [42] H. Ahn, T. Chen, N. Alnaasan, A. Shafi, M. Abduljabbar, H. Subramoni, and D. K. Panda, “Performance characterization of using quantization for DNN inference on edge devices,” in *Proc. 7th IEEE Int. Conf. Fog Edge Computing (ICFEC)*, May 2023.

- [43] P. Bjørn-Hansen, F. G. Ghinea, and G. A. M. Jørgensen, "Progressive Web Apps: The Possible Web-native Unifier for Mobile Development," *Proc. 13th Int. Conf. Web Inf. Syst. Technol. (WEBIST)*, pp. 344–351, 2017. Available: <https://www.scitepress.org/Papers/2017/63537/63537.pdf>
- [44] M. S. S. Lingolu and M. K. Dobbala, "A Comprehensive Review of Progressive Web Apps: Bridging the Gap Between Web and Native Experiences," *Int. J. Sci. Res.*, vol. 11, no. 2, pp. 1326–1334, Feb. 2022. Available: https://www.researchgate.net/publication/380908572_A_Comprehensive_Review_of_Progressive_Web_Apps_Bridging_the_Gap_Between_Web_and_Native_Experiences
- [45] R. Horn, A. Lahnaoui, E. Reinoso, S. Peng, V. Isakov, T. Islam, and I. Malavolta, "Native vs Web Apps: Comparing the Energy Consumption and Performance of Android Apps and their Web Counterparts," *Proc. 10th Int. Conf. Mobile Software Eng. Syst. (MOBILESoft)*, pp. 44–54, May 2023. Available: DOI 10.1109/MOBILSoft59058.2023.00013
- [46] A. Banker, "MongoDB and the SSPL: What You Need to Know," InfoWorld, Nov. 2020. [Online]. Available: <https://www.infoworld.com/article/3630187/mongodb-and-the-sspl-what-you-need-to-know.html>
- [47] R. Obe and L. Hsu, *PostGIS in Action*, 3rd ed., Manning Publications, 2021.
- [48] Oracle Corporation, "Oracle Spatial and Graph Overview," Oracle Technology Network, 2020. [Online]. Available: <https://www.oracle.com/database/technologies/spatialandgraph.html>
- [49] D. Richard Hipp, "SQLite Technical Documentation," SQLite.org, 2023. [Online]. Available: <https://www.sqlite.org/docs.html>
- [50] Microsoft Corporation, "SQL Server Spatial Features Overview," Microsoft Learn, 2024. [Online]. Available: <https://learn.microsoft.com/en-us/sql/relational-databases/spatial/spatial-data-sql-server>
- [51] S. Shankar, A. El-Maleh, and M. N. Qureshi, "Comparative Study of Native and Third-Party Location APIs for Mobile Sensing," *International Journal of Mobile Computing and Multimedia Communications*, vol. 13, no. 2, pp. 22-35, 2023.
- [52] R. Ali and Z. Huang, "Usability and Performance Metrics in Mobile Navigation Using Google Maps SDK," *Journal of Mobile Human-Computer Interaction*, vol. 10, no. 1, pp. 17-28, 2022.
- [53] A. Meixner and A. Papakostas, "Evaluating Mapbox for Mobile GIS Applications with Offline Constraints," *IEEE Transactions on Mobile Computing*, vol. 21, no. 4, pp. 980-990, 2022.

