ECE 445
Senior Design Laboratory
Design Document

Smart Tripod

Team No. 29
Henry Thomas
Miguel Domingo
Kadin Shaheen

Table of Contents

1. Introduction

1.1 Problem

Traditional tripods provide stability for cameras and smartphones but lack dynamic adjustability and real-time framing assistance. When setting up a shot, users must manually adjust the tripod's angle and position, often requiring multiple iterations to achieve the perfect frame. This process can be frustrating and time-consuming, especially for solo photographers, vloggers, or group shots where precise positioning is essential. Additionally, traditional tripods do not adapt to movement, making them impractical for scenarios where the subject may shift position, such as recording personal videos, filming demonstrations, or capturing action shots.

While some motorized tripods exist, they often have limited functionality, primarily offering simple pan-and-tilt adjustments without real-time feedback. Most lack an integrated system to preview the camera's live feed, requiring users to make adjustments blindly or rely on trial and error. Furthermore, existing solutions do not offer automatic subject tracking, meaning users must manually control the tripod's movements to keep themselves or their subjects centered in the frame. This gap in functionality creates a need for a more advanced, user-friendly solution that allows for remote tripod control, real-time framing adjustments, and automatic subject tracking, ultimately enhancing the convenience and precision of capturing high-quality photos and videos.

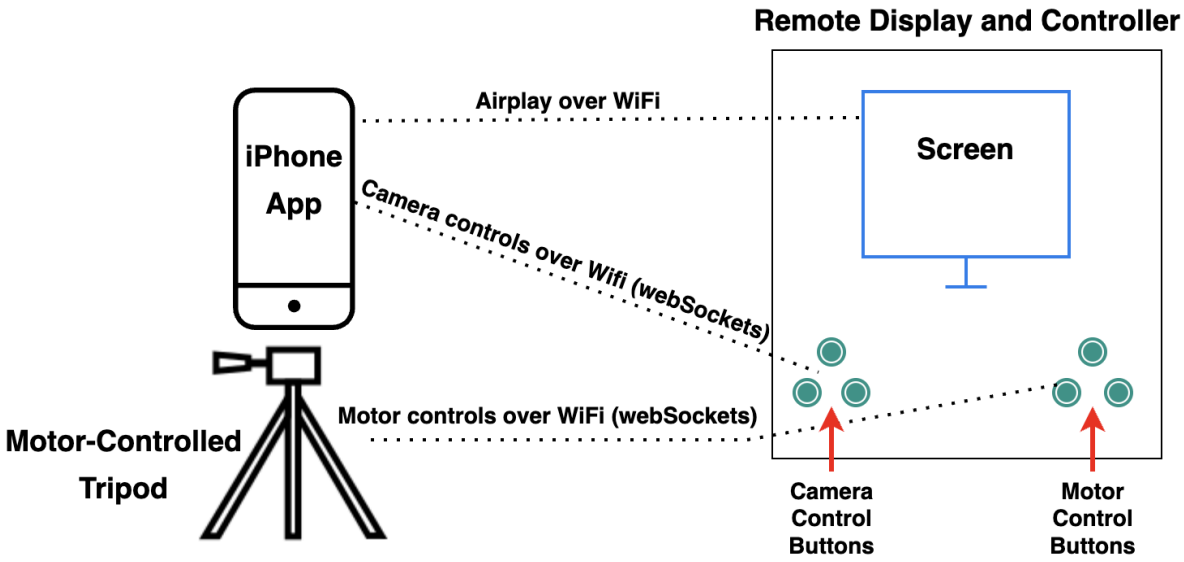
1.2 Solution

Our project introduces a Smart Tripod that enhances traditional tripods by incorporating motorized adjustments, wireless control, and automatic subject tracking. Unlike standard tripods that require manual repositioning, our system allows users to seamlessly adjust their smartphone camera's orientation using an external remote. This remote features an integrated display that mirrors the smartphone's screen, providing real-time feedback on framing and composition. In addition to live preview, the remote includes built-in controls for zooming, capturing photos, and recording videos, all seamlessly integrated through a dedicated smartphone app. By eliminating the guesswork involved in setting up a shot, users can make precise, remote adjustments with ease.

