# Smart Tripod

By:

Henry Thomas (henryjt3)

Kadin Shaheen (kadinas2)

Miguel Domingo (jdomi8)

Final Report for ECE 445, Senior Design, Spring 2025

TA: Chi Zhang

Professor: Viktor Gruev

May 2025

Project No. 29

## Abstract

This report summarizes the design and evaluation of a Smart Tripod system that enables remote, real-time control of a phone camera's orientation. The system consists of a motorized tripod, a custom iOS application, and a handheld controller that displays the iPhone’s screen via AirPlay. Users can adjust the camera angle dynamically while monitoring the view, enabling more flexible photo and video capture. The report details the integration of motor control, wireless communication, and a basic tracking algorithm. Verification of key functional requirements, including stable pan/tilt control, reliable network communication between components, and real-time video streaming are included in the report. Issues and limitations related to tracking precision and latency are discussed, along with potential improvements.

# Contents

# 1. Introduction

## 1.1 Problem and Solution

Traditional tripods provide the necessary stability for cameras and smartphones but fall short in offering dynamic adjustability and real-time framing assistance. Users must manually adjust the tripod's position and angle—often through trial and error—making the process time-consuming and cumbersome, particularly for solo content creators, vloggers, or group shots requiring precise composition. Moreover, conventional tripods are static and do not adapt to subject movement, limiting their practicality in scenarios such as demonstrations, action shots, or personal video recordings.

Although motorized tripod systems exist, their capabilities are typically limited to basic pan-and-tilt functions, often without any form of real-time visual feedback. Many lack integrated solutions for live view monitoring, relying instead on external adjustments made without immediate confirmation. Additionally, these systems rarely include subject-tracking functionality, requiring users to manually maintain framing—an inefficient and impractical approach in dynamic filming conditions.

To address these limitations, we propose the development of a Smart Tripod that integrates motorized adjustment, wireless remote control, real-time preview, and automatic subject tracking into a single, user-friendly platform. Our system allows users to control the orientation of their smartphone camera remotely via an external control unit equipped with a live display that mirrors the phone's screen. This enables precise framing without physical repositioning. The remote also offers integrated controls for zoom, photo capture, and video recording, all supported through a custom smartphone application.

For dynamic shooting scenarios, the Smart Tripod features an intelligent tracking mode that automatically adjusts the camera's orientation to keep the subject centered within the frame. This functionality ensures smooth, professional-grade footage without requiring manual intervention. By combining motorized mobility, live feedback, intuitive app integration, and subject-tracking capabilities, the Smart Tripod redefines convenience and precision in mobile photography and videography.

## 1.2 Visual Aid

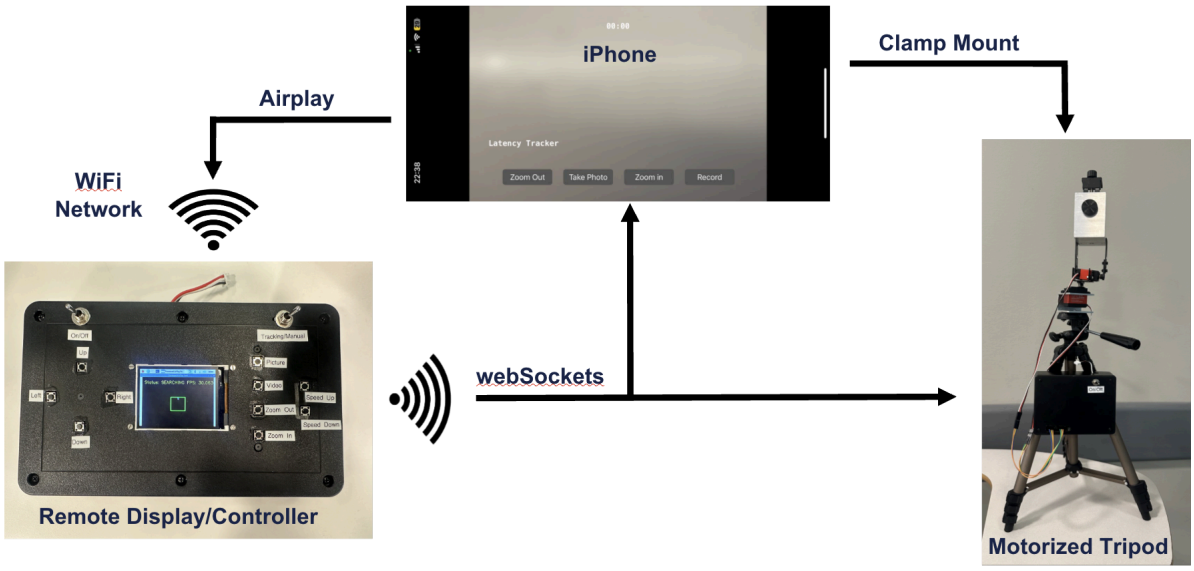


***Figure 1: Visual Aid***

Above in Figure 1 is a visual of the full system. Included is the Remote Display/Controller, the iPhone and custom IOS application, as well as the Remote Tripod.

## 1.3 High Level Requirements

### Motor Accuracy and Responsiveness

The most critical high-level requirement of our project is accurate motor response. The design must reliably position the phone according to specified azimuth and zenith angle inputs, with precise adjustments to ensure smooth framing and effective subject tracking. Achieving this will require careful consideration of signal transmission, microcontroller processing, motor inertia, and the phone's inertia.

### iPhone camera action and responsiveness

The second high-level requirement is to enable real-time remote camera operation over WiFi. This functionality will allow users to capture photos, record videos, and adjust zoom with minimal delay. Camera actions triggered through the remote interface must be executed promptly to ensure a responsive user experience. Verification of this requirement must take into account factors such as network congestion, WebSocket transmission latency, and iPhone processing time.

### Airplay and tracking

The final high-level requirement is low-latency, high-frame-rate live streaming from the iPhone to an external display via AirPlay. This enables accurate visual feedback for the user and allows the system to perform automated subject tracking. For tracking, the system should detect and follow the user using OpenCV, sending corrective commands to the motors to keep the user centered in the frame. This

requirement must address challenges such as AirPlay transmission delays, Raspberry Pi processing limitations, and the computational overhead of real-time OpenCV operations.

The Smart Tripod must provide low-latency, high-frame-rate live streaming from the iPhone to the external display via AirPlay. The video feed should maintain at least 24 FPS with <1 second of latency while streaming through a Raspberry Pi. Additionally, the system must perform subject detection via OpenCV and send corrective tracking commands to the motors within 500 milliseconds of receiving the video feed. Challenges include AirPlay transmission delays, Raspberry Pi processing constraints, and real-time OpenCV computation overhead. We will assess system performance by logging frame rates, transmission latencies, and processing times using code-based timestamp logging.

# 2 Design

Our design consists of three distinct parts that will be discussed in this section. Included is the Remote Display/Controller System, Motorized Tripod System, and iPhone App Integration System. As shown below in Figure 2, each system has a specific purpose and works cohesively with each other to ensure a functional and complete design.

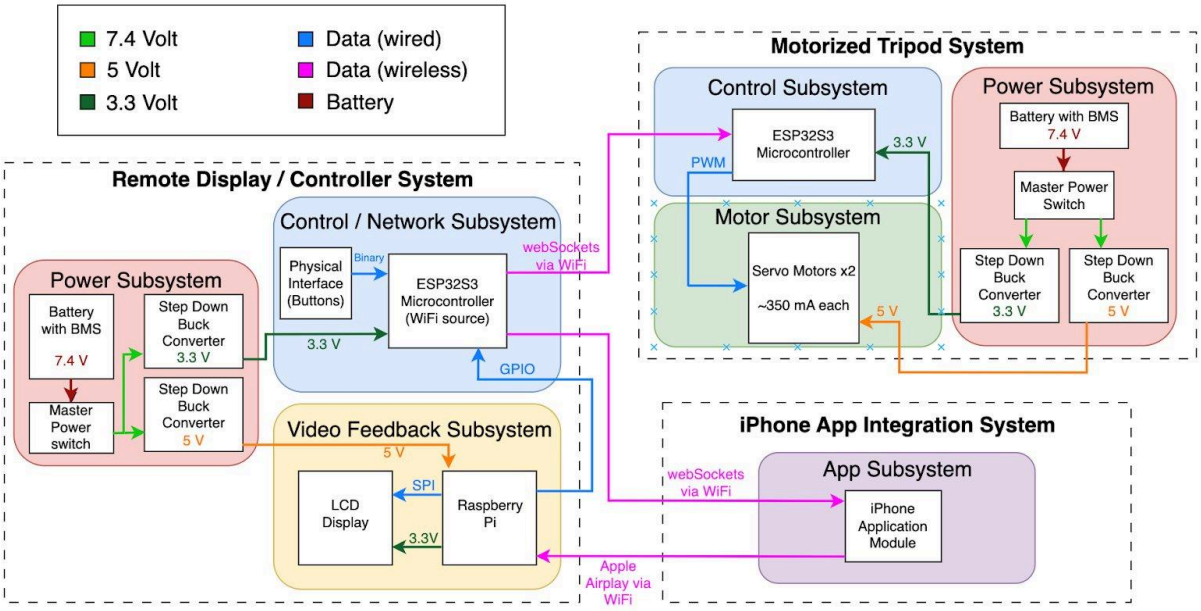


***Figure 2: Block Diagram***

## 2.1 Remote Display/Controller

The Remote Display/Controller serves as the central hub for system coordination and control. It establishes and maintains the Wi-Fi network that connects all devices, hosts the WebSocket server to enable real-time communication between components, and launches an AirPlay server to stream the iPhone's display directly to the physical controller. In addition, it processes the incoming video feed using

a tracking algorithm to support automatic subject following, and interfaces with physical buttons to trigger camera functions and motorized adjustments. This system includes the following subsystems:

### 2.1.1 Control/Network Subsystem

The physical components of this system include the ESP32-S3 microcontroller, along with all associated GPIO connections. These encompass tracking signal inputs from the Raspberry Pi, as well as button inputs used to control motor movement and camera functions. The complete circuit schematics for this subsystem are shown below in Figure 3.