- [54] A. Abd-Elrahman, M. Ghoneim, and R. H. Salem, "OpenStreetMap in Resource-Constrained Mobile GIS: Performance and Trade-offs," *Computers, Environment and Urban Systems*, vol. 88, 2021.
- [55] T. Das, L. B. Morrison, and K. L. Chen, "Integrating Third-Party Navigation Intents into Location-Based Apps: An Empirical Analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, 2023.
- [56] Y. Huang, D. Sun, and M. K. Ray, "Time Synchronization Techniques in Bluetooth-Enabled IoT Devices," *IEEE Internet of Things Journal*, vol. 9, no. 6, pp. 4302-4312, 2022.
- [57] Leaflet, "Leaflet Mobile Example," LeafletJS. [Online]. Available: <https://leafletjs.com/examples/mobile/>. [Accessed: Jun. 29, 2025.]
- [58] Google Maps Platform, "Add a marker to a map," Google Developers. [Online]. Available: <https://developers.google.com/maps/documentation/android-sdk/marker>. [Accessed: Jun. 29, 2025.]
- [59] Google Maps Platform, "Store and retrieve data," Google Developers. [Online]. Available: <https://developers.google.com/maps/documentation/javascript/examples/marker-simple>. [Accessed: Jun. 29, 2025.]
- [60] Mapbox, "Add custom markers in Mapbox GL JS," Mapbox Docs. [Online]. Available: <https://docs.mapbox.com/mapbox-gl-js/example/custom-marker-icons/>. [Accessed: Jun. 29, 2025.]
- [61] MapLibre, "Add a marker to a map," MapLibre GL JS Docs. [Online]. Available: <https://maplibre.org/maplibre-gl-js-docs/example/add-a-marker/>. [Accessed: Jun. 29, 2025.]
- [62] Leaflet, "Markers," LeafletJS. [Online]. Available: <https://leafletjs.com/reference.html#marker>. [Accessed: Jun. 29, 2025.]
- [63] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive parallel programming in Python," *arXiv preprint arXiv:1905.02158*, May 2019.
- [64] Z. A. Aziz, D. N. Abdulqader, A. B. Sallow, and H. K. Omer, "Python parallel processing and multiprocessing: A review," *Academic Journal of Nawroz University*, vol. 10, no. 3, pp. 345–354, 2021.
- [65] Z. Aslan and A. Aituov, "Evaluation of concurrency methods in Python for processor-intensive tasks on multicore systems," *Research Reviews*, vol. 9, no. 2, pp. 34-42, 2025.

- [66] A. Jones and B. Lee, "Parallel Processing and Multiprocessing in Python: A Review," *Journal of High-Performance Computing Systems*, vol. 15, no. 4, pp. 211–224, 2020.
- [67] Python Software Foundation, "multiprocessing - Process-based parallelism," *Python 3.13.5 Documentation*. [Online]. Available: <https://docs.python.org/3/library/multiprocessing.html>
- [68] Numba Developers, "Compiling Python code with @jit," *Numba User Guide*, 2025. [Online]. Available: <https://numba.readthedocs.io/en/stable/user/jit.html>
- [69] A. Milla and E. Rucci, "Performance Comparison of Python Translators for a Multi-threaded CPU-bound Application," *arXiv preprint arXiv:2203.08263*, Mar. 2022.
- [70] S. Kailasa, T. Wang, L. A. Barba, and T. Betcke, "PyExaFMM: An Exercise in Designing High-Performance Software with Python and Numba," *arXiv preprint arXiv:2303.08394*, Mar. 2023.
- [71] Le, Thithanhnhhan, Nam-Tuan Le, and Yeong Min Jang. "Performance of rolling shutter and global shutter camera in optical camera communications." In *2015 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 124-128. IEEE, 2015.
- [72] H. Angus Macleod. *Thin-film optical filters*. CRC press, 2010.
- [73] [BeagleBoard.org](https://beagleboard.org) Foundation, "BeagleBone AI-64: Design and Specifications", BeagleBoard Documentation, [Online]. Available: <https://docs.beagleboard.org/boards/beaglebone/ai-64/03-design-and-specifications.html#bbai64-design>. [Accessed: June 28, 2025].
- [74] Raspberry Pi Foundation, "Raspberry Pi 5," *Raspberry Pi Products*, [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-5/>. [Accessed: June 28, 2025].
- [75] X. Zhang, et al., "Analysis of Alignment Tolerances in Optical Systems," *Optical Engineering*, vol. 56, no. 2, 2017.
- [76] N. Hunaidi, "Measurement of Low-Frequency Groundborne Vibrations," *Journal of Vibration and Control*, vol. 6, no. 6, pp. 745–758, 2000.
- [77] U.S. Department of Defense, "MIL-STD-810H: Environmental Engineering Considerations and Laboratory Tests," 2019.
- [78] ASTM E6-15, "Standard Terminology Relating to Methods of Mechanical Testing," ASTM International, 2015.

