ELEC 490/498 Final Report

Secure Tactical Environmental Vigilance Engine
STEVE

Group 17

Group members:

1) Amy Li, 19awl2, 20220162

2) Cassidy Samoyloff, 19crs1, 20185749

3) Fenson Lin, 19fl29, 20226624

4) Kay Burnham, 19kob1, 20220414

Email to faculty supervisor(s): Dr. Joshua Marshall

Submission date: April 7, 2025

Contents

1. Introduction

This final report outlines the full design, development, and construction of Group 17's ELEC 49X capstone project: *Secure Tactical Environmental Vigilance Engine (STEVE)*. STEVE is a small-scale, indoor security robot designed to monitor controlled environments such as university buildings or offices without the need for constant human supervision. The system combines basic mobility, a live video feed, and environmental sensing into one affordable platform. This report is prepared for course instructors, including the faculty supervisor Dr. Joshua Marshall, as well as potential stakeholders such as Queen's University Campus Security and Emergency Services and the City of Kingston.

1.1 Motivation and Background

In modern days, there has been growing interest in using robotics to support security and monitoring tasks, especially in indoor settings where fixed surveillance systems may have blind spots or limited flexibility. While high-end commercial solutions for mobile surveillance exist, they are often expensive and difficult to scale for use in academic or public settings.

STEVE was developed with affordability and accessibility in mind. By using common components like a Raspberry Pi, ultrasonic sensors, and a Wi-Fi-enabled camera, the design keeps costs low while still delivering useful functionality. The total budget for the system falls under $450 CAD—significantly cheaper than even the base models of many existing security bots on the market.

In addition to its practical goals, the project serves as a learning experience for applying engineering principles in a real-world context. It brings together multiple areas of knowledge—circuit design, CAD modeling, embedded programming, and wireless communication—into one cohesive system.

The need for flexible, low-cost surveillance options is increasingly relevant, particularly for institutions managing large buildings or campuses. STEVE aims to fill this gap with a lightweight, student-developed solution. The need for flexible, low-cost surveillance options is increasingly relevant, particularly for institutions managing large buildings or campuses like Queen's University. STEVE aims to fill this gap with a lightweight, student-developed solution. While previous systems like the one by Rahouma et al. [1] focus more on smart home surveillance robots, and others like the study by Meddeb et al. [2] focus on IoT integration, STEVE's approach is intentionally simpler and more accessible. STEVE is designed with 3D-printed parts and affordable components that can be easily sourced and assembled — it requires only access to a smartphone/mobile device in terms of biometric recognition. The main goal was to build something functional but also achievable for student teams or educational use, where cost and ease of development are just as important as performance.

2. Design

2.1 Functional Requirements and Constraints

The Secure Tactical Environmental Vigilance Engine (STEVE) was initially proposed as a compact, autonomous security robot capable of performing key surveillance functions without direct human intervention.

Core functional requirements outlined at the start of the project included:

- Autonomous navigation using LiDAR and ultrasonic sensors for object detection and collision avoidance
- Low-latency video streaming via Wi-Fi to a user-facing mobile interface
- Sound level detection for monitoring environmental noise or unusual activity
- The ability to perform omnidirectional movement

As the design progressed, some features proved more complex to integrate within the time and hardware constraints. As a result, certain originally planned functionalities were reclassified as stretch goals**,** including:

- Sound location triangulation
- Notification alerts within the mobile app
- Semi-randomized path-following for surveillance routes

Additionally, the sound detection and autonomous navigation subsystems were later deprioritized due to implementation challenges, joining the list of stretch goals despite being part of the initial scope

Constraints included:

- A project budget capped at approximately $420
- Voltage limitations, requiring separate power lines for the Raspberry Pi and the motor system

- A maximum chassis size of 21cm x 25cm to conform to the capabilities of the 3D printer being used
- Processing limitations on the Raspberry Pi, requiring lightweight software solutions
- Adherence to ISO 26262-3:2011 for safety during testing, particularly in terms of motion control and system shutdown protocols

These constraints shaped design decisions and ultimately defined the scope of the final deliverable — a streamlined, low-latency video surveillance bot optimized for indoor environments.

2.4 Subsystems Identification

Each tool and technology listed above was selected based on a combination of familiarity, availability, and reliability. SolidWorks was used for CAD modeling because most team members had prior experience with it from coursework, and it integrates well with the 3D printing workflow. FDM and resin printing were chosen to balance structural durability with fine detail printing for component mounts.

The Raspberry Pi was selected as the main controller due to its accessible GPIO layout and compatibility with both circuit components and camera modules. Breadboards were used during development for flexibility in wiring and iteration.

ROS and Python were initially proposed for navigation and sensor integration. While ROS was ultimately not implemented, its inclusion in the plan reflected industry-standard practice and provided a structure for future work. Python was chosen for prototyping sensor logic due to its simplicity and the team's familiarity with it.

The Arducam and ReSpeaker microphone array were chosen for their plug-and-play compatibility with the Raspberry Pi and their performance in capturing clear audio/video data. Finally, Android Studio was used for UI development because one of the group members had prior experience building mobile interfaces and could leverage that skillset.

2.5 Subsystems Analysis and Implementation

2.5.1 CAD & Chassis Design

The chassis was modeled in SolidWorks with the specific constraint of staying within a 24x24 cm footprint to meet printing limitations. The design includes a lower base that connects electronic components and houses the wiring, and an upper cover that provides protection while maintaining accessibility for future maintenance or upgrades. PLA filament was used for structural body components due to its balance of strength and ease of printing, while smaller parts such as sensor mounts were printed in resin for improved precision. The design evolved through multiple iterations to incorporate bottom-side ventilation, modular mounting slots for easy hardware swaps, and side guards to shield wiring and internal electronics from external impact.

2.5.2 Circuit Design

A modular breadboard setup was selected for the development phase to allow flexible and rapid prototyping of the circuit. Components were separated into two power domains: a 5V line for the Raspberry Pi and a 6V line for the motors provided by an external battery pack. This prevented issues caused by power surges or voltage drops, which were observed during early bench tests. ISO 26262 safety principles were followed to ensure safe operation during testing phases — including limiting maximum current, reducing motor

speed, and adding isolation where necessary. The circuit setup also facilitated plug-and-play testing for various sensors and peripherals.

2.5.3 Navigation and Sensing

Originally, this subsystem was planned to integrate LiDAR and ultrasonic sensors for autonomous navigation. These components were meant to feed distance and obstacle data into a ROS-driven control system, enabling the robot to follow a predefined path or avoid collisions dynamically. Although full ROS integration was deferred, the subsystem was still analyzed, and a Python-based alternative was briefly prototyped. The sensors were evaluated for range, accuracy, and positioning — with the ultrasonic modules intended for close-range feedback and LiDAR for long-range mapping. Due to timeline constraints, this subsystem was later removed from the MVP.

2.5.4 Sound Detection

A ReSpeaker USB microphone array was selected to allow for multi-directional audio input. The goal was to detect anomalous noise levels, triggering a system response or alert. The design plan included a simple sound processing routine to identify spikes in volume that may correspond to unexpected events (e.g., a door slamming, shouting, or glass breaking). Triangulation features were explored using the microphone's channel array, but were marked as stretch goals due to complexity. Although sound detection was deprioritized later in development, the subsystem remains compatible with future upgrades and was fully planned in the initial system design.

2.5.5 Camera and Video Streaming

For visual monitoring, the B0309 Arducam was selected due to its high compatibility with

the Raspberry Pi and its wide-angle field of view. The stream was intended to run via MJPEG through the Raspberry Pi's onboard Wi-Fi, allowing for near real-time video transmission to a mobile application. This subsystem was central to the robot's surveillance role and prioritized from the beginning. Resolution, frame rate, and transmission latency were all tested in the design phase to ensure they met usability thresholds (<3 seconds latency). This module operated independently of navigation, making it viable even when autonomous driving was removed.

2.5.6 User Interface and API Integration

The user interface was developed for Android using Android Studio, with the goal of displaying a live video feed and potentially delivering alerts to the user. Communication between the Raspberry Pi and the app was structured over local Wi-Fi, making use of standard IP-based streaming protocols. From a system design standpoint, the UI had to be lightweight and responsive, capable of adapting to live video input while leaving room for future feature expansion (such as directional controls or event notifications). This subsystem bridged the gap between hardware function and user interaction and was a critical part of overall project usability.