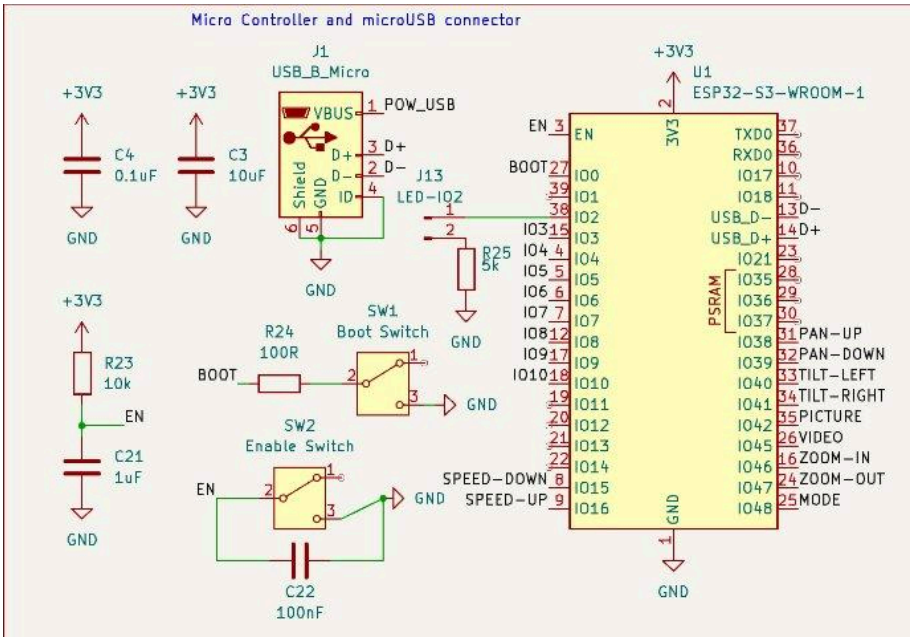


***Figure 3: Control Circuit Schematic for Remote Display/Controller PCB***

Figure 3 shows the complete microcontroller schematic, excluding the GPIO connections. This schematic includes the ESP32-S3 microcontroller, two decoupling capacitors, EN (enable) and BOOT switches, and a micro-USB header. The micro-USB header is directly connected to the D+ and D– pins of the microcontroller and is specifically intended for flashing firmware. Additionally, the micro-USB connection will only supply power to the microcontroller when the PCB is in flash mode, which is controlled by a dedicated switch in the power circuit.

For each of the user controlled buttons(**up, down, left, right, speed up, speed down, picture, video, zoom in, zoom out**), it is necessary to debounce and filter out any unintended input noise. For example, a button that initiates a photograph must be debounced to prevent accidentally capturing multiple photos with a single press. For this circuit, the design shown below in figure 4 was used. A button press in this circuit is defined as **active low** (0V when pressed).

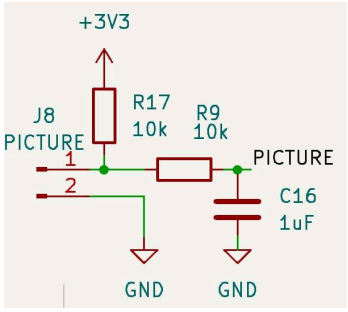


***Figure 4: Button Debouncing Circuit Schematic***

We can calculate the RC time constant as follows:

$$\tau_{discharge} = R_{discharge}C = 10kOhm * 1\mu F = 10ms$$
$$\tau_{charge} = R_{charge}C = 20kOhm * 1\mu F = 20ms$$

With time constants within 10ms and 20ms, the circuit will provide a reasonable debounce time to filter out noise and not create a large amount of delay.

This subsystem also includes the ESP32-S3 firmware, which is responsible for managing core communication and control functions. Its capabilities include establishing the Wi-Fi network, hosting the WebSocket server, maintaining active GPIO connections, and transmitting WebSocket commands based on inputs from both the user interface and the Raspberry Pi. For further reference, see Appendix B for a link to the complete code repository.

### 2.1.2 Video Feedback Subsystem

The video feedback subsystem consists of a Raspberry Pi 4B paired with a Waveshare 2.4-inch LCD SPI Display. This subsystem functions as an AirPlay server, enabling the iPhone to wirelessly connect and mirror its screen, which displays the custom iOS application. The mirrored video feed is shown on the display and processed using OpenCV to detect and track faces in real time. Based on the position of the detected face, the Raspberry Pi sets specific GPIO outputs to trigger the corresponding motor movements needed to re-center the subject within the frame. A flowchart illustrating the tracking algorithm is provided below in Figure 5.

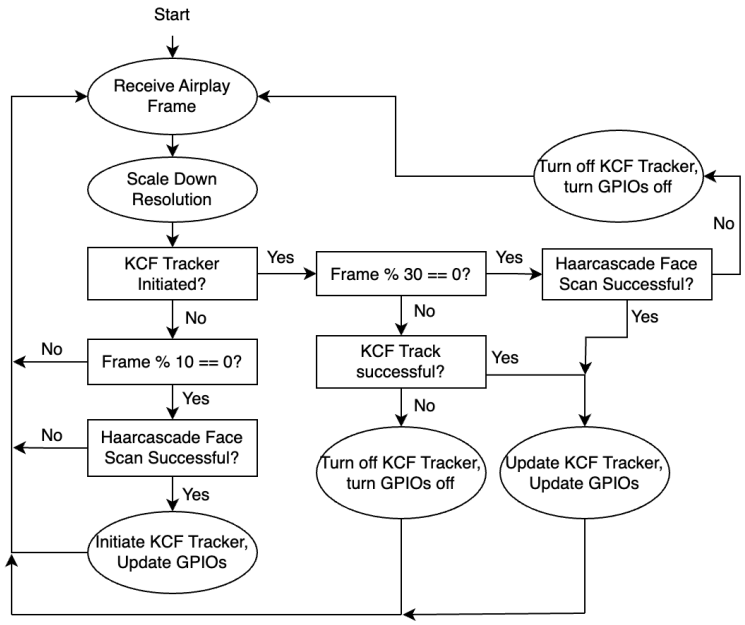


***Figure 5: OpenCV Tracking Algorithm***

The tracking algorithm begins by capturing and downscaling each frame to reduce processing time. If the KCF tracker is not yet initialized, the system performs face detection using the HaarCascade classifier once every ten frames. Upon detecting a face, it switches to KCF tracking, which updates the target region every frame. To maintain accuracy, HaarCascade detection runs once every 30 frames to verify the face remains within the tracked area. HaarCascade was chosen for its simplicity and speed, while KCF offers efficient region tracking with significantly lower computational load. Given the hardware constraints, the system prioritizes speed and low latency over absolute accuracy. Alternative tracking algorithms(body detection, advanced motion detection) were found to be too complicated and required much higher computational overhead causing them to be less effective. For further reference, see Appendix B for a link to the complete code repository.

### 2.1.3 Power Subsystem

The power subsystem consists of a battery with an internal BMS, a 7.4V to 5V step-down converter, a 7.4V to 3.3V step-down converter, and a flash switch. It supplies power to the main board and its peripherals, including but not limited to the onboard ESP32-S3, Raspberry Pi, and button interfaces. The power system used in the Remote Display/Controller System is identical to that of the Motorized Tripod

System. The schematic for the final power system is shown below.

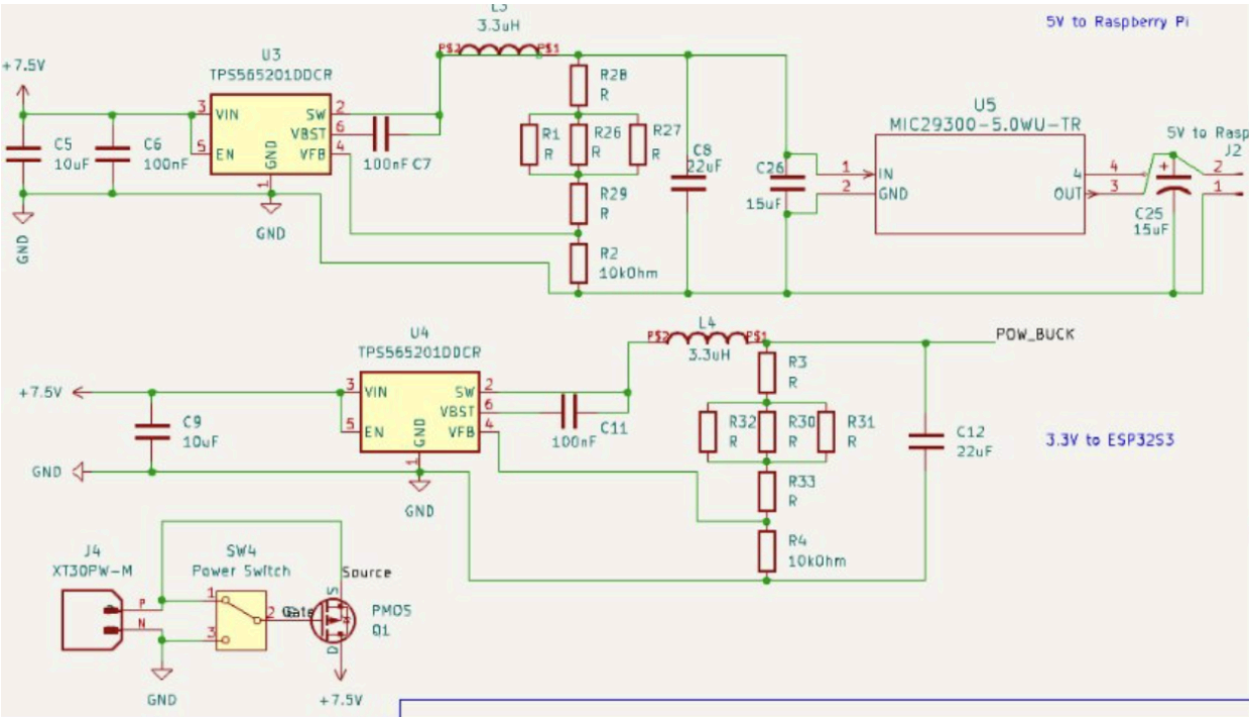


***Figure 6: Power systems, including battery power switch, 3.3 V converter, and 5 V converter.***

As shown in Figure 6, the 3.3V system includes a step-down conversion from 7.4V to 4.7V using a 2A TPS562201DDCR buck converter. Similarly, the 5V buck converter steps down from 7.4V to 5.6V using a 5A TPS565201DDCR. Both ICs utilize 3.3 µH inductors, which are at the upper end of the recommended range, chosen to help reduce switching noise at low power loads. Output capacitors were also added to suppress high-frequency noise.

The 3.3V buck converter output is set using a resistor divider network (R28–R29). Additional resistor pads were included to fine-tune the output voltage for optimal system efficiency. The output voltage of the buck converter is calculated using the following equation:

$$Vout = 0.768 * (1 + \frac{R1}{R2}) \quad [5]$$

In this design, R1 is the resistor network (R28–R29), with selected values of R2 = 10 kΩ and R1 ≈ 51.5 kΩ.

The 5V buck converter follows a similar design approach, as shown in Figure 6. Its output voltage is calculated with a slightly different equation:

$$Vout = 0.760 * (1 + \frac{R1}{R2}) \quad [6]$$

For this configuration, R2 = 10 kΩ and R1 ≈ 63 kΩ.

Initially, the power system included only these two buck converters. However, during testing, a persistent noise margin of approximately 200 mV peak-to-peak was observed, making it difficult to stay within the noise limits specified in the (R and V table insert). To address this, the output voltages of the buck converters were increased, and two low-dropout (LDO) regulators were added: the AZ1117-3.3 and the MIC29300-5.0WU-TR. The performance of these LDOs will be discussed later in Section _____.

The 3.3V linear regulator is positioned at the output of the flash switch, as shown in Figure 7. This setup allows selection between power input from the battery or from the micro-USB, enabling the board to be flashed while isolating the 5V power planes. Schottky diodes in reverse polarity were considered for this purpose; however, their inclusion caused the board voltage to drop below 5V, which was insufficient to reliably power both the Raspberry Pi and the servo motors.