- [79] ASM International, *ASM Handbook Volume 2: Properties and Selection: Nonferrous Alloys and Special-Purpose Materials*, ASM International, 1990.
- [80] ASTM E466-15, “Standard Practice for Conducting Force Controlled Constant Amplitude Axial Fatigue Tests of Metallic Materials,” ASTM International, 2015.
- [81] R. C. Hibbeler, *Mechanics of Materials*, 10th ed. Boston, MA: Pearson, 2016.
- [82] F. P. Beer, E. R. Johnston Jr., J. T. DeWolf, and D. F. Mazurek, *Mechanics of Materials*, 7th ed., New York, NY: McGraw-Hill, 2015.
- [83] J. M. Gere and B. J. Goodno, *Mechanics of Materials*, 8th ed. Stamford, CT: Cengage Learning, 2012.
- [84] [4] F. P. Incropera, D. P. DeWitt, T. L. Bergman, and A. S. Lavine, *Fundamentals of Heat and Mass Transfer*, 7th ed., Wiley, 2011.
- [85] IEEE Standards Association, *IEEE Recommended Practice for Powering and Grounding Electronic Equipment (Emerald Book)*, IEEE Standard 1100-2005, 2005. [Online]. Available: <https://standards.ieee.org/ieee/1100/1640/>
- [86] IPC. (1998). *IPC-2221A: Generic Standard on Printed Board Design*. Association Connecting Electronics Industries. Retrieved from [https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A\(L\).pdf](https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf)
- [87] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed., Upper Saddle River, NJ, USA: Prentice Hall, 2008.
- [88] A. K. Jain, *Fundamentals of Digital Image Processing*, Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.
- [89] J. Twicken, “Difference Imaging and Centroid Analysis,” NASA Ames Design Note, 2011.
- [90] Texas Instruments, *MSP430x2xx Family User Guide*, SLAU144J, Rev. J, Jan. 2017. [Online]. Available: <https://www.ti.com/lit/ug/slau144j/slau144j.pdf>
- [91] Z. Zhang, Y. Wang, J. Lin, and T. Li, “Real-time object detection on embedded devices: A survey,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6135–6144, Sep. 2020.
- [92] P. Jiang, W. Zhao, Y. Tang, and L. Li, “Speed-aware object tracking for real-time intelligent transportation,” *Pattern Recognition Letters*, vol. 125, pp. 595–601, Jul. 2019.

- [93] Y. Liu, J. Zhang, and Q. Chen, "Lens distortion impacts on CNN-based object detection," *Optical Engineering*, vol. 61, no. 8, pp. 1–10, Aug. 2022.
- [94] A. Singh, R. Verma, and K. Bhattacharya, "USB 3.0 throughput optimization for embedded vision," *IEEE Embedded Systems Letters*, vol. 13, no. 2, pp. 48–51, Jun. 2021.
- [95] N. Mahajan, T. Ramachandran, and H. Kumar, "Evaluating rolling vs global shutter for mobile vision tasks," *IEEE Sensors Journal*, vol. 19, no. 5, pp. 1870–1879, Mar. 2019.
- [96] H. Sun, *Laser Diode Beam Basics*. Dordrecht, Netherlands: Springer, 2012. [Online]. Available: <https://doi.org/10.1007/978-94-007-4663-3>
(Replace with exact file title if different; source appears to be from Anna's Archive)
- [97] E. Hecht, *Optics*, 5th ed. San Francisco, CA, USA: Pearson Education, 2016. [Online]. Available: <https://emineter.wordpress.com/wp-content/uploads/2020/04/hecht-optics-5ed.pdf>
- [98] National Renewable Energy Laboratory, "Reference Solar Spectral Irradiance: Air Mass 1.5," [Online]. Available: <https://www.nrel.gov/grid/solar-resource/spectra-am1.5>
- [99] Hamamatsu Photonics, "Si photodiodes," KSPDB0001E, 2014. [Online]. Available: https://www.hamamatsu.com/content/dam/hamamatsu-photonics/sites/documents/99_SALES_LIBRARY/ssd/si_pd_kspd0001e.pdf
- [100] Mouser Electronics, "OSRAM SFH 4715AS Infrared Emitter Datasheet," [Online]. Available: https://www.mouser.com/datasheet/2/588/asset_pdf_25493258-3418825.pdf
- [101] Arducam, "12MP IMX708 USB 3.0 Camera Datasheet," [Online]. Available: https://blog.arducam.com/downloads/datasheet/B0474_12MP_IMX708_USB3.0_Camera_Datasheet.pdf
- [102] UCTRONICS, "B0332 OV9281 Global Shutter UVC Camera Datasheet," [Online]. Available: https://www.uctronics.com/download/Amazon/B0332_OV9281_Global_Shutter_UVC_Camera_Datasheet.pdf
- [103] Raspberry Pi Foundation, "Camera Module 3 Product Brief," [Online]. Available: <https://datasheets.raspberrypi.com/camera/camera-module-3-product-brief.pdf>
- [104] Mallory Sonalert Products, Inc., *ASI09N27MFD-03TRQ Magnetic Buzzer — Specifications*, MSP Indy, Accessed: Feb. 2025. [Online]. Available: <https://mspindy.com/spec-sheets/ASI09N27MFD-03TRQ.pdf>
- [105] Mallory Sonalert Products, Inc., *ASI09N27MFD-03TRQ Magnetic Buzzer — Specifications*, MSP Indy, Accessed: Feb. 2025. [Online]. Available: <https://mspindy.com/spec-sheets/ASI09N27MFD-03TRQ.pdf>