2.6 Modeling and Simulation

2.6.1 CAD Modeling

Each chassis component was iteratively modeled and test-printed using SolidWorks. Care was taken to ensure the chassis would fit on a 3D printing bed without being divided into more parts than strictly necessary. This resulted in size limitations that affected later construction, including the limit on internal power. Recesses were made in the top chassis

and wheelbase parts to facilitate installation of the breadboard, R-Pi, and LiDAR/microphone mount, as seen in section 3.3.

2.6.2 Circuit Construction

Basic electrical simulations were carried out to verify compatibility between components and ensure stable power distribution. A multimeter was used to measure voltage and current during early-stage testing. One of the early test setups included a microphone triggering a basic LED response, validating input-output flow before final wiring.

The Rpi5 has two usable MIPI display / camera connectors, each accepting a 22 pin ribbon cable. The camera model used was highly compatible, immediately being recognized in the default port once added to the configuration setup of the board.
Motors ran in parallel with an external 6V battery supply that shared a common ground to the Rpi, while each motor had a direct connection to an on board DPIO port. This allowed them to be accessed directly by Python script within the Rpi operating system based on their respective pin locations.

The system was powered by an outlet via usb-c cable. For testing, the bot was required to be connected to the power supply to function. In a real life application this would not be the case, and an on board power supply would be fitted to the design.

While they were not implemented in the final prototype, the LiDAR and microphone array were compatible with the Rpi5 via USB ports.

2.6.3 Software Simulation

Significant testing was done on the software side to simulate real-world streaming and connectivity. Multiple latency configurations were tested by varying resolution, bitrate, and frame rate, to optimize performance on lower-powered devices. Several Wi-Fi routers were used in different environments to assess stability and consistency, as the strength and quality of the connection directly impacted stream reliability.

In addition, network firewall settings were carefully adjusted to allow broadcast traffic between the Raspberry Pi and the Android device. This included port forwarding and temporary disabling of default firewalls on testing laptops to ensure the MJPEG stream could be received properly. Final latency values were tested both in emulation using Android Studio's simulator and on a physical device, providing realistic performance metrics for live usage.

2.7 Key Design Decisions

2.7.1 Chassis Size and Constraints

The chassis was designed to remain within a 24x24 cm footprint for the sake of single element printing. This allowed the design to avoid any joints that could pose as a point of failure due to the weight the chassis would be supporting. The smaller form factor also simplified transportation and bench-top testing throughout the design phase. If implemented in a commercial environment, this size would have to be scaled up to support on board power systems, and motors with higher speed capabilities.

2.7.2 Independent Power Lines

To avoid voltage instability during operation, separate power sources were used: The raspberry pi was provided power from a wall outlet. While the motors can run on 5-7V, the Rpi output was not used to prevent brown outs and protect the board from the inherent risk of power surges that comes with multiple motors. Instead, an external battery pack supplied 6V in parallel to the motors. This separation minimized brownouts and interference, particularly during periods of high motor activity.

2.7.3 Selection of Software Tools and Libraries

MJPEG-streamer was selected as the primary tool for video streaming due to its low latency, compatibility with Android platforms, and minimal hardware overhead. This decision aligned with the project's goal of maintaining efficient processing on limited hardware.

2.7.4 Late Design Modifications

Several key features were adjusted during the latter half of the project:

- Full integration of the Robot Operating System (ROS) was initially planned but later deferred to focus on core system functionality. Setting up ROS nodes and inter-device communication proved too time-intensive within the project timeline. Future iterations may revisit this integration.
- Autonomous navigation and sound detection were excluded from the final implementation due to time constraints and integration complexity. Although

preliminary development was completed for both, real-world testing showed that additional calibration would be necessary to ensure reliable performance.

- Battery operation was not implemented. Due to size constraints, availability and budget limitations, a suitable battery could not be sourced in time. As a result, the robot was operated using a continuous power connection, reducing mobility but maintaining system consistency for testing and demonstration purposes.

These decisions prioritized system stability and ensured delivery of a streamlined security bot focused on low-latency video streaming — fulfilling the project's core objectives while allowing room for future expansion.

3. Implementation

3.1 Operation

As of the completion of this project, the robot is operated by manually starting and stopping the local program on the R-Pi to manage both livestream and movement. This requires the code to be loaded into active use and a connected keyboard or mouse to run the program. The livestream can then be accessed by opening the app and selecting the appropriate button (“What does STEVE see?”) from the menu.

3.2 Code Blocks

3.2.1 Mobile App

The streaming app was programmed in Android Studio to facilitate real-time visualization of the user-facing interface. It connects over Wi-Fi, starting the feed using a socket connection when the ‘Start Stream’ button is pressed. The incoming frames are handled using a SurfaceView for smooth display. To ensure the stream is done over a secure network, only specific IP addresses were allowed by configuring AndroidManifest.xml and adding instructions in network_security_config.xml. Greater focus was put into streaming configurations on the ROS system rather than the user API to ensure the app is kept lightweight, ensuring low latency and reliable performance on mobile devices.

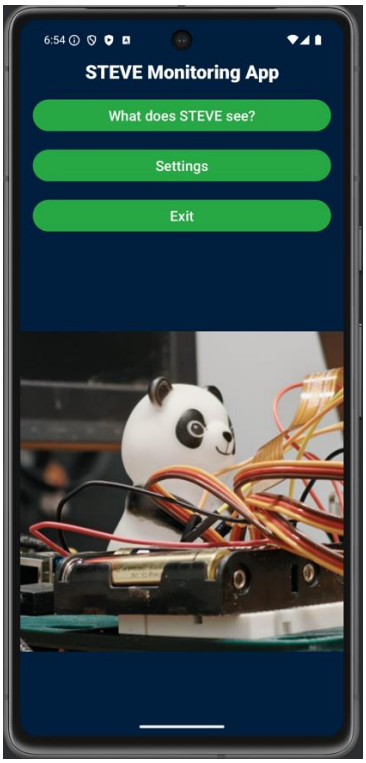

Figure 1. Mobile interface previewed through Android Studio.

3.2.2 Bot Programming

Coding for the bot's livestream broadcast and movement input was written in the native R-Pi environment, using repositories as a basis for setting up IP-hosted streaming and wheel encoding for directional movement. The code used on the bot was documented via Notion.

3.3 CAD

Figure 2 shows the disassembled chassis in three parts. The top frame was designed to fit the LiDAR and microphone array with sufficient space to receive audio input, and to fit exactly into the indented surface of the top shell that protects the internal wiring. The bottom wheelbase was designed to fit the specified breadboard and R-Pi with a built-in camera mount, although this was later replaced by a resin mount provided with the camera that allowed for camera angle adjustment.

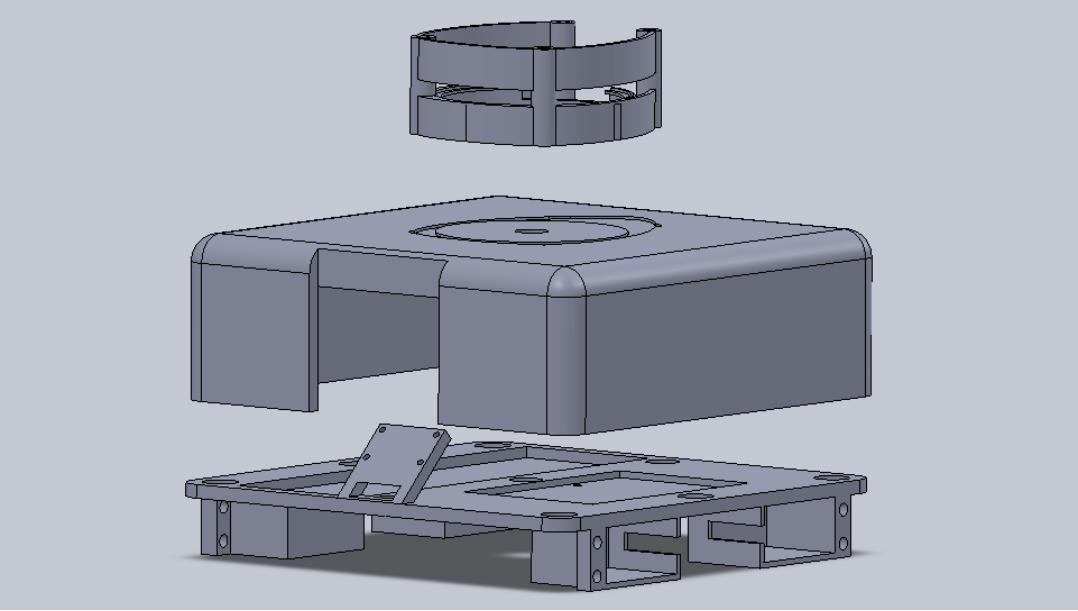

Figure 2. CAD model of the robot chassis.