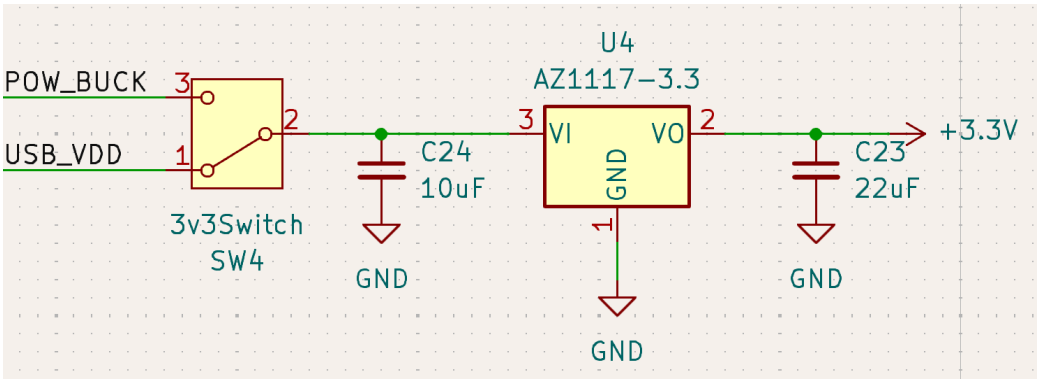


***Figure 7: Power flash switch and the AZ1117-3.3 LDO.***

## 2.2 Motorized Tripod

### 2.2.1 Control Subsystem

The control subsystem of the motorized tripod is responsible for connecting to the Wi-Fi network, interfacing with the WebSocket server, and driving the servo motors via PWM signals. Its microcontroller circuit is identical to that of the Remote Display/Controller shown in Figure 3, with the exception that GPIO pins 4 and 5 are configured as PWM outputs instead of button inputs. The firmware enables the microcontroller to establish network and WebSocket connections, interpret motor control commands, and maintain stable PWM outputs for precise servo operation. For further reference, see Appendix B for a link to the complete code repository.

### 2.2.2 Motor Subsystem

The motor subsystem utilizes two 25KG high-torque RC servo motors mounted to the iPhone clamp, providing pan and tilt rotation up to 270 degrees. To support both manual and tracking modes, the system is designed to achieve directional adjustments within ±2 degrees of accuracy. Each motor has

three connections: a data pin linked directly to a designated GPIO on the ESP32-S3, a ground pin tied to the PCB's ground plane, and a power pin connected to the 5V output of the power circuit. Originally, our design planned to use stepper motors, however these required higher voltages(12V) and were less optimal for our motor mount setup which ultimately caused our team to make the design change.

### 2.2.3 Power Subsystem

The power subsystem of the Motorized tripod works is designed to the same specifications as the Remote Display/Controller System power subsystem. This system provides power to the onboard ESP32-S3, its button peripherals, and . The exact specifications of the design can be found in Section 2.1.3.

## 2.3 iPhone Application

The iPhone Application is directly integrated into the rest of the hardware using the webSocket server established by the Control/Network Subsystem. The intention for this integrated application is to allow users to be able to use their own IOS device to be used as the camera when utilizing the smart tripod. The seamless connection allows for bi-directional communication between the phone and the rest of the tripod. This application can be made available to users by downloading on their own IOS devices. The iPhone Application uses webSockets to receive commands directly from the ESP32-S3 over WiFi as well as providing real-time feedback via AirPlay to the Raspberry Pi which can then be viewed on the external display.

### 2.3.1 App Subsystem

The application itself was developed through the use of Apple's very own integrated development environment, xCode. With the use of the AVFoundation framework, along with establishing a webSocket connection through the Smart Tripod's own dedicated network, the App emulates the Camera App by allowing the user to take pictures and record videos with the external remote created for the Smart Tripod. The App subsystem listens for any webSocket commands sent as control buttons are pushed which then interprets the string message to execute the respective commands on the iPhone. The AVFoundation framework allows for inspecting, playing, capturing, recording, and processing audiovisual media on IOS devices. This framework was utilized in allowing the app, with necessary permissions by the user, to use the device's camera for real-time feedback as well as necessary camera controls.

# 3. Design Verification

The following section will describe the high level requirement verification and will delve into proof of verification. Some of these high level requirements require specified lower requirements that are specified in our R and V table found in Appendix A

## 3.1 Motor Accuracy and Responsiveness

To ensure motor accuracy and responsiveness, the motors were designed to respond within 250 ms and achieve an accuracy of 2°. Motor accuracy was verified across various step sizes using the iPhone's built-in Compass and gyroscope apps, which demonstrated sub-1° step sizes for multiple small adjustments. Responsiveness was measured using internal WebSocket commands, resulting in a total

delay from command issuance to signal transmission of 10–20 ms, as shown in Figures 8 and 9.

```
I (43334) wifi_ws: Button press @Timestamp: 42996830 us
I (43334) wifi_ws: Sent 'start-up' to 1 clients
I (43334) wifi_ws: Websocket sent at @Timestamp: 43000403 us
I (43334) wifi_ws: Time elapsed: 3 ms
I (43644) wifi_ws: Sent 'stop-up' to 1 clients
```

***Figure 8: Delay from Button to webSocket command sent ~ 2-4ms***

```
I (108606) wifi_ws_client: Received command: start-left
I (108616) CommandProcessor: Processing command: start-left
I (108616) CommandProcessor: Recieved command @Timestamp: 108270036 us
I (108616) ServoControl: Servo 1 position updated to duty: 1505
I (108626) CommandProcessor: Elapsed time: 6 ms
```

***Figure 9: Delay from command received to motor adjustment ~ 5-8ms***

## 3.2 iPhone Camera Action and Responsiveness

In order to ensure that the camera actions work as intended without any issues that may stall other parts of the system, we decided from our tolerance analysis that the camera actions triggered from the external remote interface must execute within 500ms. This ensures that the user experience is not disrupted by delay's that may occur. The anticipated sources of delay include network congestion, and the iPhone application processing time. These delays were both captured through logging timestamps and finding the difference between when the button is pushed to when the webSocket command is sent as well as when the webSocket is received by the application to when it begins execution of the command. Total delay is well within the requirements and shows almost no delay when executing commands to the iPhone application.

```
I (43334) wifi_ws: Button press @Timestamp: 42996830 us
I (43334) wifi_ws: Sent 'start-up' to 1 clients
I (43334) wifi_ws: Websocket sent at @Timestamp: 43000403 us
I (43334) wifi_ws: Time elapsed: 3 ms
I (43644) wifi_ws: Sent 'stop-up' to 1 clients
```

***Figure 10: Delay from Button to webSocket command sent (~ 2-4ms)***

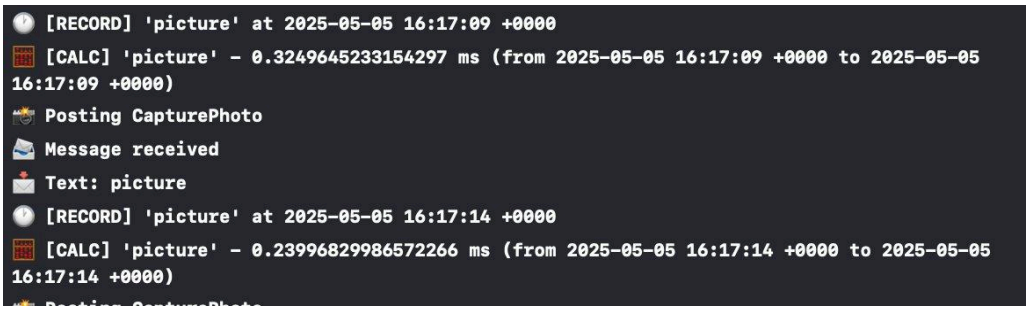


***Figure 11: Delay from webSocket command sent to execution (~ 0.2-0.4ms)***

## 3.3 AirPlay and Tracking

To ensure a reliable and user-friendly experience, the AirPlay connection and automatic tracking were required to maintain a latency of under one second and a minimum frame rate of 24 FPS at a resolution of 320×240—the native resolution of the LCD screen. These performance criteria were validated according to the Requirements and Verification (R&V) table outlined in Appendix A.

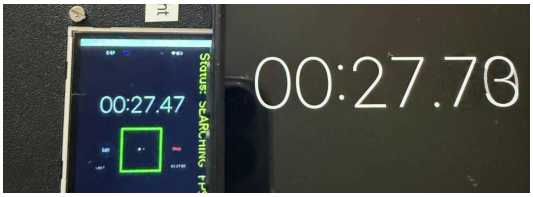


***Figure 12: Airplay Latency (~200-600ms)***

Figure 12 illustrates the timer test used to evaluate AirPlay latency. As shown, the system consistently maintained latency well below one second during normal operation.

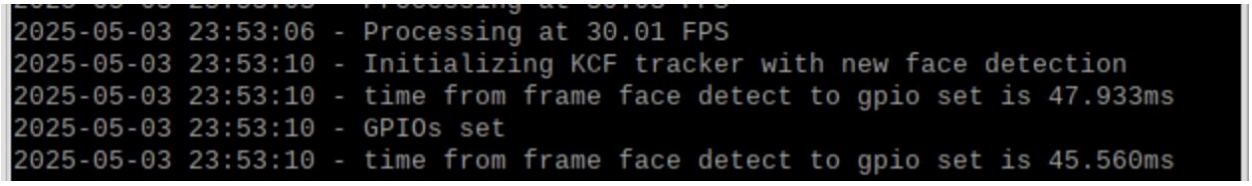


***Figure 13: FPS and Tracking Latency(~28-30FPS, ~45-60ms respectively)***

Figure 13 presents the recorded frame rate and tracking latency resulting from the tracking algorithm. Timestamps were used to calculate both metrics. The results met all specified requirements, confirming the effectiveness of the AirPlay connection and tracking system.

## 3.4 Power Systems

The power system was verified against the requirements outlined in the R and V table in Appendix A. For detailed specifications, please refer to the R and V table. Our testing results remained within the defined tolerance limits, as shown in Figures 14 and 15 below. This established a reliable power distribution network for all onboard components, ensuring compliance with the high-level system requirements.