[106] PUI Audio, Inc., *SMT-0827-TW-3V-R — Magnetically Driven SMD Transducer, Datasheet*, PUI Audio, Accessed: Feb. 2025. [Online]. Available: <https://api.puiaudio.com/filename/SMT-0827-TW-3V-R.pdf>

[107] CUI Devices (Same Sky), *CMI-9705-0380-SMT-TR Magnetic Buzzer — Datasheet*, Accessed: Feb. 2025. [Online]. Available: <https://www.alldatasheet.com/datasheet-pdf/pdf/1188736/CUI/CMI-9705-0380-SMT-TR.html>

[108] Adafruit Industries LLC, *Ultimate GPS Breakout v3 — MTK3339 Module, Datasheet*, Jul. 2015. Accessed: Feb. 2025. [Online]. Available: https://www.adafruit.com/datasheets/Adafruit_Ultimate_GPS_v3.pdf

[109] CDTOP Technology, *PA1616D GPS Receiver Module — Datasheet V0.5*, Accessed: Feb. 2025. [Online]. Available: https://cdn-shop.adafruit.com/product-files/5186/5186_PA1616D_Datasheet.pdf

Appendix B | Copyright Permission

TG Travis Grant
To: sales@cytron.io
Mon 6/9/2025 1:33 AM
Retention: UCF Delete after 10 Years (10 years) Expires: Thu 6/7/2035 1:33 AM

Hello,

My name is Travis Grant, I'm a Computer Engineering student at the University of Central Florida working on a group capstone project.

My team and I were hoping to obtain permission to use the following images from the article "YOLO vs SSD: A Detailed Comparison for Object Detection," written by kristisswaren. Proper credit and citation will be used.

Thanks,

Travis

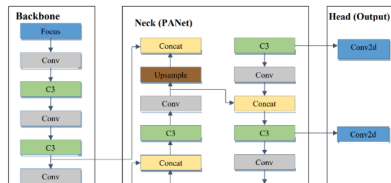


TG Travis Grant
To: seysisola25@yahoo.com; +2 others
Tue 6/10/2025 1:36 AM

Hello,

My name is Travis Grant, I'm a Computer Engineering student at the University of Central Florida who is currently working on a group capstone project. My team and I was hopping to obtain permission to use the attached photo from "A Comparative Study of YOLOv5 and YOLOv7 Object Detection Algorithms" to be used in the research portion of our paper. Proper credit and citation will be applied.

Thank you,
Travis Grant



SR Sri Tulasi Ram <sales@cytron.io>
To: Travis Grant
Mon 6/9/2025 9:41 PM
Retention: UCF Delete after 10 Years (10 years) Expires: Thu 6/7/2035 9:41 PM

Hi Travis Grant,

Thank you for your patience.

I have consulted with my team, and I can confirm that there should be no issue with using the images from the article "YOLO vs SSD: A Detailed Comparison for Object Detection" as long as you provide proper credit and citations in your thesis.

Let us know if you need any further clarification. Best of luck with your capstone project!

Best regards,

Sri Tulasi Ram
Cytron Support Team

...

Reply Forward

OO oluwaseyi olorunsola <seysisola25@yahoo.com>
To: Travis Grant
Tue 6/10/2025 11:53 PM
Retention: UCF Delete after 10 Years (10 years) Expires: Fri 6/8/2035 11:53 PM

Hello Travis Grant.