3.4 Final Assembly

Wiring of all wheels to both the power supply and to the control signals was performed during assembly based on the motor datasheet.

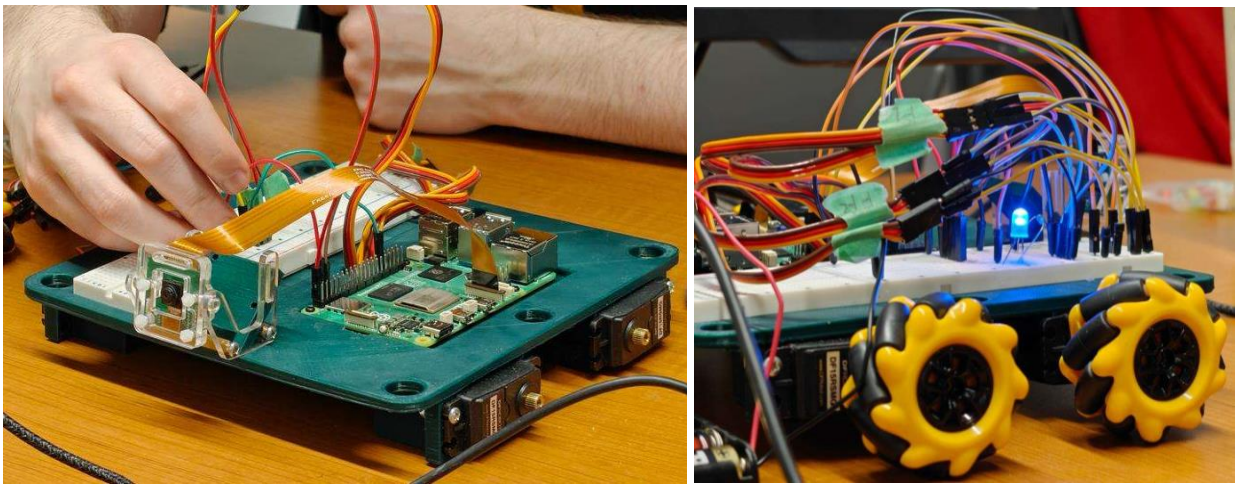

Figure 3a and b. Layout of internal board and completed wiring. The blue LED indicates functioning input power for troubleshooting.

3.5 Budget

Table 1 details a bill of materials for the security robot, including replacement components as previously outlined in section 2.

Table 1. Final bill of materials.

Item	Supplier	Price ($)
Raspberry Pi 5	Amazon	132.80
Lidar	Faculty	0
Microphone Array	Faculty	0
“Mecanum” Wheels (×4)	Amazon	26.45
Motors (×4)	Robot Shop	4 motors × 28.26 = 113.04
B0309 Camera (Arducam)	Robot Shop	60.07
Power Supply	Amazon	50
Bread Board	Amazon	23.99
PLA Filament	Amazon	31.99
Liquid Resin	Amazon	29.99

4. Testing

The testing and evaluation phase was a critical stage in the progress that took up a significant portion of time, which was necessary to ensure STEVE met the design specifications as outlined in the proposal and blueprint, with the main goal being a working prototype capable of low latency video monitoring via a user interface. Furthermore, STEVE needed to demonstrate its ability to operate reliably in different environments with the possibility of scaling for future expansion of this project in real-world scenarios including potential stakeholder requirements. The V-Model and Agile Testing methodologies were used during the iterative development of this robot for both hardware and software performance as outlined in the approach section of the blueprint.

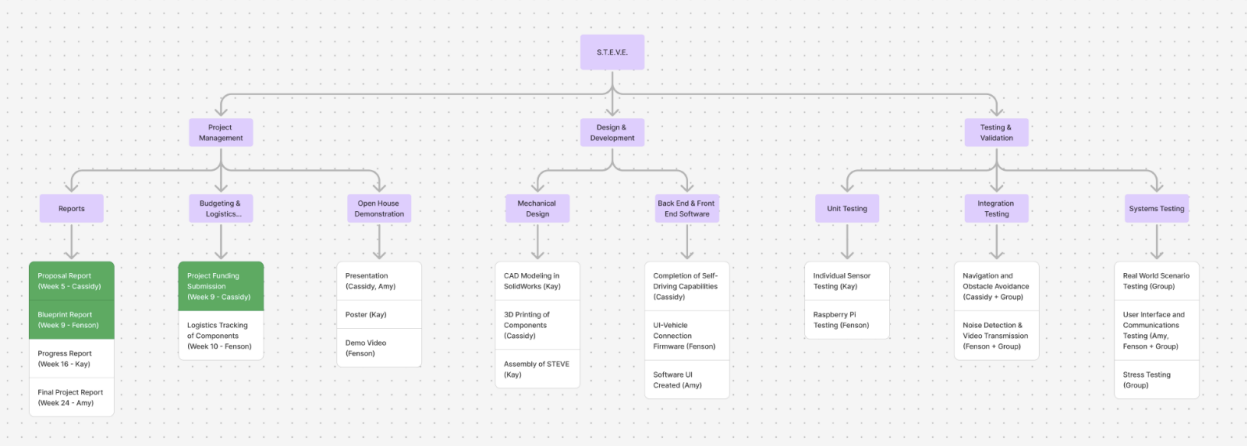

Figure 4. Work Breakdown Structure from Blueprint Report.

4.1 Testing Methodology

To further expand on the testing methodology planned, the team followed a three-tiered testing structure: Unit testing, integration testing, and system testing. Unit testing was used to ensure each of the sensor components were verified to be working independently, while integration testing ensured combining the sensors worked reliably with each other for

specific subsystems such as camera streaming through WiFi. Lastly, system testing was implemented in a controlled real-world environment to demonstrate STEVE's ability to operate without the need for human interaction. Iterative improvements to each component were implemented whenever necessary during each phase, and testing was repeated to make sure new additions did not break existing functionality.

4.2 Unit Testing

Each major component in STEVE was tested individually to validate full functionality before being integrated into the system.

4.2.1 Raspberry Pi 5

The Raspberry Pi 5 served as the central computing unit for STEVE and was chosen due to the flexibility the hardware offered with 8GB of ram, numerous ports for connectivity, the integrated WiFi and Bluetooth chip, a dedicated camera out port, and the extensive documentation online that could aid with the integration process of sensors. A Lexar 128GB MicroSD card was used as the storage and boot drive for this system, using the first party Raspberry Pi Imager tool on MacOS to flash the latest OS. At this point, it was verified the USB-C port was working as intended to power up the device at the proper voltage with the correct power brick, the 4 USB-A ports were working with the peripherals used to control the OS such as the keyboard and mouse, and lastly, the micro-HDMI port which was used to connect to an external monitor. Evidently, the WiFi and Bluetooth module were also confirmed to work properly with the network used to set up the OS, in addition to using Bluetooth accessories such as a wireless mouse.

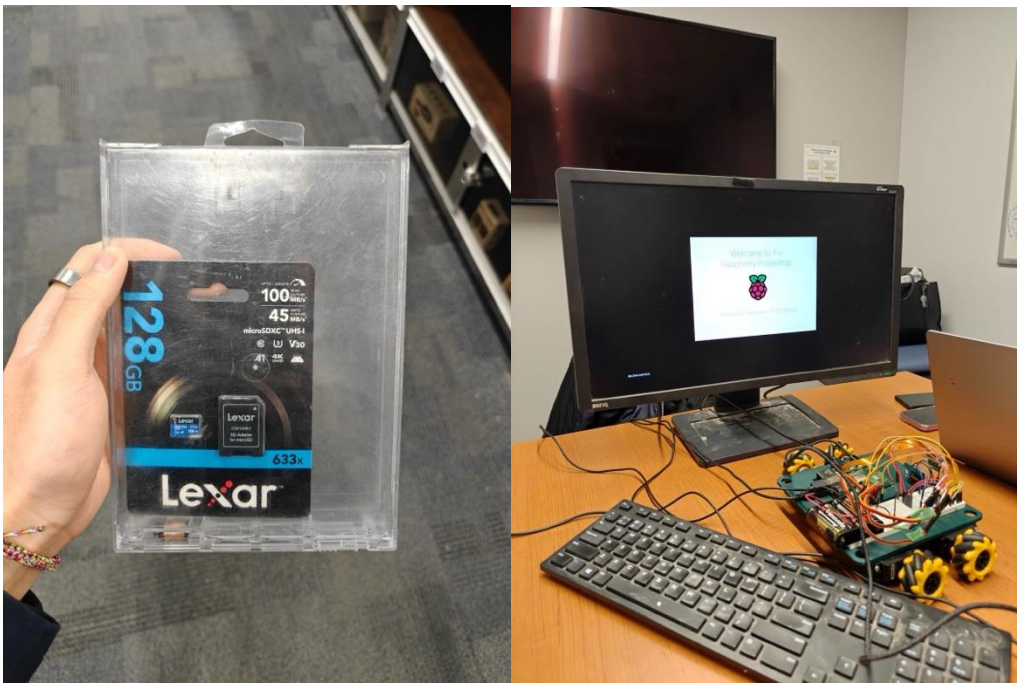

Figure 5. The MicroSD Card Used for RPi 5 and the OS Booting Up.