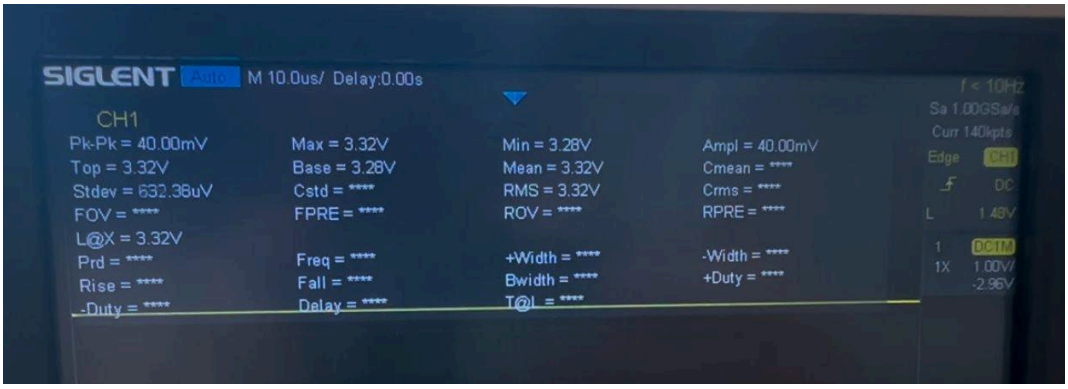


***Figure 14: Output of the 3.3V LDO with 3.32V mean and 40mV peak-peak***

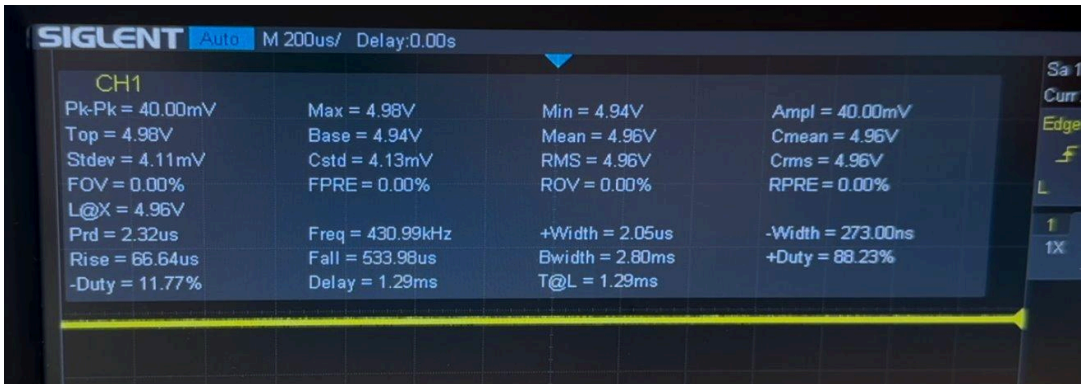


***Figure 15: Output of the 5V LDO with 4.96V mean and 40mV peak-peak.***

## 4. Costs

Our project consists of the development of multiple PCBs, and the development of an IOS application. In addition, each PCB will need a dedicated power distribution system, and additional peripheral control including servo motors, and a Raspberry Pi. Due to the extensive nature of our project, we estimate that each team member will contribute 12 hours per week to complete this project.

The average salary of a UIUC EE graduate is approximately $88,000 and the salary of a UIUC CE graduate is approximately $109,000 which results in an average salary of $98,500. Assuming working 40 hours per week with two weeks vacation, this results in: $98,500/2000 hours = $49.25/hour

Since our group has three members, and this project will take roughly 8 weeks to complete, the total labor cost can be calculated:

$$Labor\ Cost\ =\ (hourly\ rate)(total\ hours\ spent)\ *\ 2.5\ per\ partner$$

$$Labor\ Cost\ =\ (49.25\ \$/hour)(12\ hours/week\ *\ 8\ weeks)\ *\ 2.5\ *\ 3\ partners\ =\ \$35460$$

In addition to our group's labor cost, we have each part and their costs shown below in table 1:

**Table 1 Parts Costs**

| Part | Quantity | Retail Cost ($) | Bulk Purchase Cost ($) | Manufacturer |
|---|---|---|---|---|
| 25KG High Torque RC Servo | 2 | 16.99 | 33.98 | amazon |
| ESP32-S3-WROOM-1-N16 | 2 | 5.92 | 11.84 | digikey |
| Raspberry Pi 4B | 1 | 55 | 55 | digikey |
| Smartphone clamp mount | 1 | 6.99 | 6.99 | amazon |
| BMS for battery | 1 | 8.99 | 8.99 | amazon |
| Tripod | 1 | 15.39 | 15.39 | amazon |
| PCEONAMP 7.4V Battery | 2 | 19.99 | 39.98 | amazon |
| Waveshare 2.4in LCD SPI Screen | 1 | 18.99 | 18.99 | amazon |
| Pan and Tilt Servo Bracket Mount | 1 | 9.95 | 9.95 | amazon |
| PCB | 2 | 5 | 10 | pcbway |
| AZ1117-3.3 | 2 | 1.78 | 3.56 | digikey |
| TPS565201DDRC | 2 | 1.22 | 2.44 | digikey |
| TPS562201DDRC | 2 | 0.35 | 0.7 | digikey |
| Amphenol Micro USB B Connector | 2 | 0.47 | 0.94 | digikey |
| Push Buttons | 8 | 0.1 | 0.8 | digikey |
| Switches | 9 | 0.77 | 6.93 | digikey |
| XT30 PCB Connector | 2 | 0.81 | 1.62 | tme |
| 3.3uH Inductor - IHLP3232DZER3R3M | 4 | 1.1 | 4.4 | digikey |

| | | | | |
|---|---|---|---|---|
| 01 | | | | |
| Green Clear SMD LED | 3 | 1.04 | 3.12 | digikey |
| Red Clear SMD LED | 1 | 1.04 | 1.04 | digikey |
| 0.1uF 0805 Capacitor | 10 | 0.08 | 0.8 | digikey |
| 1uF 0805 Capacitor | 11 | 0.08 | 0.88 | digikey |
| 10uF 0805 Capacitor | 9 | 0.15 | 1.35 | digikey |
| 22uF 0805 Capacitor | 3 | 0.12 | 0.36 | digikey |
| 15uF 1210 Capacitor | 2 | 0.23 | 0.46 | digikey |
| 56kOhm Resistor | 1 | 0.1 | 0.1 | digikey |
| 33kOhm Resistor | 2 | 0.1 | 0.2 | digikey |
| 10kOhm Resistor | 25 | 0.1 | 2.5 | digikey |
| 5kOhm Resistor | 1 | 0.12 | 0.12 | digikey |
| 100 Ohm Resistor | 4 | 0.1 | 0.4 | digikey |
| 60.4 kOhm Resistor | 1 | 0.1 | 0.1 | digikey |
| 60 Ohm Resistor | 1 | 0.1 | 0.1 | digikey |
| 140 Ohm Resistor | 1 | 0.1 | 0.1 | digikey |
| 260 Ohm Resistor | 1 | 0.1 | 0.1 | digikey |
| MIC29300-5.0WU | 2 | 3.55 | 7.10 | digikey |

The parts needed for the product sum to a total of $251.33.

In addition to these parts, the machine shop quoted our project at 2 days of work, at $56.12 per hour, giving a total machine shop cost of: (cost per hour)*(hours per day)*(days) = (56.12)*(7.5)*(2) = $841.8

## 4.3 Total Cost

The list above along with the cost analysis for labor concludes our total cost for the entire project from top to bottom. This grand total which includes the parts list and labor cost from all group members as well as with help from the machine shop will be:

$$Total\ Cost\ = \$35460\ +\ \$841.80\ +\ \$248.70\ =\ \$36550.50$$

# 5. Conclusion

## 5.1 Accomplishments

Overall, our design proved to be highly successful. During the planning phase, we were diligent in selecting components and cross-referencing multiple sources to ensure well-informed decisions. Recognizing the importance of stable power delivery, we prioritized designing a reliable and efficient power system capable of handling the variable demands of the motors and Raspberry Pi. This process deepened our understanding of power circuit design for high-current applications.

Firmware flashing was accomplished using a streamlined approach by leveraging the ESP32-S3's built-in USB-to-UART converter. This decision eliminated the need for a custom programming header and enabled reliable, issue-free flashing via micro-USB.

All major software components performed effectively. The Wi-Fi network and WebSocket servers maintained stable connections, and the face-tracking algorithm achieved consistent and accurate results—an area we initially identified as a technical risk.

This project challenged us to apply critical thinking, work through complex design decisions, and ultimately build a unique, functional system while gaining valuable hands-on experience.

## 5.2 Uncertainties

While we met nearly all of the design specifications, there was one R and V requirement we were unable to satisfy: the network throughput capacity requirement outlined in Section 4 of the R and V table in Appendix A. Although many microcontrollers include internal network monitoring tools that can be evaluated using iPerf, the ESP32-S3 does not support running iPerf, as the relevant ESP-IDF commands are only compatible with earlier ESP32 models. As a result, we were unable to verify this specific requirement.

## 5.3 Ethical considerations

Our project raises several ethical concerns that can be categorized into four key areas: Privacy and Surveillance, Bias in Computer Vision Tracking, Transparency and Honesty, and Accessibility and Inclusivity.

### 5.3.1 Privacy and Surveillance

Privacy is a primary concern due to the Smart Tripod's use of object tracking, remote control functionality, and live camera feeds. These features inherently involve wireless communication, screen mirroring, and data transmission, which could pose risks if not properly managed. To align with IEEE Code of Ethics I.1, which emphasizes the responsible handling of user data and privacy protection, we will ensure that users are fully aware of what data is being collected at all times. Clear visual indicators will notify users when tracking or remote access features are active, preventing unauthorized or unnoticed usage.

### 5.3.2 Bias in Computer Vision Tracking

To ensure fair and unbiased tracking, the object detection system must be trained on diverse datasets to prevent any unintended biases that could cause the algorithm to favor or exclude certain individuals. This aligns with IEEE Code of Ethics II, which calls for fairness and respect in technology development. By testing the system with a wide range of skin tones, clothing variations, and lighting conditions, we will work to eliminate biases and ensure accurate tracking for all users.

### 5.3.3 Transparency and Honesty

Maintaining transparency and honesty throughout the development process is essential. Users must have confidence that the Smart Tripod operates as advertised and that its limitations and risks are communicated clearly. To adhere to IEEE Code of Ethics I.3, which emphasizes honesty in engineering practices, we will provide detailed documentation of the system's capabilities and limitations. This includes publishing clear user guidelines, performance expectations, and any potential risks associated with the product.

### 5.3.4 Accessibility and Inclusivity

The Smart Tripod should be intuitive and accessible to all users, regardless of technical expertise. In accordance with ACM Principle 1.4, which promotes fairness and inclusivity in technology, we will design the interface to be user-friendly, with clear controls and straightforward setup. Additionally, users will have full control over recording and tracking functions, ensuring that the system does not unintentionally capture footage or track subjects without explicit consent.

By addressing these ethical considerations proactively, we aim to develop a secure, fair, and transparent product that respects user privacy while enhancing the photography and videography experience.

## 5.4 Future work

### 5.4.1 Hardware

After working extensively with the power system and control interfaces, our group identified two key design improvements for future iterations.

First, we would prioritize redesigning the power system entirely. Our initial power requirements were significantly overestimated, as we followed the recommended power specifications outlined in the Raspberry Pi and ESP32-S3-WROOM1 documentation. Given the observed lower power consumption in practice, we would consider eliminating the 3.3V buck converter altogether. Currently, the system steps down to a regulated 5V, while the buck converter reduces the voltage to 4.7V before it is regulated down to 3.3V. The combined efficiency of the buck converter and linear regulator is likely comparable to that of a direct 5V-to-3.3V step-down. Based on the actual power demands—approximately 1W per board—and recognizing that the 5V system requirements were lower than anticipated, we would consider switching to a 3.7V single-cell battery paired with a 3.3V LDO and a 5V step-up converter. This approach would reduce the overall footprint and improve system efficiency.