You may please use the attached photo in your research project provided you cite the authors of the image (Nepal & Eslamiat, 2022).

Best Wishes

Olorunshola Oluwaseyi Ezekiel, PhD

...

Attach a file

Appendix C | ChatGPT Prompts and Outcomes

5.1.1 Case Study 1

Prompt: “Tell me about MCUs”

Response:

What is an MCU?

A Microcontroller Unit (MCU) is a small, self-contained computer on a single integrated circuit (IC) designed to perform specific tasks in embedded systems. It typically includes:

- A CPU (central processing unit) – processes instructions
- RAM – temporary data storage
- Flash Memory – stores firmware (program code)
- I/O ports – connect to external sensors, actuators, and peripherals
- Timers, ADCs, communication interfaces – enable real-world interaction

What do MCUs Do?

MCUs are used to control hardware in real-time. They’re found in thousands of devices you use every day, from smartwatches and appliances to cars and industrial machines.

Typical MCU tasks include:

- Reading sensor data
- Controlling motors, lights, displays
- Managing timing (with interrupts and timers)
- Handling basic decision-making based on inputs

Table 5.1.1: Examples of Popular MCUs

Family	Brand	Notes
--------	-------	-------

MSP430	Texas Instruments	Ultra-low power, good for sensing and control
STM32	STMicroelectronics	ARM Cortex-M based, powerful and versatile
ATmega328P	Microchip	Easy for beginners, widely supported
ESP32	Espressif	MCU + Wi-Fi/BLE, great for IoT

Advantages of MCUs

- Low power – ideal for battery-operated devices
- Small size – fits into compact systems
- Real-time control – responds to hardware events with low latency
- Cost-effective – many are under \$5
- Highly customizable – code tailored for specific hardware tasks

Limitations of MCUs

- Limited processing power – not suitable for tasks like full image processing or AI inference
- Constrained memory – not good for running large operating systems
- Specialized development tools – often need IDEs like STM32Cube, Code Composer, or MPLAB

Table 5.1.2: *MCU vs. CPU vs. Microprocessor*

Component	Function	Use Case
MCU	CPU + memory + peripherals in one	Embedded control (sensors, motors)
CPU	Just the processing unit	General-purpose computing (desktop, server)
Microprocessor	CPU with external memory and I/O	PCs, Raspberry Pi, smartphones

[22]

5.1.2 Case Study 2

Prompt: “What are the benefits of using switching regulators vs LDO regulators?”

Response:

Great question! Both voltage regulators and LDOs (Low Dropout Regulators) serve the purpose of providing a stable output voltage, but they differ in efficiency, complexity, heat dissipation, and use cases.

Voltage Regulator Types

When people say “voltage regulator”, it can refer to two main categories:

1. Linear regulators (e.g., 7805, LM317, LDOs)
2. Switching regulators (e.g., LM2576, buck/boost converters)

An LDO is a type of linear regulator, so let's compare LDOs vs. Switching Regulators since that's usually the decision point in a design.

LDO (Low Dropout Regulator)

Advantages

- Simplicity: Minimal components – just a couple of capacitors
- Low noise: No switching noise; ideal for analog/RF circuits
- Fast transient response
- Cheap and compact: Small footprint, easy to design in
- Low dropout: Can regulate with very little headroom (e.g., 3.3V output from 3.6V input)

Disadvantages

- Inefficient at high voltage drops or high current
- Efficiency = V_{out} / V_{in}
- Example: 5V to 3.3V - 66% efficient
- Heat dissipation: Excess voltage is burned off as heat
- Limited current: Typically a few hundred mA, some up to ~1.5A

Switching Regulators (Buck, Boost, Buck-Boost)

Advantages

- High efficiency (80-95%+) even with large voltage differences
- Wide input voltage range
- Can step-up (boost) or step-down (buck) the voltage
- Higher current output – several amps or more

Disadvantages

- More complex: Requires inductors, diodes, and control logic
- Switching noise: Can interfere with sensitive analog circuits
- Larger PCB area (especially for through-hole parts like LM2576)
- Slower transient response in some cases