Docker was then installed to be used as a platform for software to run in isolated environments such as containers that were created for different subsystems within STEVE. The Robot Operating System 2 (ROS 2) was also installed based on the TA's suggestion, as it contained software libraries and tools necessary for building robot applications and allows communication between the sensors that were used with STEVE.

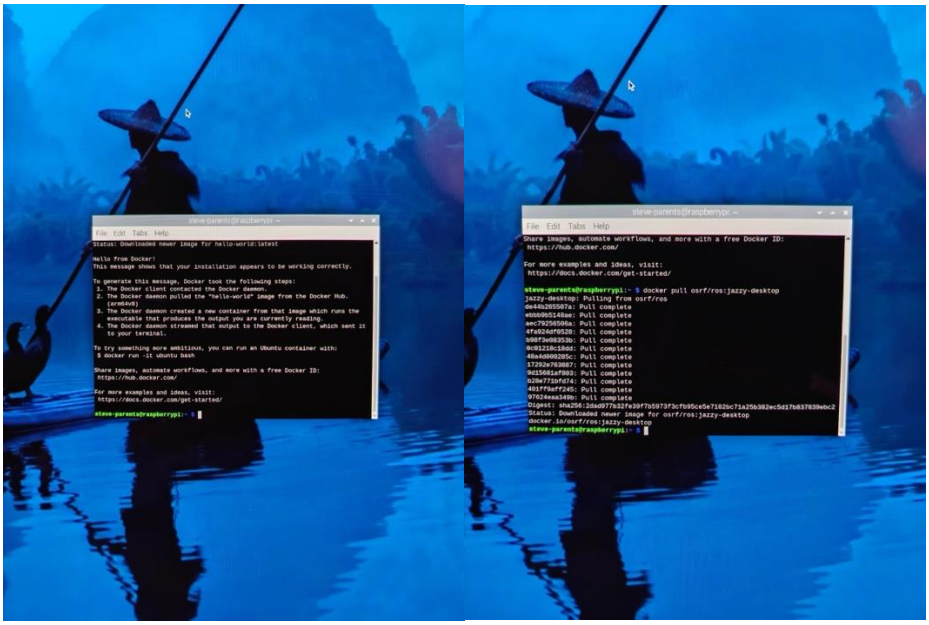

Figure 6. Successful Installation of Docker and ROS 2 in Terminal.

It was then verified that the Raspberry Pi 5 could run the ROS 2 environment smoothly with Docker for an extended period of time by using a simple communication test inside ROS 2. For this test, the ROS 2 talker and listener demo nodes were run inside two separate terminals, in which the talker node repeatedly published the string “Hello World: #”, while the listener node received and printed the messages in real time without any form of communication errors or slow performance.

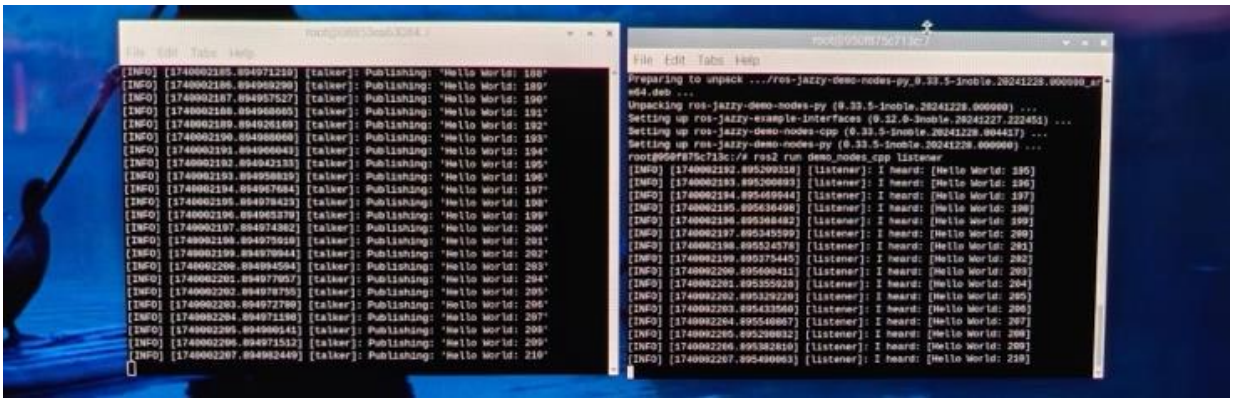

Figure 7. The ROS 2 Talker/Listener Test in Action.

4.2.2 Freenove Raspberry Pi Camera Module Rev 1.3:

The first camera module chosen for this project as outlined in the blueprint's parts list was the Arducam UC-A37 Rev A. However, despite many attempts with this camera module and even using the official documentation, it was not properly being recognized by the Raspberry Pi 5 due to kernel overlay issues, unsupported drivers, and inconsistent V4L2 device detection.

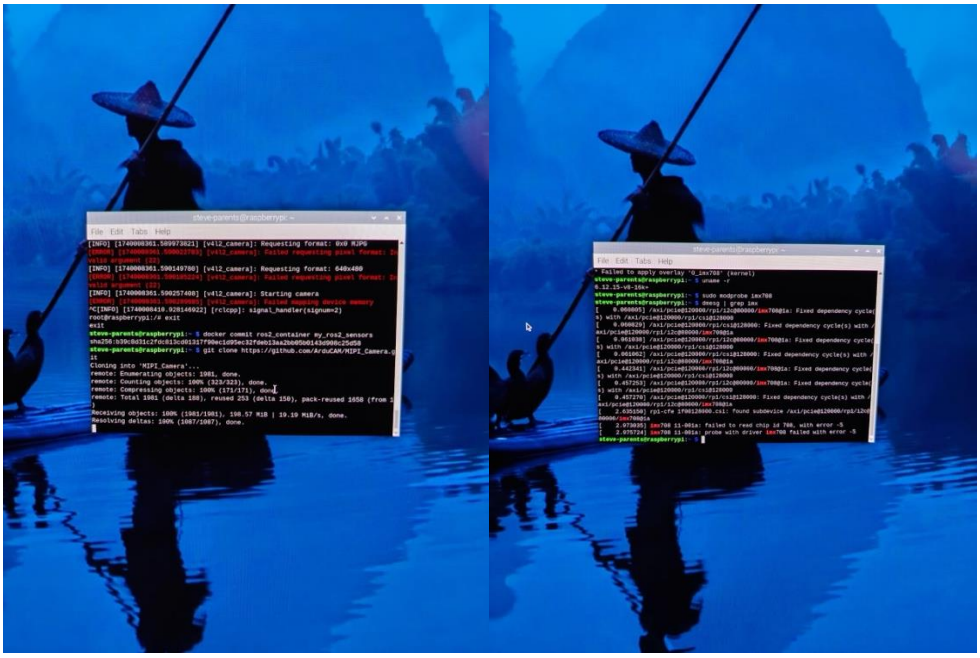

Figure 8. Example of Errors Encountered During Arducam Setup.

The Freenove camera module was then purchased to rule out any hardware issues with the camera ribbons that were used or ribbon cable port directly on the Raspberry Pi 5. Upon plugging in the module for the first time, it was successfully recognized using the libcamera stack. By running a local hardware stream directly from the Raspberry Pi 5 using

GStreamer, it was verified there was a proper video feed output with low latency and a stable frame rate.