The second proposed change is to increase the display size to enhance user experience. Budget permitting, we would also like to integrate a touchscreen interface to replace the physical buttons. This would not only enlarge the usable screen area but also boost the system's market appeal.

### 5.4.2 Firmware/Software

After gaining knowledge on xCode and its multitude of features, there are many ways to elevate and customize the application and add even more features that can tailor to an even larger majority of users who like to partake in media creation, personal or professional. Apple has several frameworks that can be implemented to have even more control over the device. One of these frameworks is called SensorKit, which allows developers to select raw data, or metrics that the device processes from a sensor. One of many sensors that can have a positive impact on the Smart Tripod system is the Ambient Light Sensor (ALS) which captures the intensity of light which can then be used to automatically adapt to the environment and provide the best lighting which would also work to improve the automatic tracking feature. Further work also includes creating an even more user-friendly UI along with even more indicators to ensure that users are always informed about what is happening in the app. A great example of implementing more features would be how certain apps emulate digital cameras, this would allow users to be able to edit their pictures in real-time instead of after capturing their videos and pictures.

## References


[1] AG, Infineon Technologies. “Battery Management Systems (BMS).” *Infineon Technologies*, www.infineon.com/cms/en/applications/solutions/battery-management-system/ . Accessed 21 Feb. 2025.

[2] *Amazon.Com: 25KG High Torque RC Servo, MIUZEI Waterproof Servo Motor Compatible with 1/6, 1/8, 1/10, 1/12 RC Car, Full Metal Gear Steering with 25T Horn 270 Degree (1Pcs) : Toys & Games*, www.amazon.com/Servo-Miuzei-Torque-Waterproof-Control/dp/B0C5LTNXDH . Accessed 7 Mar. 2025.

[3] *Amphenol Connectors | Cable Assemblies | Interconnects | Mobile, ...*, www.amphenol-cs.com/media/wysiwyg/files/documentation/datasheet/inputoutput/io_micro_usb_2_gmcb05.pdf . Accessed 7 Mar. 2025.

[4] *Buy A Raspberry Pi 4 Model B – Raspberry Pi*, www.raspberrypi.com/products/raspberry-pi-4-model-b/ . Accessed 22 Feb. 2025.

[5] “ESP32-S3-WROOM-1-N16.” *DigiKey Electronics*, www.digikey.com/en/products/detail/espressif-systems/ESP32-S3-WROOM-1-N16/16163979?gclsrc=aw.ds&&utm_adgroup=&utm_source=google&utm_medium=cpc&utm_campaign=PMax+Shopping_Product_Medium+ROAS+Categories&utm_term=&utm_content=&utm_id=go_cmp-20223376311_adg-_ad-__dev-c_ext-_prd-16163979_sig-CjwKCAiArKW-BhAzEiwAZhWsIHmmfeULgcV3QKfGFJ3WSrSNnp6vaR6E52gP28VsN1jJly0nU1UN9hoCJDMQAvD_BwE&gad_source=1&gclid=CjwKCAiArKW-BhAzEiwAZhWsIHmmfeULgcV3QKfGFJ3WSrSNnp6vaR6E52gP28VsN1jJly0nU1UN9hoCJDMQAvD_BwE . Accessed 22 Feb. 2025.

[6] “TPS565201 4.5-V to 17-V Input, 5-A Synchronous Step-Down Voltage Regulator.” *Ti.Com*, www.ti.com/lit/ds/symlink/tps565201.pdf?HQS=dis-dk-null-digikey mode-dsf-pf-null-wwents=1741257815034&ref_url=https%253A%252F%252Fwww.ti.com%252Fgeneral%252Fdocs%252Fsuppproductinfo.tsp%253FdistId%253D10%2526gotoUrl%253Dhttps%253A%252F%252Fwww.ti.com%252Flit%252Fgpn%252Ftps565201 . Accessed 24 Feb. 2025.

[7] *Waveshare*, www.waveshare.com/2.4inch-lcd-module.html. Accessed 18 Feb. 2025.

[8] Apple Inc. (n.d.). *AVFoundation - Apple Developer*. Apple Developer. https://developer.apple.com/av-foundation/.

[9] “*PMEG3050EP 30V 5A low Vf schottky diode*. (n.d.). Nexperia BV. Retrieved March 6, 2025, from https://assets.nexperia.com/documents/data-sheet/PMEG3050EP.pdf

[10] *ACM Code of Ethics*. (n.d.). ACM. Retrieved March 6, 2025, from https://www.acm.org/code-of-ethics

[11] *IEEE Code of Ethics*. (n.d.). IEEE. Retrieved March 6, 2025, from https://www.ieee.org/about/corporate/governance/p7-8.html

[12] "TPS562201 4.5-V to 17-V Input, 2-A Synchronous Step-Down Voltage Regulator." *Ti.Com*, https://www.ti.com/general/docs/suppproductinfo.tsp?distId=10&gotoUrl=https%3A%2F%2Fwww.ti.com%2Flit%2Fgpn%2Ftps562201. Accessed 7 May. 2025.

## Appendix A: Requirement and Verification Table

**Table X System Requirements and Verifications**

| Requirement | Verification | Verification status (Y or N) |
|---|---|---|
| 1. 3.3V Power supply - Controller system and motor subsystem<br><br>The power subsystem should be capable of supplying power to the control system at 3.3 V $\pm$ 100mV from the step down converter to ensure safe operation of the ESP32S3 and its peripherals. It should be able to supply this current and voltage irrespective of voltage drops from the battery pack. | 1. Verification<br><br>a. Plug in the battery to the PCB.<br>b. Connect an oscilloscope channel pair across the output voltage of the buck converters inductor, and the ground plane.<br>c. Measure the average voltage as well as the peak to peak ripple. Compare these values to the requirement | Y |
| 1. 5V power supply - Controller system and motor subsystem<br><br>The power subsystem should be capable of supplying power to the motor subsystem or Raspberry Pi at 5V $\pm$ 100 $mV$, to ensure stable operation of the motors. | 2. Verification<br><br>a. Plug in the battery to the PCB.<br>b. Connect an oscilloscope channel pair across the output voltage of the buck converters inductor, and the ground plane.<br>c. Measure the average voltage as well as the peak to peak ripple. Compare these values to the requirement | Y |
| 2. Motor Button Response and Tracking Control Latency<br><br>The ESP32-S3 must process manual button inputs and subject tracking data and send | 3. Verification<br><br>a. Ensure the ESP32S3 is plugged in via Micro USB.<br>b. Open a serial terminal to monitor the ESP32S3 output. | Y |

| | | |
|---|---|---|
| servo motor commands with ≤125 ms latency, allowing ample time for motors to receive and react to commands within the 250ms high level requirement | c. Press a motor button and verify the two timestamps(time at button pressed, time at websockets sent) are within 125ms of each other.<br><br>Timestamps will be recorded within ESP32S3 flashed firmware. | |
| 3. Camera Button Response Latency<br><br>The ESP32-S3 must process manual button inputs for camera control and send iPhone camera control commands with ≤250 ms latency, allowing sample time for the iPhone to receive and react to commands within the 500ms high level requirement. | 4. Verification<br><br>a. Ensure the ESP32S3 is plugged in via Micro USB.<br>b. Open a serial terminal to monitor the ESP32S3 output.<br>c. Press a camera button and verify the two timestamps(time at button pressed, time at websockets sent) are within 250ms of each other.<br><br>Timestamps will be recorded within ESP32S3 flashed firmware. | Y |
| 4. Network Throughput Capacity<br><br>The ESP32-S3's dedicated WiFi network must support simultaneous motor control, camera commands, and tracking data transmission without exceeding 80% of available bandwidth to prevent congestion. | 5. Verification<br><br>a. Ensure the ESP32S3 is plugged in via Micro USB.<br>b. Open a serial terminal to monitor the ESP32S3 output<br>c. Enter test_network into the terminal to check network statistics. The maximum bandwidth will be printed.<br>d. Now, begin running Airplay and enabling tracking mode on the tripod.<br>e. Enter the following command into the ESP32S3 terminal: network_status<br>f. Ensure that the output does not show network exceeding 80% bandwidth over normal operation<br><br>ESP32S3 firmware will utilize the esp_wifi and WIFI C++ libraries to monitor network traffic. | N |
| 5. Network Stability<br><br>The ESP32-S3 must maintain a stable connection with ≤5% packet loss, ensuring reliable data transmission across all system components. | 6. Verification<br><br>a. Ensure the ESP32S3 is powered.<br>b. Log in to the ESP32S3's WiFi network using an external computer.<br>c. Run this command:<br>**ping 192.168.4.1 -n 100** (default IP address of | Y |

| | | |
|---|---|---|
| | ESP32S3)<br>d. Ensure that the output has packet loss less than or equal to 5% | |
| 6. Power Consumption - Controller System and Motor System<br><br>The ESP32-S3 must operate a ≤1.5W power budget to ensure efficiency in portable use and to prevent excessive overheating. | 7. Verification<br><br>a. Unplug the battery from the remote control PCB.<br>b. Unplug the Raspberry Pi 5v power connector from the PCB, and power using the USBC port instead.<br>c. Plug in the MicroUSB cable to power the ESP32S3, but keep the USB/Battery switch on the Battery setting.<br>d. Connect the 3v3USB pin directly to the 3.3V pin on the ESP32S3(or equivalent trace) using a multimeter.<br>e. Wait for ESP32S3 to boot up and begin normal operation, including Airplay. Then measure the current draw(average over span).<br>f. Calculate power (P = V * I) and ensure the value is below 1.5W | Y |
| 7. AirPlay Streaming Latency<br><br>The iPhone's screen must stream to the Raspberry Pi with ≤1s latency at ≥24 FPS for real-time monitoring. | 8. Verification<br><br>a. Ensure that the Raspberry Pi is powered, and ready to receive Airplay connection.<br>b. Connect the iPhone to the Airplay server and verify that the screen is mirrored on the display.<br>c. Ensure that the displayed FPS is at least 24 FPS<br>d. On the mirrored iPhone, begin a timer for ten seconds, and begin a secondary timer simultaneously with a separate device.<br>e. Stop the secondary timer when the remote display shows the timer being completed. Ensure that the result is within 1 second of the expected 10s timer.<br>f. Repeat 5 - 10 times until certain latency is under 1 second. | Y |
| 8. OpenCV Tracking Accuracy<br><br>The OpenCV algorithm should process and send tracking commands to keep the Subject within a 190x160 centered rectangle on the display(340x240) | 9. Verification<br><br>a. Power on the motorized tripod.<br>b. Plug into the Raspberry Pi via USBC, and log into the desktop with microHDMI.<br>c. Make sure the Raspberry Pi is unconnected from the PCB power source, | Y |