Table 5.1.3: *When to Use LDO or Switching Regulator*

Use Case	Best Option
Small voltage drop (e.g., 3.6V - 3.3V)	LDO
Low current + noise-sensitive analog	LDO
Battery-powered, power-sensitive	Switching Regulator
High voltage drop (e.g., 12V - 3.3V)	Switching Regulator
High current loads (e.g., >1A)	Switching Regulator
Space- and cost-sensitive designs	LDO (if efficiency isn't critical)

Example

If you're powering an ESP32 (3.3V, 300 mA peak) from 12V, then:

- LDO - low efficiency, lots of heat
- Switching buck converter - much more efficient, cooler, better for battery life

But if you're powering a low-power MSP430 from 3.6V down to 3.3V, an LDO is ideal – it's simpler, smaller, and the power loss is negligible.

[22]

5.1.3 Case Study 3

Prompt: “What are the pros and cons to use these serial interfaces for the Bluetooth module?”

Response:

Absolutely! Here’s a detailed breakdown of the main serial communication interfaces used in embedded systems: UART, SPI, and I2C – with in-depth pros, cons, and ideal use cases.

1. UART (Universal Asynchronous Receiver/Transmitter)

Table 5.1.4: UART Pros

Feature	Explanation
Low pin count	Only 2 main wires: TX and RX. Optional RTS/CTS for flow control.
Simple protocol	No clock signal required. Easy to implement in firmware and drivers.
Asynchronous	Devices don’t need to share a clock – great for connecting two devices running independently.
Point-to-point	Great for direct communication between two devices (e.g., MCU - Bluetooth module).
Wide support	Almost every microcontroller and peripheral supports UART natively.
Variable baud rates	Flexible speeds from 300 bps to several Mbps (usually 115200-1M baud).

Table 5.1.5: UART Cons

Feature	Explanation
---------	-------------

Only 1-to-1	Not inherently multi-device. Can't share a UART bus like I2C.
No error detection/correction	Basic framing/parity only. CRC must be handled in software.
Not synchronized	Requires accurate baud rate matching.
Lower throughput	Compared to SPI, data rates are lower (usually <1 Mbps reliably)

Best For

- Connecting to BLE modules
- GPS receivers
- Debugging/console output
- Simple, low-speed serial devices

2. SPI (Serial Peripheral Interface)

Table 5.1.6: SPI Pros

Feature	Explanation
Very Fast	Data rates can exceed 10-50 Mbps – great for high-speed data transfers.
Full duplex	Simultaneous send and receive (MOSI and MISO).
Simple protocol	No addressing; just select the device with CS (Chip Select) and send bytes.
Flexible	Works well with multiple slaves (if you have extra CS lines).
Precise control	You fully control the clock timing, data edges, and protocol framing.

Table 5.1.7: SPI Cons

Feature	Explanation
More pins	Needs at least 4 wires: MOSI, MISO, SCLK, CS. One CS per slave.
No standard protocol	Unlike I2C/UART, you must define the protocol (command formats, timing, etc.)
Shorter cable distances	Susceptible to noise if traces/wires are long or not impedance-controlled.
No inherent device addressing	All devices must have their own dedicated CS line.

Best For

- High-speed devices: flash memory, displays, ADCs
- Short-range, high-bandwidth communication
- Real-time control systems

3. I2C (Inter-Integrated Circuit)

Table 5.1.8: I2C Pros

Feature	Explanation
Two wires	Only needs SCL (clock) and SDA (data) regardless of number of devices.
Built-in addressing	Every device has a 7- or 10-bit address. Master can talk to many slaves on the same bus.
Multi-master capable	More than one controller can exist on the bus (with care).
Ack/Nack support	Protocol supports basic acknowledgement and flow control.
Good for sensors	Many sensors (IMUs, temperature, etc.) are designed to use I2C.

Table 5.1.9: I2C Cons

Feature	Explanation
Slower	Standard = 100 kHz, Fast = 400 kHz, Fast+ = 1 MHz, High Speed = 3.4 MHz (but not widely used).
Requires pull-up resistors	Needs 4.7kOhm (typical) on SDA and SCL lines for correct operation.
Noisy in longer runs	Not ideal for distances over ~1 ft without shielding or buffering.
One device can block bus	If a slave holds the bus low, the master can't communicate with others.