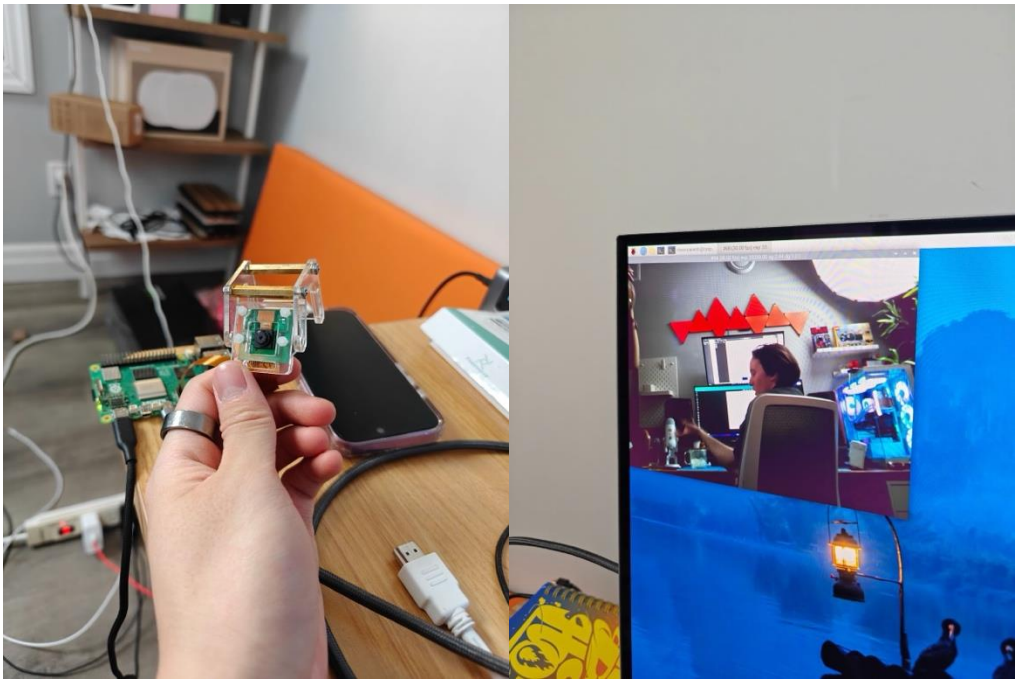

Figure 9. The New Camera Module with Working Output from Local Stream.

4.2.3 ReSpeaker USB Microphone Array v2.1

The ReSpeaker USB Microphone Array provided by faculty supervisor Dr. Joshua Marshall is a plug and play module that connects with the Raspberry Pi 5 through the USB-A port. It features 4 high performance digital microphones that can be used to pick up real-time multi-directional sounds, with existing documentation for speech algorithms and features such as voice activity detection and direction of arrival which could be used for future expansions of this project. With the plug and play nature, a simple “lsusb” command verified the recognition of the microphone array upon a physical connection, which was further verified with the “arecord -l” command used to list out capture hardware devices. A

sample audio recording test was also carried out to validate the ability to record audio using the “arecord” function.

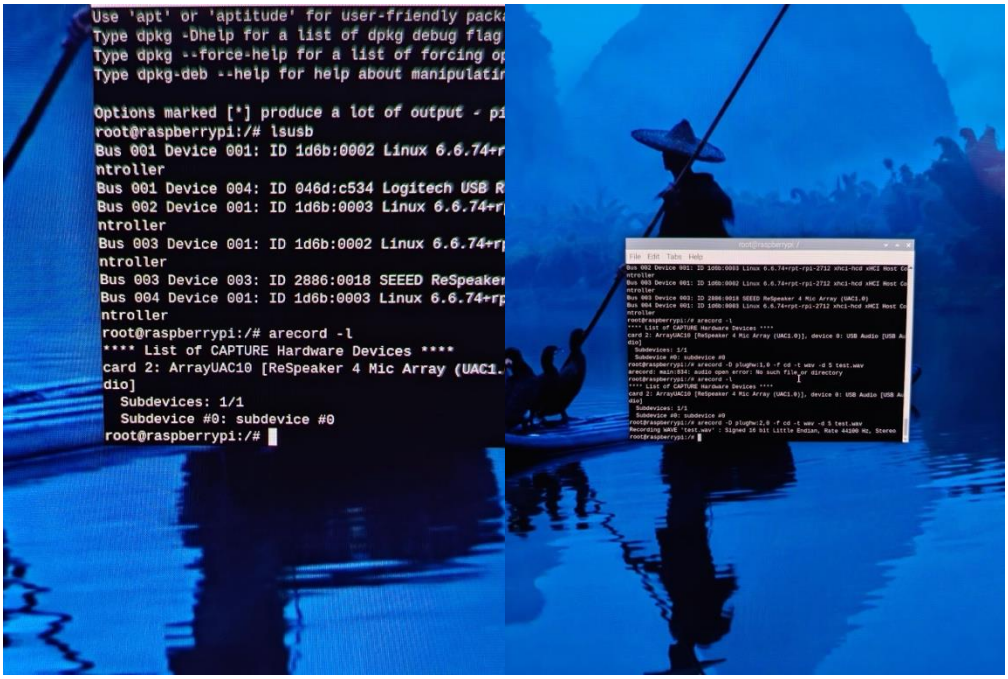

Figure 10. Verification of Mic Detection with Live Recording Executed.

To test the audio file recorded, a pair of headphones was plugged into the microphone array’s 3.5mm headphone jack, in which using the function “aplay”, it was confirmed the recorded audio file was playing, in addition to the 3.5mm headphone jack working as intended with audio output devices. These tests were carried out inside the ROS 2 container and the setup configuration was saved to allow for future iterations of potential stretch goals.

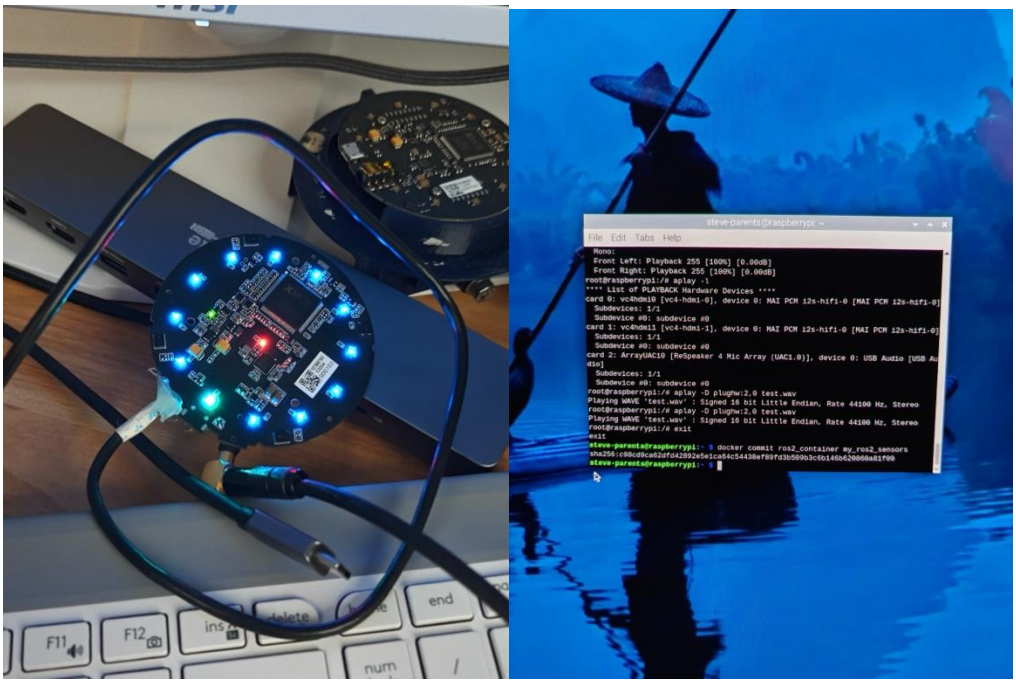

Figure 11. Mic Array Outputting Recorded Audio to Connected Headphones.

4.2.4 RPLIDAR A1

The RPLIDAR A1 360° Laser Range Scanner was also provided by faculty supervisor Dr. Joshua Marshall and similar to the microphone array, is also plug and play compatible. Although not necessary for the scope of the project that was earlier proposed, it was implemented within the design phase of the robot to allow for future iterations of STEVE with potential stretch goals such as autonomous navigation and obstacle detection. This specific sensor was actually developed to be used with robot vacuums, which are capable of real-time mapping of the environment in addition to obstacle detection during navigation which would be ideal for STEVE.