| | | |
|---|---|---|
| | and plug in the remote controllers battery to start the ESP32S3.<br>d. On the Raspberry Pi, run the following file: tracking_test.py<br>e. Connect your iPhone to the Airplay, and initiate a tracking scenario.<br>f. Verify that the algorithm is able to keep the subject within the 190x160 rectangle at a moderate speed. | |
| 9. OpenCV Processing Speed<br><br>The Raspberry Pi must process subject tracking data and send tracking commands with ≤500 ms latency to ensure smooth camera adjustments. | 10. Verification<br><br>a. Ensure that the Raspberry Pi is powered, and ready to receive Airplay connection<br>b. Plug in a micro HDMI cable to the Raspberry Pi to view the desktop.<br>c. Run the following python file: processing_speed_test.py<br>d. Connect your phone to the Airplay server, and set up a tracking scenario with a subject.<br>e. Observe the timestamps recorded (just before the subject is detected and just after the tracking command is sent) and ensure the difference is within 500ms. | Y |
| 10. Power Consumption<br><br>The Raspberry Pi and Waveshare LCD must operate within a ≤10W(Raspberry Pi 4 full stress power consumption) power budget to ensure efficiency in portable use and to prevent excessive overheating. | 11. Verification<br><br>a. Ensure the remote control PCB is powered by the battery.<br>b. Remove the 5V Raspi Power connector and replace it with a multimeter.<br>c. Begin normal tracking operation of the system, and measure the current draw(average over time).<br>d. Calculate the power (P = V * I) and ensure it is below 10W. | Y |
| 11. Motor Button Response and Tracking Control Latency<br><br>The ESP32-S3 must send stepper motor commands after receiving with ≤125 ms latency, allowing accurate positioning with relatively small delay | 12. Verification<br><br>a. Ensure the ESP32S3 on the tripod PCB is plugged in via Micro USB, and that the remote controller is turned on and connected.<br>b. Open a serial terminal to monitor the ESP32S3(tripod PCB) output.<br>c. Initiate a motor command by pressing a motor button on the remote controller and verify the two timestamps(time at websockets received, time at PWM sent) | Y |

| | | |
|---|---|---|
| | are within 125ms of each other.<br><br>Timestamps will be recorded within ESP32S3 flashed firmware. | |
| 12. Network Stability<br><br>The ESP32-S3 must maintain a stable connection with ≤5% packet loss, ensuring reliable data transmission between the controller and the tripod. | 13. Verification<br><br>a. Ensure the ESP32S3 on the tripod PCB is plugged in with Micro USB, and connected to the remote controller network.<br>b. Open a serial terminal to monitor the ESP32S3(tripod PCB) output.<br>c. Type the following command into the terminal: network_status<br>d. Read the IP address, and then on a separate computer, log into the remote controller's network and run this command: **ping <IP_address> -n 100**<br>e. Verify that the packet loss is less than or equal to 5%. | Y |
| 13. Motor power consumption<br><br>The motors should draw ≤ 7.5 W of power at any given moment, to ensure safe power consumption and decent battery lifetime of the tripod. | 14. Verification<br><br>a. Disconnect the motors from the PCB.<br>b. Connect an ammeter in series with the ground wire of the motor. Connect a voltmeter between the power and ground terminals of the motor.<br>c. Start the motorized tripod and use it under normal operating conditions.<br>d. Watch the two meters while running under normal conditions, and record the maximum values that appear across both the meters.<br>e. Calculate power (P = V * I) and ensure the value is below 7.5 W. | Y |
| 14. Motor Step Accuracy<br><br>The motor subsystem should be capable of stepping in ≤ 2° increments. This will create fine control over position to accurately position the tripod's camera holder. | 15. Verification<br><br>a. . Ensure the ESP32S3 on the tripod PCB is plugged in with Micro USB, and connected to the remote controller network.<br>b. 2. Place a phone on the tripod mount. Open a gyroscope app on the iphone.<br>c. 2. Open a serial terminal to the ESP32S3(tripod PCB)3. Type the following command into the terminal: turn_right.<br>d. 4. Record the difference in position. Compare this to the requirement. | Y |
| 15. Camera Command Execution Latency | 16. Verification | Y |

| | | |
|---|---|---|
| The app must execute camera control commands (e.g., photo capture, video recording, zoom adjustments) within 250 ms after receiving them via WebSockets over WiFi, allowing ample time for the ESP32-S3 to send commands to adhere to the 500 ms latency high level requirement. | a. Ensure the remote controller system is turned on and running.<br>b. Connect your iPhone to the remote controller's network, and plug into a computer via USB.<br>c. Open and run the application through Xcode and view the console.<br>d. Initiate a camera action by pressing one of the camera buttons on the remote control.<br>e. View the timestamps in the console(when websockets command is received, and when the camera action is initiated) and ensure they are within 250ms. | |
| 16. Power Consumption<br><br>The iPhone application and Airplay action must operate within a ≤6W power budget to ensure efficiency in portable use and to prevent excessive overheating. | 17. Verification<br><br>a. Ensure the remote controller system is turned on and running.<br>b. Connect your iPhone to the remote controller's network, and plug into a computer via USB.<br>c. Open and run the application through Xcode.<br>d. Initiate Airplay connection to the Raspberry Pi.<br>e. In Instruments, click on the Energy Usage Graph, and locate the average power consumption. Ensure that this value stays less than or equal to 6W during normal operation. | Y |

# Appendix B: Miscellaneous

## Repositories:

Raspberry Pi and ESP32-S3 code: https://github.com/henry-thomas-1/ECE-445-Smart-Tripod/tree/main

IOS Application code: https://github.com/migueldomingo13/ECE-445-Smart-Tripod-App