Best For

- Connecting multiple low-speed devices (sensors, EEPROMs, RTCs)
- Short-distance board-level communication
- Simpler MCU-to-sensor integration

Quick Comparison Table**Table 5.1.10: "UART vs SPI vs I2C"**

Feature	UART	SPI	I2C
Wires	2 (TX/RX)	4+ (MOSI, MISO, SCLK, CS)	2 (SDA, SCL)
Devices Supported	1-to-1	1 master, many slaves (each needs CS)	Many (addressed)
Speed	Low-Medium (~1 Mbps)	High (>10 Mbps)	Low-Medium (~400 kbps)
Full Duplex	Yes	Yes	No

Protocol Simplicity	Very easy	Protocol must be defined	Well-defined
Error Handling	Basic (parity)	Manual	Basic (ACK/NACK)
Ideal Use Case	Simple 1:1 comms	Fast peripherals	Multiple sensors

So, For Your Project:

You're sending moderate data (GPS, timestamps, maybe flags/images) to a Bluetooth 5.0 module – go with UART unless your module explicitly supports SPI (very rare for BLE modules).

[22]

Appendix D | Codes

Testing RGB Camera and Trained YOLOv5n Model

```
1 import cv2
2 import numpy as np
3 import onnxruntime as ort
4 import time
5
6 camera_index      = 1 # SWITCH ON RPS
7 width, height     = 1920, 1200
8 fps               = 15 #CHANGE TO 30FPS ON RPS
9 onnx_model_path   = r"C:\Users\travi\yolov5\runs\train\pothole_yolov5n_v2\weights\best-int8.onnx" # SWITCH PATH ON RPS
10 confidence_thresh = 0.8
11 input_size        = 640
12 pothole_count     = 0
13
14 # Debugging RGB gain as needed
15 blue_gain = 1.00
16 green_gain = 1.00
17 red_gain  = 1.00
18
19 # Load model using defined path
20 session = ort.InferenceSession(onnx_model_path, providers=["CPUExecutionProvider"])
21 input_name = session.get_inputs()[0].name
22
23 # Open Camera with MSMF Driver (SWITCH DRIVER WHEN USING RP OS)
24 cap = cv2.VideoCapture(camera_index, cv2.CAP_MSMF)
25 cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
26 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
27 cap.set(cv2.CAP_PROP_FPS, fps)
28
29 if not cap.isOpened():
30     raise RuntimeError("Cannot open camera via MSMF backend")
31
32 # Show what driver, resolution, and FPS was actually used
33 print("Using backend:", cap.getBackendName())
34 print("Actual resolution:", cap.get(cv2.CAP_PROP_FRAME_WIDTH),
35       "\t", cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
36 print("Actual FPS:", cap.get(cv2.CAP_PROP_FPS))
37
```

```
38 # Preprocess
39 def preprocess(frame):
40     r = cv2.resize(frame, (input_size, input_size))
41     r = cv2.cvtColor(r, cv2.COLOR_BGR2RGB)
42     r = r.astype(np.float32) / 255.0
43     r = np.transpose(r, (2, 0, 1))[None, ...]
44     return r
45
46 # Main
47 last = time.time()
48 while True:
49     ret, frame = cap.read()
50     if not ret:
51         print("Frame grab failed")
52         break
53
54 # Debug for RGB issues as needed
55 f32 = frame.astype(np.float32)
56 b, g, r = cv2.split(f32)
57 frame = cv2.merge([b*blue_gain, g*green_gain, r*red_gain])\
58         .clip(0,255).astype(np.uint8)
59
60 cv2.imshow("MSMF Camera Feed", frame)
61
62 # Calculate Real FPS
63 now = time.time()
64 if now - last < 1.0/fps:
65     if cv2.waitKey(1) & 0xFF == ord('q'):
66         break
67     continue
68 last = now
69
70 # Print Pothole Count in Terminal
71 inp = preprocess(frame)
72 outs = session.run(None, {input_name: inp})[0]
73 for det in outs[0]:
74     conf = float(det[4])
```

```

75     class_id = int(det[5])
76     if class_id==0 and conf>confidence_thresh:
77         pothole_count += 1
78         x,y,w,h = det[0:4]
79         print(f"Pothole {pothole_count}: W={w:.1f}, H={h:.1f}, C={conf:.2f}")
80
81     if cv2.waitKey(1) & 0xFF == ord('q'):
82         break
83
84 cap.release()
85 cv2.destroyAllWindows()