To initialize the LiDAR sensor, the built-in rplidar_ros package inside the ROS 2 environment allowed the startup of this sensor with a launch command. As shown in the terminal output, the LiDAR sensor was successfully initiated, displaying device details such

as the firmware version, health status, and the current scan configuration which can later be adjusted.

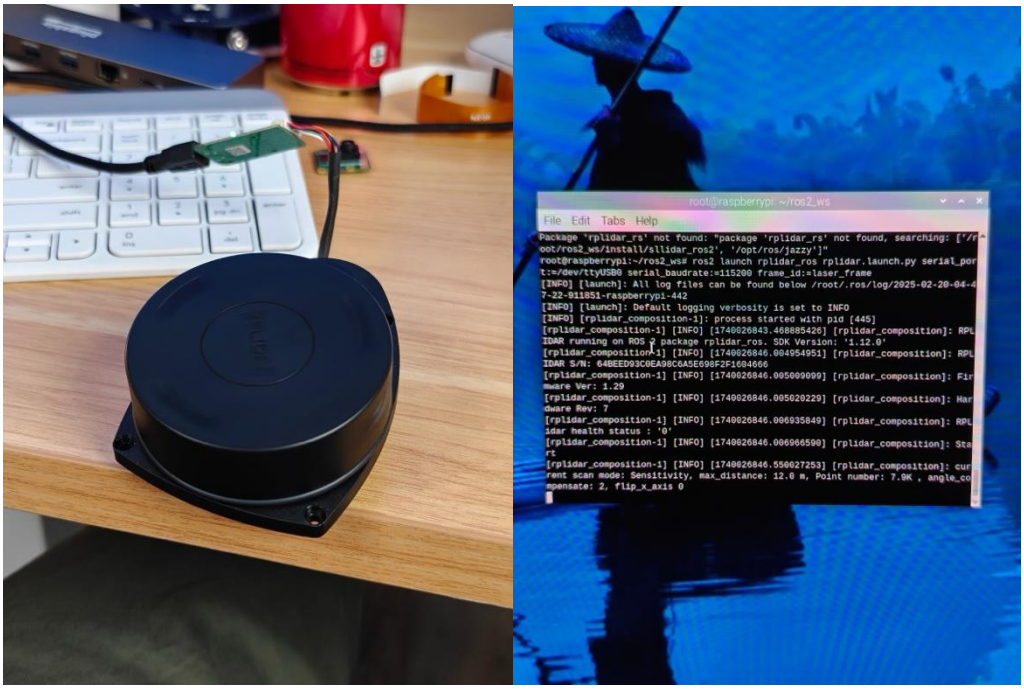

Figure 12. LiDar Sensor in Acition with Terminal Confirmation.

To prepare the LiDAR sensor for future implementation in potential stretch goal functionality, RViz2, a 3D visualization tool in ROS 2, was set up to allow for viewing the sensor data outputted by the LiDAR sensor. By correctly configuring the properties within the visualization tool, this will allow for any scan data to be published directly on the graph, in which it was further confirmed by outputting the live scan rate into the terminal.

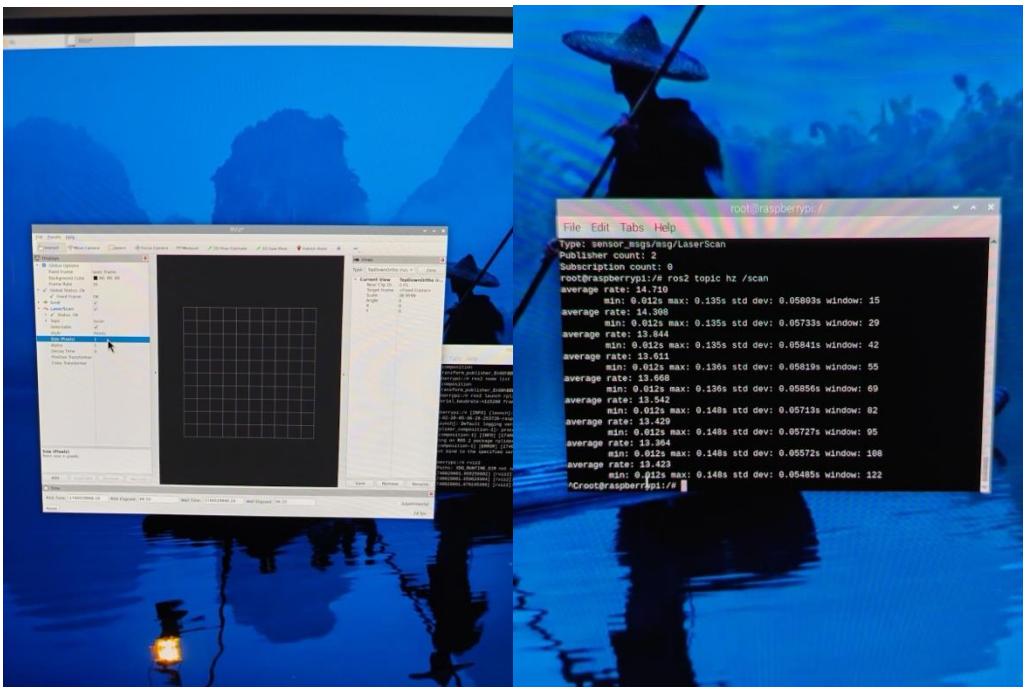

Figure 13. RViz2 Visualization Tool with Live Scan Refresh Rate Data in Terminal.

4.2.5 DFROBOT SER0035 DC Motors

The motors selected were the SER0035 model servo motors for their high torque capabilities despite being a relatively low cost motor, as well as PWM capability. When first acquired, a voltmeter was used to confirm that the motors were all electronically functioning and outputting expected values. They were confirmed to be usable.

Once the motors were connected to the DPIO ports of the Raspberry Pi, PWM signals could be transferred between the motors and built in controller within the Pi. However, one could not be found in the configuration despite the other three working. Originally thought to be an Rpi based problem, the pins were individually tested using motors known to be working. This did not yield any results, and so a voltmeter was once again taken to the fourth motor and proved it was no longer internally conducting.

Despite a missing wheel, the robot was still able to function with basic mobility including straight lines and slight turns. Running at 6V the motors provide about 10 kgcm of torque which proved enough to compensate and support the bot.

4. 3 Integration Testing

The integrated subsystem required for the scope of this project was the live video streaming capabilities of STEVE, in which the live camera feed would be received by the dedicated Android application developed in Android Studio. The primary objective of this subsystem was to enable low latency remote viewing from STEVE over the same WiFi network which was achieved and validated to be working in different controlled environments.

To begin the streaming process, a web hosting server was required, in which Apache2, an open-source HTTP server protocol was implemented. The streaming pipeline was then built using GStreamer, an open-source multimedia framework used to handle audio/video playback, recording, and streaming pipelines within the Raspberry Pi 5. With the functionality of GStreamer, full control over the video format, resolution, frame rate, and encoding settings were also able to be modified to best fit STEVE's use case and after testing various configurations in different environments, these properties were decided as the best fit for STEVE:

- Video Format: NV12
    - NV12 in specific is a type of YUV 4:2:0 planar color format, which was chosen thanks to its high efficiency by using less memory and bandwidth in the system, which allowed for fast responsiveness and reduced processing overhead
- Frame Rate: 30 FPS

    - A frame rate of 30 FPS was chosen to provide a decently smooth video without looking choppy, while preventing frame drops and increased CPU/GPU loads from higher frame rates, which is not ideal for a small system like the Raspberry Pi 5
- Resolution: 1080 px x 810 px
    - This frame rate was chosen primarily to match the display dimensions of the test device (Google Pixel 5) used for emulating the Android Studio application that was developed for displaying the video stream
- Video Scale Method: 6
    - Videoscale method=6 was chosen due to its high quality scaling algorithm that allowed for sharp visuals and readability specifically for lower resolutions, while reducing aliasing and artifacts at the same time
- Encoding Format: x264enc
    - The encoding format x264enc was chosen due to it being highly optimized for 30 FPS streaming, in which the ultrafast and zero latency configuration was used for efficiency
- Container Format: MPEG-TS
    - The container format “mpegtsmux” was used to compress the video outputted by the camera module into a TS stream, which is commonly used for streaming purposes thanks to its support for continuous playback and its wide compatibility with video players
- Network Sink Element: TCP Connection

- Tcpserversink was used to stream the compressed video data through a TCP connection into the specified IP address of the local network specifically through port 8080 for security purposes. During testing, it was found that TCP was the most reliable connection protocol due to its reliability with handling video streams, in addition to its easily implementation within Android Studio and even video players such as VLC which allows for a TCP connection for playback

```
//start apcache2 to start the HTTP server
sudo systemctl start apache2
sudo systemctl enable apache2
sudo systemctl status apache2

gst-launch-1.0 -v libcamerasrc ! video/x-raw, \
format=NV12,width=1920,height=1080,framerate=30/1 \
! videoscale method=6 ! video/x-raw,width=1080,height=810 \
! videoconvert ! x264enc tune=zerolatency bitrate=1000 speed-preset=ultrafast '
! mpegtsmux ! tcpserversink host=0.0.0.0 port=8080 sync=false & disown
```

Figure 14. The Code Used to Initialize the HTTP Server and Camera Stream.