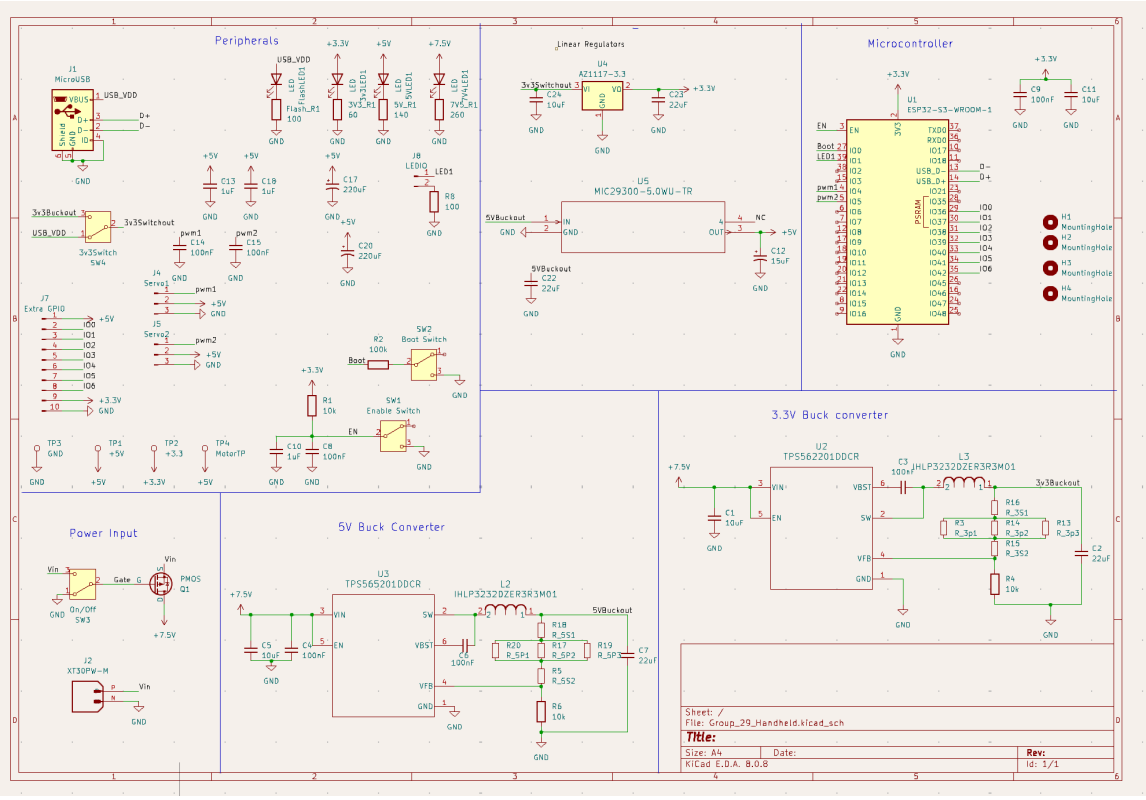


Figure 16: Motorized Tripod PCB schematic

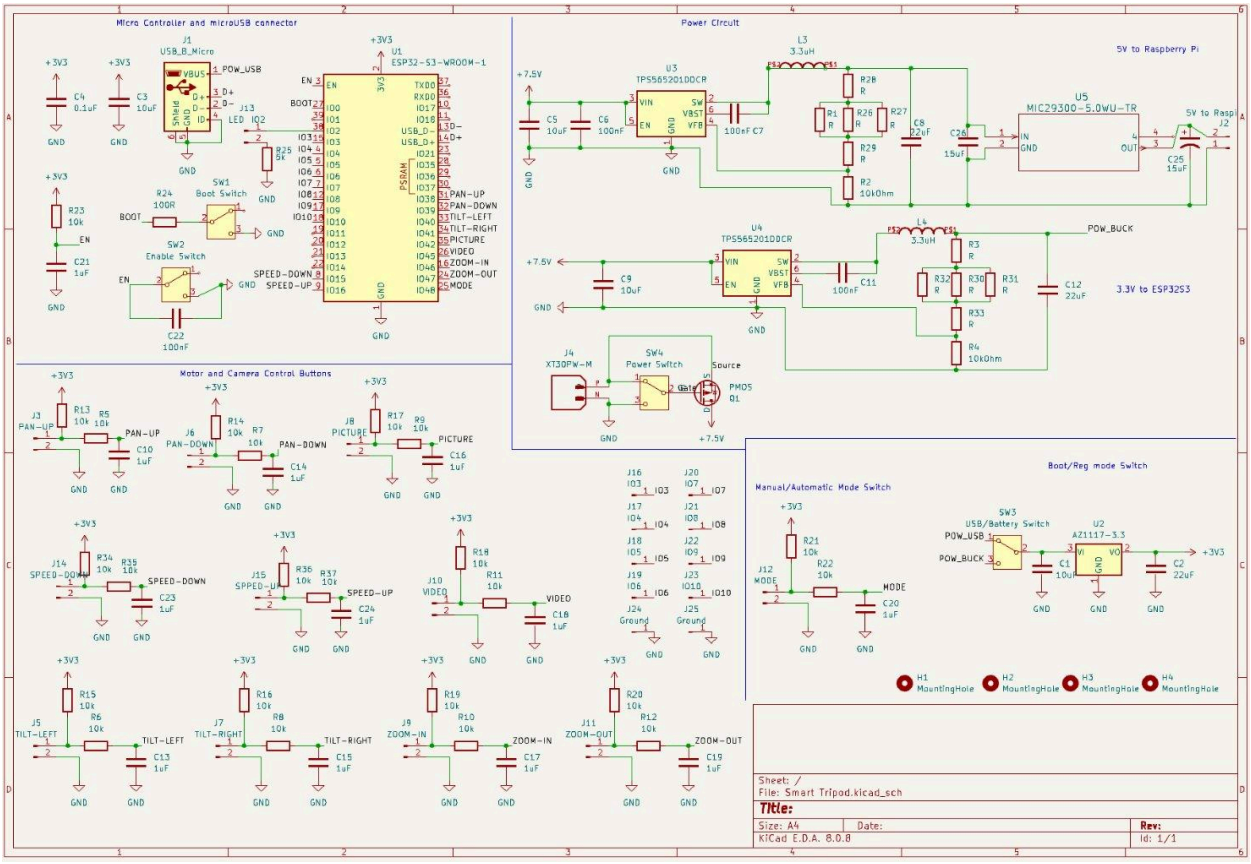


Figure 17: Remote Display/Controller PCB Schematic

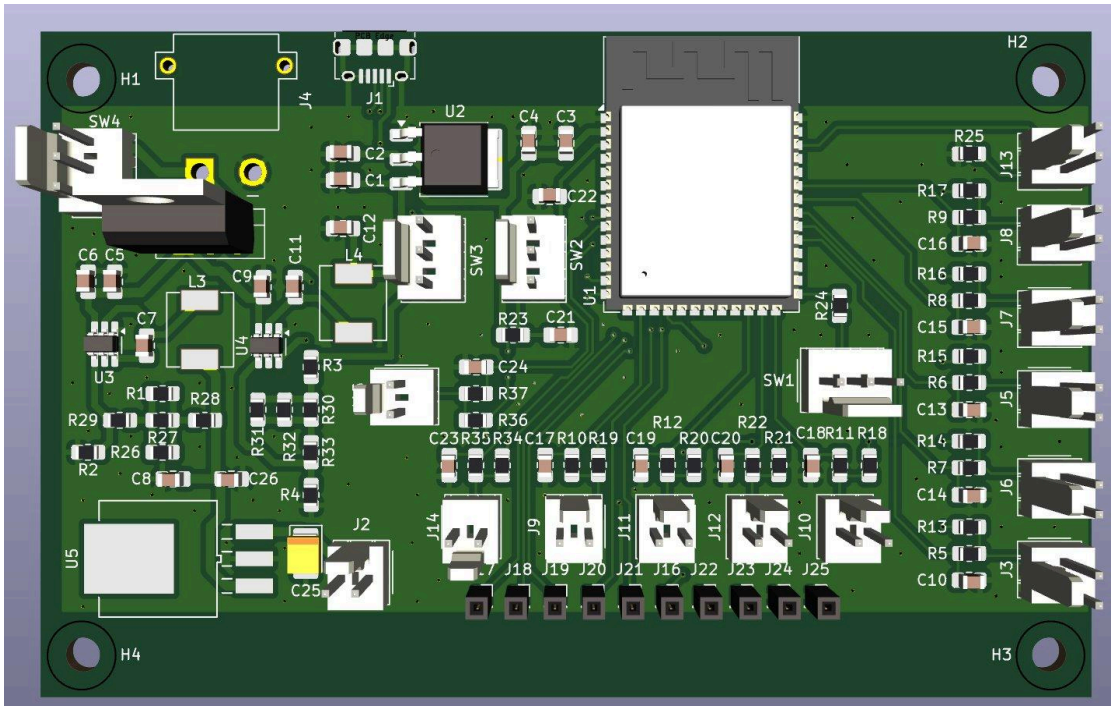


Figure 18: Remote Display/Controller PCB

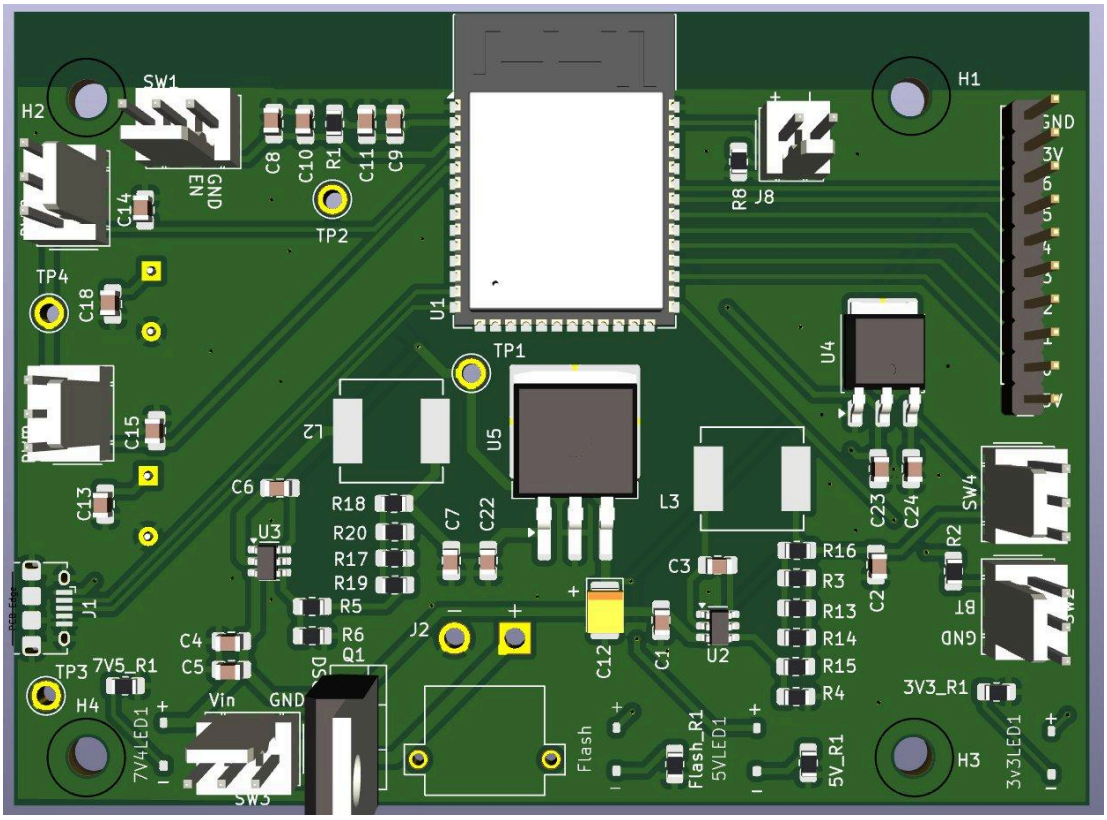


Figure 19: Motorized Tripod PCB

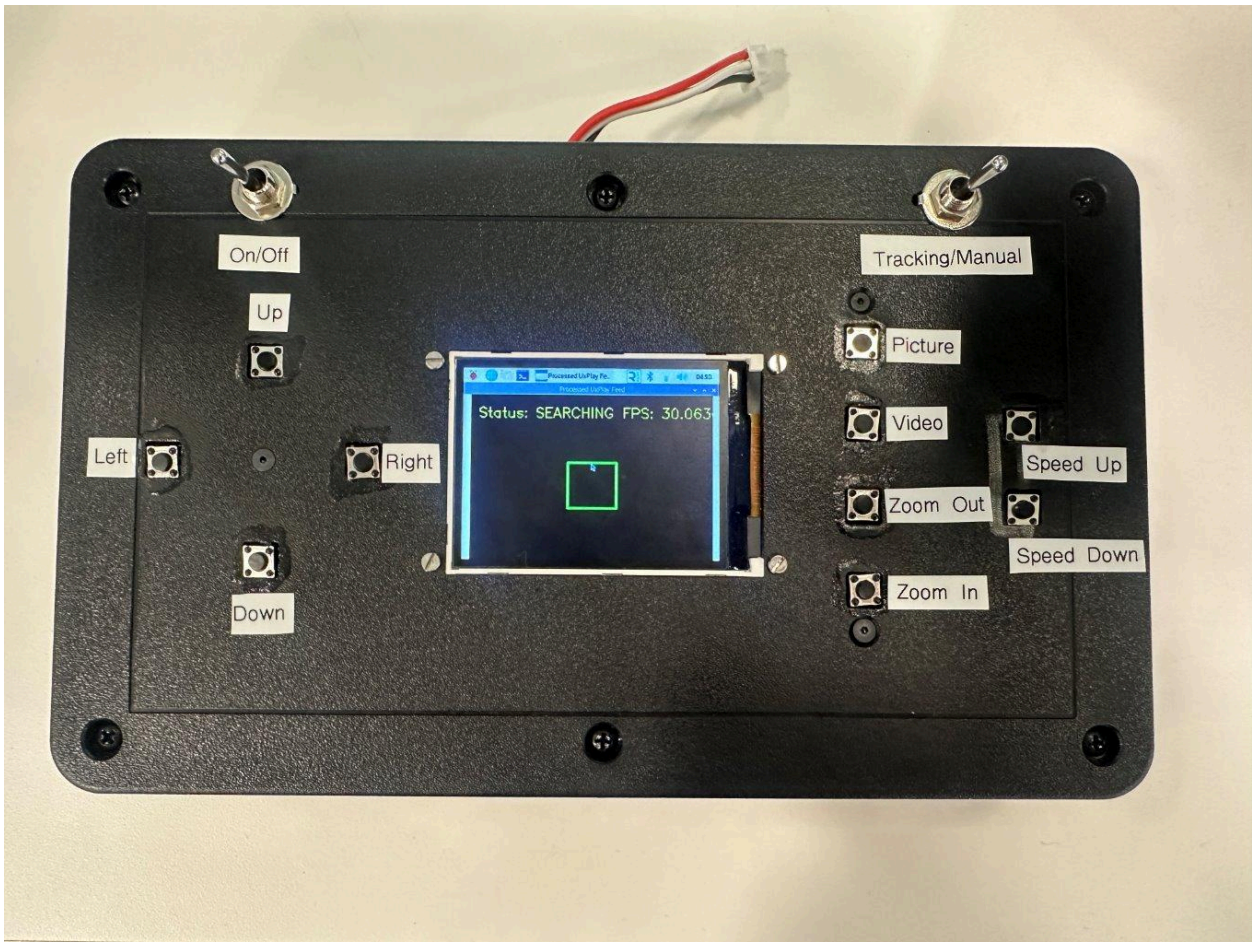