```

Quantization of YOLOv5n Model

```

1  from onnxruntime.quantization import quantize_dynamic, QuantType
2
3  quantize_dynamic(
4      model_input='C:/Users/travi/yolov5/runs/train/pothole_yolov5n_v2/weights/best.onnx',
5      model_output='C:/Users/travi/yolov5/runs/train/pothole_yolov5n_v2/weights/best-int8.onnx',
6      weight_type=QuantType.QInt8
7  )

```

MSP430 SPI Connection Demo

```

SPI Connection Demo > main.c > main
1  #include <msp430.h>
2
3  #define RED_LED BIT0
4  #define GREEN_LED BIT7
5
6  volatile unsigned char rx_byte = 0;
7  volatile unsigned int spi_received = 0;
8
9  int main(void)
10 {
11     WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
12     PM5CTL0 &= ~LOCKLPM5;       // Unlock GPIOs
13
14     // Configure LEDs
15     P1DIR |= RED_LED;            // Set P1.0 (red LED) as output
16     P9DIR |= GREEN_LED;          // Set P9.7 (green LED) as output
17     P1OUT &= ~RED_LED;           // Turn off red LED initially
18     P9OUT &= ~GREEN_LED;         // Turn off green LED initially
19
20     // Startup indicator: flash green LED 3 times
21     int i;
22     for (i = 0; i < 3; i++) {
23         P9OUT |= GREEN_LED;
24         delay_cycles(cycles: 200000);
25         P9OUT &= ~GREEN_LED;
26         delay_cycles(cycles: 200000);
27     }
28
29     // Configure UCA0 SPI pins with confirmed working pins:
30     // P2.0 = UCA0SIMO (MOSI from Pi) - confirmed working
31     // P2.1 = UCA0SOMI (MISO to Pi) - to be tested
32     // P2.2 = UCA0CLK (SCLK from Pi) - confirmed working
33
34     P2SEL1 &= ~(BIT0 | BIT1 | BIT2); // Clear SEL1 first
35     P2SEL0 |= BIT0 | BIT1 | BIT2;    // Set SEL0 for UCA0 function
36
37     // Let eUSCI control pin direction
38     P2DIR &= ~(BIT0 | BIT1 | BIT2);

```

```

40 // Configure eUSCI_A0 for SPI slave mode
41 UCA0CTLW0 = UCSWRST; // Hold eUSCI in reset
42 UCA0CTLW0 |= UCSYNC; // Synchronous mode (SPI)
43 UCA0CTLW0 |= UCMSB; // MSB first
44 UCA0CTLW0 &= ~UCMST; // Slave mode
45 UCA0CTLW0 &= ~(UCCKPL | UCCKPH); // SPI mode 0 (CPOL=0, CPHA=0)
46
47 UCA0CTLW0 &= ~UCSWRST; // Release eUSCI from reset
48
49 // Clear any pending interrupts
50 UCA0IFG &= ~(UCRXIFG | UCTXIFG);
51
52 // Enable RX interrupt
53 UCA0IE |= UCRXIE;
54
55 // Enable global interrupts
56 __enable_interrupt();
57
58 // Main loop
59 while (1) {
60     // Green LED blinks every 2 seconds to show we're alive
61     P9OUT |= GREEN_LED;
62     delay_cycles(cycles:100000); // Short on pulse
63     P9OUT &= ~GREEN_LED;
64     delay_cycles(cycles:1900000); // Long off period
65
66     // If SPI data was received, flash red LED once
67     if (spi_received) {
68         P1OUT |= RED_LED;
69         delay_cycles(cycles:200000);
70         P1OUT &= ~RED_LED;
71         spi_received = 0; // Clear flag
72     }
73 }
74 }

```

```

76 // SPI interrupt service routine
77 #pragma vector=USCI_A0_VECTOR
78 __interrupt void USCI_A0_ISR(void)
79 {
80     switch (__even_in_range(val:UCA0IV, range:USCI_SPI_UCRXIFG))
81     {
82     case USCI_SPI_UCRXIFG: // RX buffer full
83         rx_byte = UCA0RXBUF; // Read received byte (clears interrupt flag)
84         spi_received = 1; // Set flag for main loop
85         // Just receive - don't send anything back
86         break;
87     default:
88         break;
89     }
90 }

```