With these specific properties validated as the optimal solution for video streaming, extended testing was successful in terms of streaming to the mobile app on the Google Pixel 5 using different forms of networks such as private home networks and even mobile hotspots from personal mobile devices. This checked off the main objective for the scope of this project, which was the integration of the live camera feed to a mobile application with low latency.

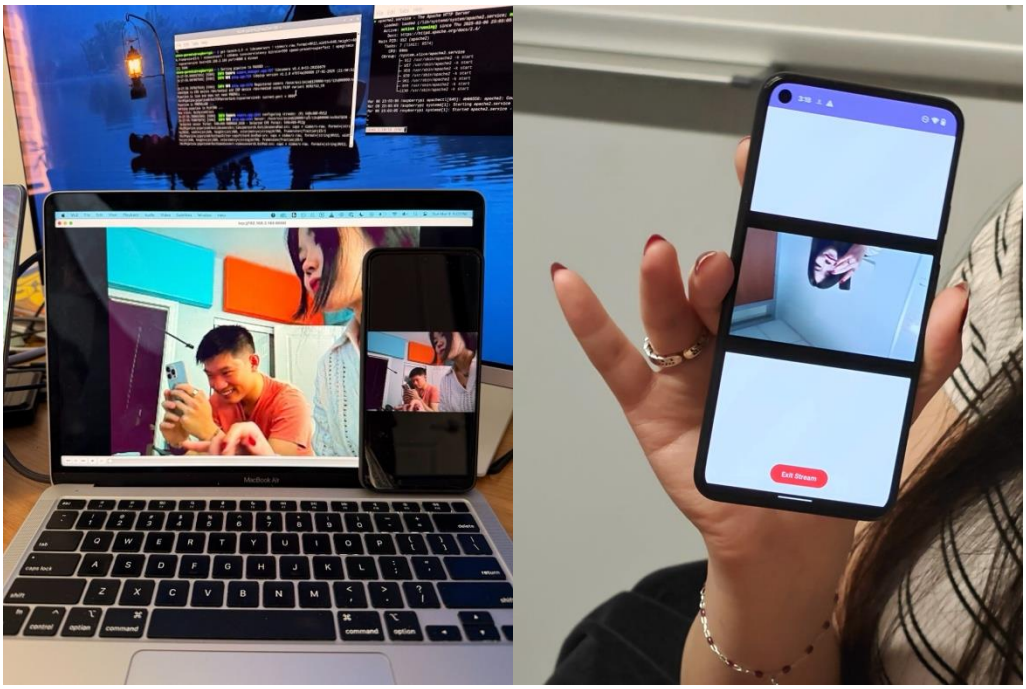

Figure 15. Testing of TCP Streaming of Live Camera Feed with the Live Mobile App Working.

4.4 System Testing

For system testing, the objective was to ensure STEVE could meet the requirements set within the scope of the project, specifically its ability to stream real-time video from the camera feed with basic movement of the robot using the motors and mecanum wheels. The full working system was deployed in a controlled indoor environment where the robot was powered through a wired connection to the outlet and connected to the mobile hotspot of one member's personal mobile device. The mobile app developed was sideloaded onto the test device, the Google Pixel 5, to use wirelessly and independently from Android Studio which simulated the use case of being able to monitor the environment STEVE was set in from a distance within the same network. And as optimized in the integration testing phase, the video feed was enabled on the mobile hotspot network, while the robot's movement was activated through a Python script coded in Thonny, an IDE natively built

into the Raspberry Pi 5's OS. With these two actions running, it was confirmed that the video stream and mobility system were able to run in parallel, independent from any human interaction, confirming STEVE's autonomous nature which was set out in the scope of the project.

This was further exemplified during the Capstone Showcase where the fully integrated system was on display for the judges, professors, and other fellow students. During the course of the 3 hours, participants were able to observe STEVE's mobility on the table in the room, while live streaming the camera feed from the robot. It has to be noted that STEVE's movement was constricted to the table and the length of the cable used to connect the Raspberry Pi 5 to the outlet, as a battery system was not yet implemented onto the robot, which is something that can be done for future iterations of STEVE which would be necessary for real-world usage.

The 3 hours in which STEVE had to stay powered on also demonstrated the robot's ability for endurance and durability, in which the video stream was able to remain connected to the remote monitoring mobile application throughout the entire period, without any drops in connectivity or errors demonstrating the reliability of the streaming connection. This also validated the choices that were made for the video streaming properties as the Raspberry Pi 5 did not overheat or shut down unexpectedly due to the CPU or GPU overloading.

With everything that was observed about STEVE during real-world testing and Capstone Showcase, it can be confidently said that STEVE successfully operated reliably without the use of human intervention, and makes an excellent stepping stone for future expansion of

this project with the stretch goals that were previously set out, such as autonomous navigation with obstacle navigation and the ability to detect intruders using the microphone array's voice detection and directional algorithms.

4.5 Comparison to Commercial Products

To compare STEVE to commercial solutions readily available on the market, the Knightscope K5 [4] was chosen due to its popularity on the market and similarities as a larger scale model to STEVE. This comparison will assess STEVEs current approach to autonomous security and see how it stacks up against this commercial approach to aid in identifying areas needed for potential future development.

To dive into more detail, the Knightscope K5 is a fully autonomous AI-driven security robot meant specifically for operation in businesses, campuses, and public spaces. It includes a wide range of features such as 360° HD video capture capabilities, two way audio communication, autonomous navigation within the environment, AI-driven data analytics through the application, and the ability to run 24/7 outdoors in any weather condition. The big caveat is the high operating costs in which Knightscope uses a subscription model which can cost an estimated $60,000 USD per year which is inaccessible to smaller businesses in need of an autonomous security solution.

Figure 16. Features of the Knightscope K5 Autonomous Security Robot.

In contrast, STEVE was developed with affordability in mind, being a small-scale solution of the Knightscope K5 with many of the same functionality. Not to mention, a big differentiator would be the environment the robot would operate in, as STEVE is meant to run in a controlled indoor environment covering smaller areas without the need for hiring security guards for patrolling, saving on hourly wages thanks to STEVE's low operating costs. Taking this case study into account, some great features from the Knightscope K5 that can be developed for future iterations of STEVE include the ability to autonomously path map and navigate using the LiDAR sensor that is already setup in the ROS 2 environment, the use of the microphone for live audio streaming and message broadcasting, and a more developed app interface to include functions such as automatic

recording when intruders are detected, live notifications to the user for intruder alerts, and safety measures such as low battery warnings.

In conclusion, STEVE demonstrates how the commercial security robot like the Knightscope K5 can be replicated on a small scale with the same fundamental surveillance capabilities at a much more affordable price, without the need for human hires. There is a very bright future for STEVE where much of the foundation has already been laid for future improvements and iterations.

5. Results

A blueprint was created in November 2024 with the purpose of highlighting component, functional, and performance specifications of the proposed STEVE project. This section aims to discuss the similarities and differences between the expected project path and actual final output.

5.1 Blueprint Spec Comparison

This table was featured in the Blueprint Report submitted in November. It focuses on quantifiable hardware attributes of the robot and their expected values. In this case, it has been appended with the "Achieved Value" column to display the differences between expected and final values.

Table 2. Functional specifications from project blueprint.

	Specification	**Target value**	**Tolerance**	**Achieved Value**
1	Operating Voltage (based on raspberry pi)	5 V	5%	5 – 6 V
2	Motor Speed	300 RPM	15%	275 RPM
3	Object Detection Range	100 cm – 1000 cm		N/A
4	Wide Angle Camera View	100+ degrees		62 degrees
5	Over Night Battery Life	8+ hours		N/A
6	Raspberry Pi RAM	8 GB		8GB

A table was also created to provide a summary of the robot's software specifications including functional capabilities, what will be included in the user interface, and performance goals. It also highlights which aspects of the product are guaranteed "base" goals that will be met, and which are "stretch" goals, hoping to be met given the right time and resources.