FIgure 20: Physical Design of Controller system

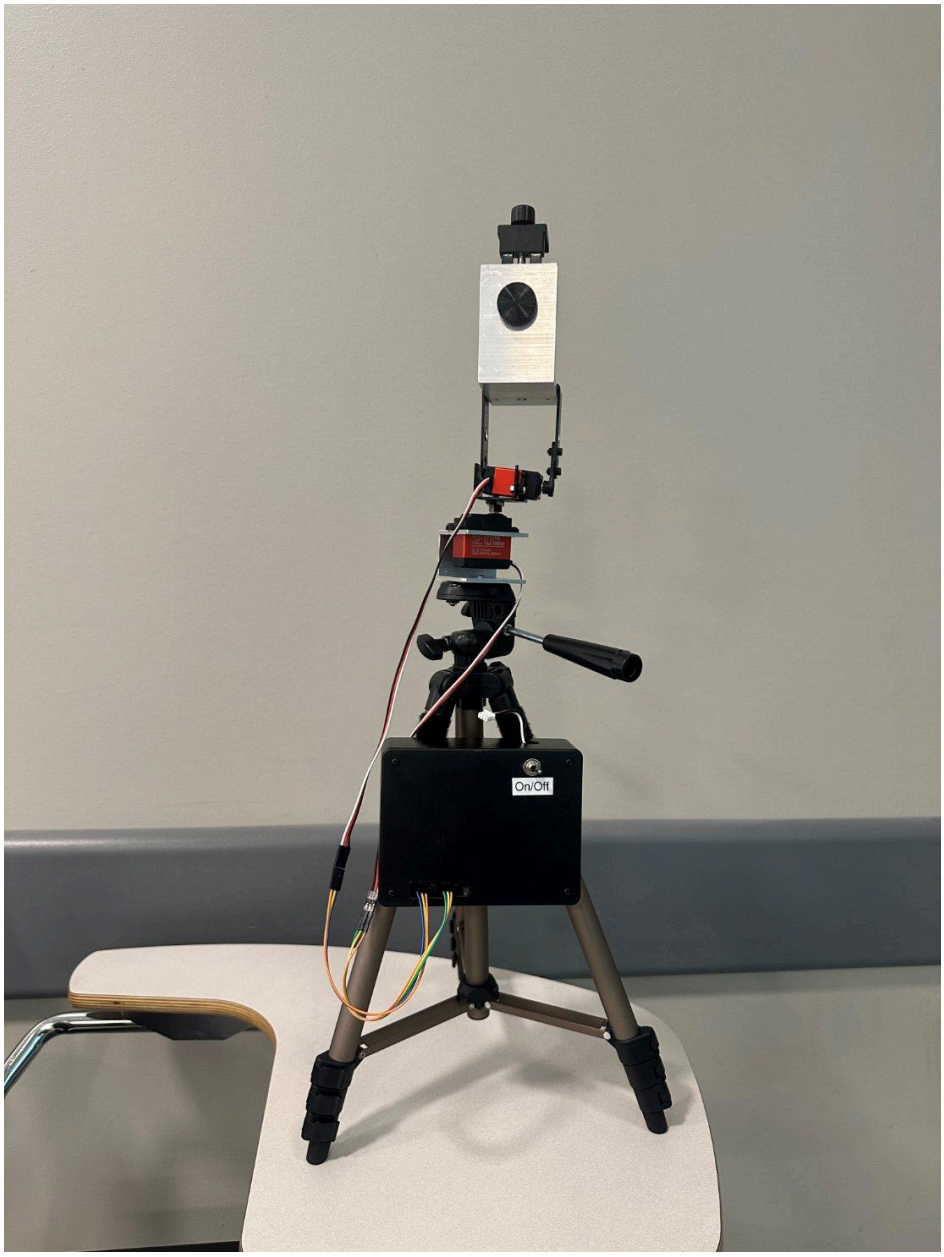


Figure 21: Physical Design of Tripod system

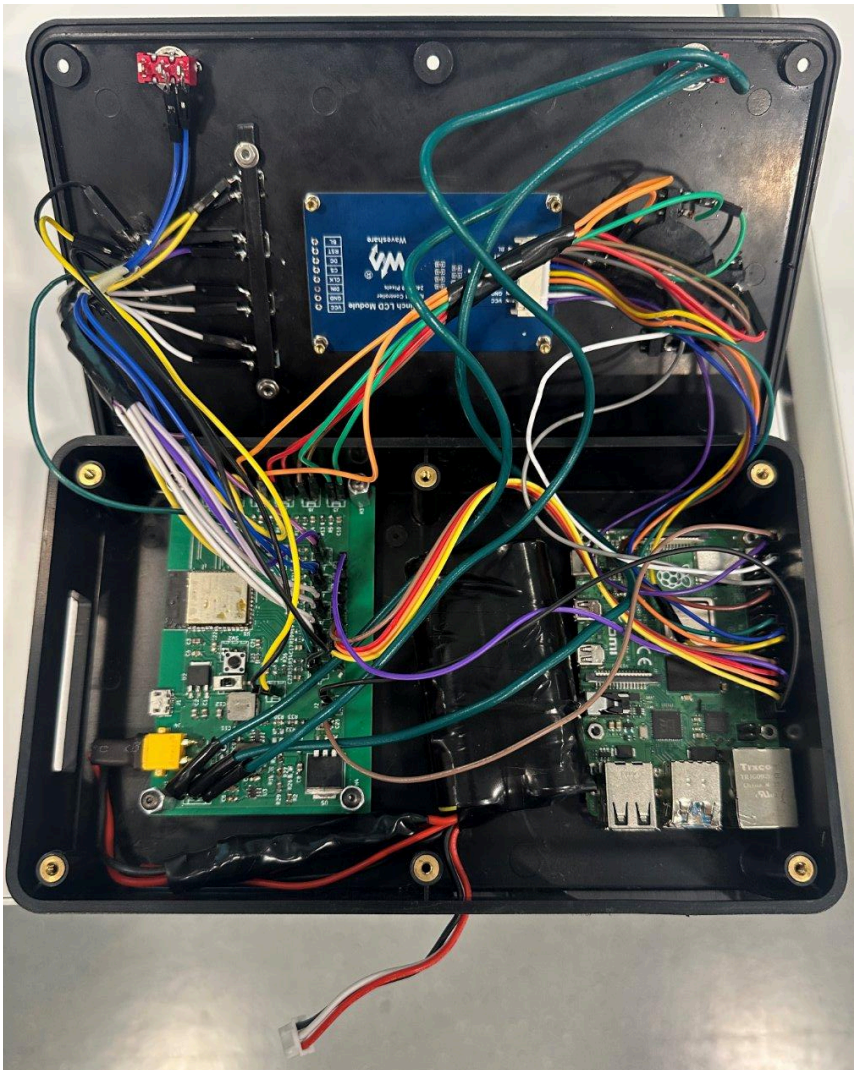

FIgure 22: Remote Display/Controller box Internals

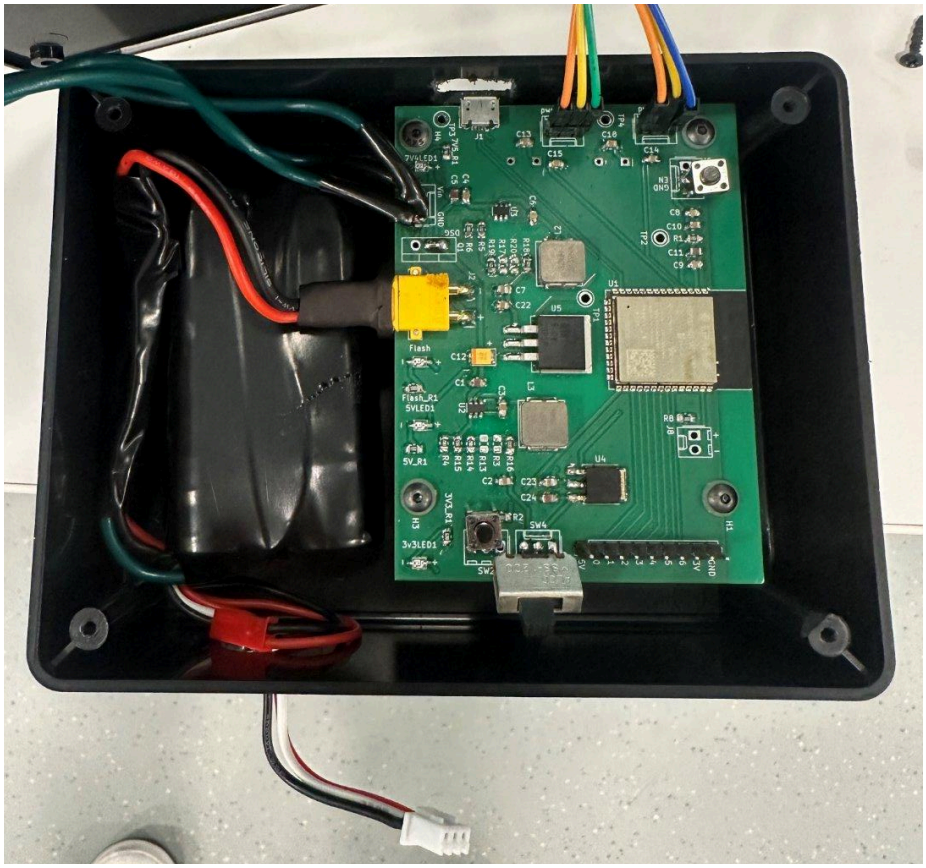

Figure 23: Motorized Tripod System Electrical Box Internals

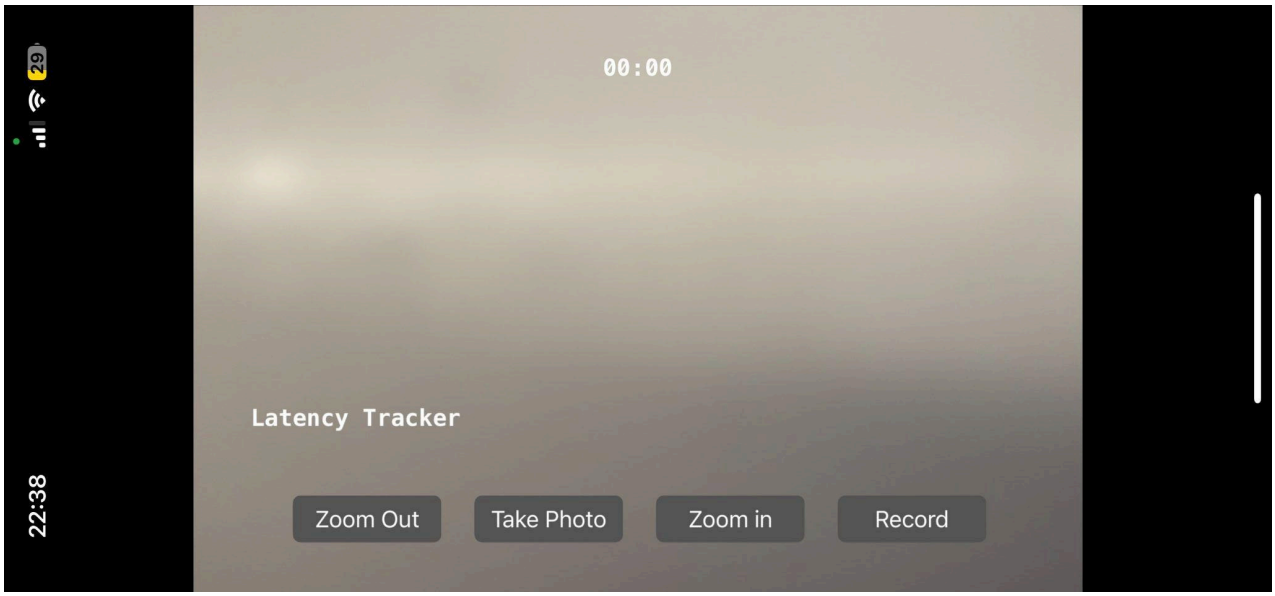


Figure 24: Smart Tripod IOS Application UI