Table 3. Functional performance based on blueprint metrics.

1	**Functional requirements**	**Goal Type**	**Specification Met?**
1.1	Object Detection & Collision Avoidance	Base	No
1.2	Wifi Based Video Streaming	Base	Yes
1.3	Sound Level Detection + Processing	Stretch	No
1.4	Sound Location Triangulation	Stretch	No
2	**Interface requirements**	**Goal Type**	
2.1	Viewable, Live Camera Feed	Base	Yes
2.2	Notification Alerts	Stretch	No
3	**Performance requirements**	**Goal Type**	
3.1	Low to Ultra Low Latency Video Streaming (0- 3 second delay)	Base	Yes
3.2	Semi Randomized Path Following for Surveying	Stretch	No

Of the four base goals aimed to be achieved with STEVE, three have been fully met. The robot can stream video via wifi to an external device with a relatively short delay. The one base goal that was not achieved in full was the object detection and collision avoidance. Instead, basic motion was hard coded to the robot.

Given the fact that one base goal was still incomplete, four unique stretch goals were not attempted or completed. With this said, it is worth noting that the stretch goals were never intended to be vital parts of the design, rather ideas that could improve upon a finished product in the case there was extra time for advanced features.

5. 2 Source of Discrepancies

The largest difference between the proposed design blueprint and final product was that of the object detection and collision avoidance features. This was a base goal that directly relied on the autonomous functionality of STEVE. Due to unpredicted incompatibility issues that surrounded various sourced components, the physical construction of STEVE

took longer than initially allotted. Once the robot was fully assembled and had basic camera functionality, the software implementation of motion began.

It was during this time that it was discovered that one of the four wheel motors was irreparably damaged, and proved unusable. Tests were conducted and despite originally showing electronic functionality while using a multimeter, the motor not longer accepted any input upon updated testing. It is possible that it was physically damaged during set up, or sent with faulty wiring. Due to the time it had taken to reach this point in construction, the shipping time needed to access a replacement motor would extend beyond deliverable dates.

As the robot utilized mecanum wheels, it was possible to hard code basic mobility into each of the remaining three motors and allow the fourth wheel to act as a passive wheel (with a fully disconnected motor), but this prevented any further development of autonomous capabilities.

Of the four incomplete stretch goals, three relied on this autonomous capability - semi randomized path following could not be implemented at all because of the lack of this. Sound level detection and location triangulation could still be implemented to some degree, but could not be utilized as planned to command the robot's movement based on the results.

Though feasible additions to the app software, the live stream notifications were delayed due to a lack of support for audio and video anomaly detection.

5. 3 Final Robot Performance

Despite the absence of a base goal, the robot performed to a satisfactory degree. STEVE was able to move on a pre-planned path developed in Python, utilizing the three working motors. Simultaneously, the camera streamed live video from the connected camera to an external device. While sound level detection and location triangulation were not implemented in the final robot, the microphone array was tested and set up in the ROS environment – proving the possibility if there were functional mobility.

6. Conclusion

6.1 Further Development

More research and development could be done in various areas of this project. Expanding first on the speed and power efficiency would improve the mobility of the robot. Including a built-in, rechargeable power source to the bot would allow longer, untethered routes.

If the vigilance engine were chosen as a capstone project in future years, building on the existing progress would allow greater focus on automated driving and path planning. Different approaches to the physical design might be necessary to facilitate additional power to supply the R-Pi for long enough to travel and for exact navigation using the attached LiDAR sensor.

6.2 Potential Market

The primary modification to this design to make it suitable for commercial markets would be to change the main board. While internet connection is required for streaming functionality, many of the features of the raspberry-Pi went unused during this project. To reduce power consumption and remove unnecessary features such as Bluetooth and extra ports, the R-Pi could easily be replaced with a dedicated PCB by integrating a Wi-Fi module and ports capable of connection with the microphone, LiDAR, and wheels as peripherals. Similar practical functionality could be achieved by instead using dedicated, customized circuits on machined PCBs to replace the modifiable but easily damaged breadboard wiring. Combining the breadboard and main chip into one dedicated PCB would be much safer for clients and manufacturers.

A cost breakdown for these components is outlined below in Table 2, assuming the dimensions of the bot do not change and it is being prepared for small-batch production.

Table 4. Breakdown of replacement costs for commercialization of the project result.

Feature	Cost ($)	Replacement	Cost ($)	Cost Reasoning
R-Pi	130	PCB [3]	~20 ea. +50-500 upfront	Complex PCB design with high upfront costs, requires Wi-Fi and USB.
Breadboard	15	PCB	~10 ea. +50-75 upfront	Less complex than R-Pi, can be single-layer and
Portable Charger Block	90	Internal Power Supply (Rechargeable) [5]	10-30	LiPo battery: high capacity, rechargeable

Considering the current market for home surveillance devices it is likely that a market exists for a security bot like this one. Performance would need to be on par with successful competitors such as Ring, Blink, and Google smart security cameras to be viable in a growing market, offering significant advantages over other household names. With further polishing of external aesthetic choices and internal functionality and reliability

There are two potential directions for abuse of this product: one is the manipulation of security and information systems by 'hacking' to gain access to the livestream, and the other is the misuse of the bot to access forbidden or private locations. Both put individual privacy and security at risk by exposing potentially sensitive information and footage.

Mitigating these adverse impacts would require highly secure 'black box' programming and streaming to prevent unwanted access. For general security, eliminating the customizable features of the R-Pi by hardcoding bot functions would prevent the robot from tampering or adjusting core livestreaming or navigational features. Centralizing the livestream

hosting on an external and secure server and fully encrypting the signal would prevent illicit access to the video stream. Misuse of the bot is difficult to prevent outright, but there is potential to implement a tracking system into each bot and have these locations known centrally, allowing suspicious behavior to be reported or monitored by a centralized security team.

6.3 Team Dynamics

As requested in this report, a table reporting the overall effort expended by each team member on all aspects of the project is shown below.

Table 5. Group effort.

Group Member	**Overall Effort (%)**
Amy Li	100%
Cassidy Samoyloff	100%
Fenson Lin	100%
Kay Burnham	100%

Overall, the team managed to collaborate well through the course of the project, working together to ensure that deadlines were met. Opportunities for growth would see better communication between group members and supporting faculty members, as well as setting realistic expectations for deliverables from the project's inception.

6.4 Final Remarks

Major technical experience was gained in the process of programming live video for the bot and in troubleshooting the functionality of the mecanum wheels. Working extensively with the R-Pi native OS to manage onboard functionality and connectivity was a large component of the challenge faced.

Configuring the new R-Pi model 5 for external cameras was only possible using an open-source repository, which provided new experiences for the team. Learning to properly access and integrate the repository with local code, then managing troubleshooting issues related to the specific components used, are skills that will be useful in robotics work.

Overall, undertaking the project to complete a vehicle capable of livestreaming data while travelling along a specified path was a worthwhile and insightful venture into the world of security. Useful skills in both the technical and managerial aspects of the project were learned and will continue to complement the team's existing expertise.

7. References

[1] K. H. Rahouma, A. Ragab, and Y. B. Hassan, "Design and Implementation of a Surveillance Smart Home Robot," unpublished. [Online]. Available: https://www.researchgate.net/publication/365233781_Design_and_Implementation_of_a_Surveillance_Smart_Home_RobotResearchGate

[2] H. Meddeb, Z. Abdellaoui, and F. Houaidi, "Development of surveillance robot based on face recognition using Raspberry-PI and IOT," *Microprocessors and Microsystems*, vol. 96, Feb. 2023, Art. no. 104728. [Online]. Available: https://doi.org/10.1016/j.micpro.2022.104728

[3] King Sun PCB, "Average PCB Manufacturing Cost (2020-2025): A Complete Breakdown," March 19, 2025. [Online]. Available: https://www.kingsunpcb.com/average-pcb-manufacturing-cost-2020-2025/.

[4] Knightscope, "K5 - Autonomous Security Robot," Knightscope, [Online]. Available: https://knightscope.com/products/k5. [Accessed: Apr. 6, 2025].

[5] RobotShop, "Lithium Polymer Batteries (LiPo)," [Online]. Available: https://www.robotshop.com/collections/lithium-polymer-battery-packs.