For users capturing personal videos or dynamic scenes, the Smart Tripod includes an advanced tracking mode that automatically adjusts the smartphone's orientation to keep the subject centered in the frame. This ensures smooth and professional-looking footage without the need for constant manual corrections. By combining motorized control, live screen sharing, app-integrated camera controls, and intelligent

tracking, our solution offers a seamless, user-friendly experience for capturing high-quality photos and videos with minimal effort.

1.3 Visual Aid



1.4 High-level requirements list

1.4.1 Motor Accuracy and Responsiveness

The Smart Tripod's motorized control system must provide fast and precise adjustments to ensure smooth framing and subject tracking. Specifically, the motors should respond to user inputs and automated tracking commands for azimuth and zenith adjustments within 250 milliseconds, with a positioning accuracy of ±2°. Given the use of stepper motors, we anticipate potential sources of delay from signal transmission, microcontroller processing, and motor inertia. To verify compliance, we will log and analyze motor response times using timestamped command executions and track angular accuracy using the iPhone's built in gyroscope and external motion capture.

1.4.2 iPhone Camera Action and Responsiveness

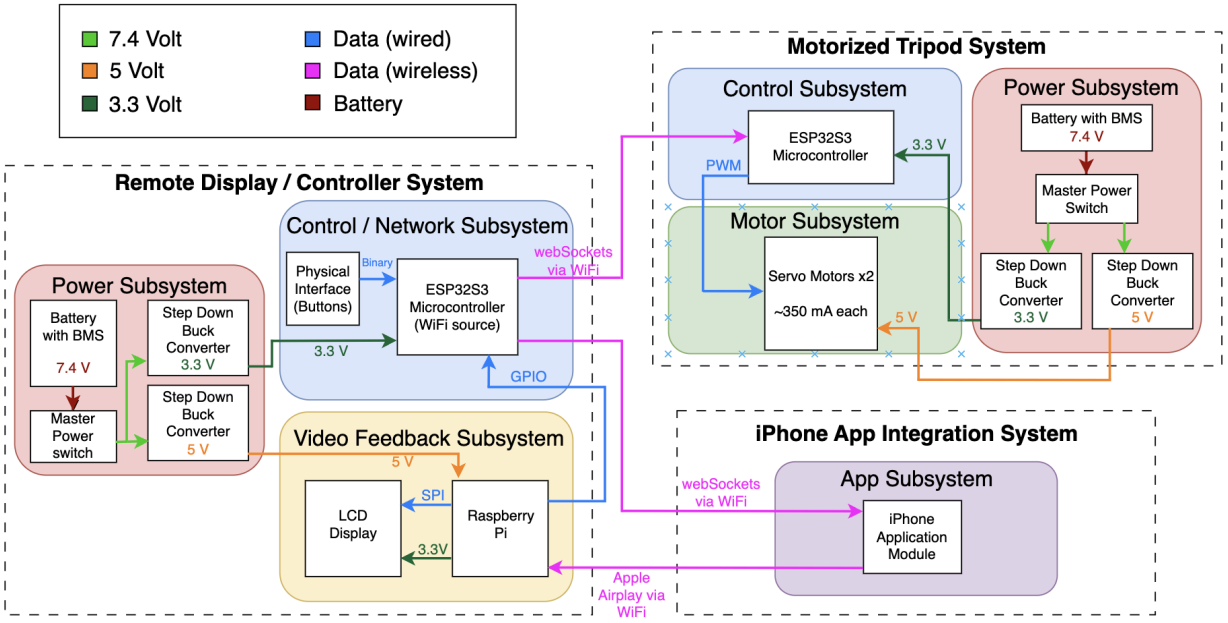
The system must provide real-time remote camera operation over WiFi, allowing users to capture photos, record videos, and adjust zoom with minimal delay. Camera actions triggered from the external remote interface should execute within 500 milliseconds via WebSockets communication to ensure a smooth user experience. Factors that may affect performance include network congestion, WebSocket transmission latency, and iPhone processing time. We will evaluate this requirement by measuring round-trip command execution time from the moment a user triggers an action on the remote to the confirmation of execution on the iPhone, using code-based timestamp logging and network performance analysis tools.

1.4.3 Airplay and Tracking Responsiveness

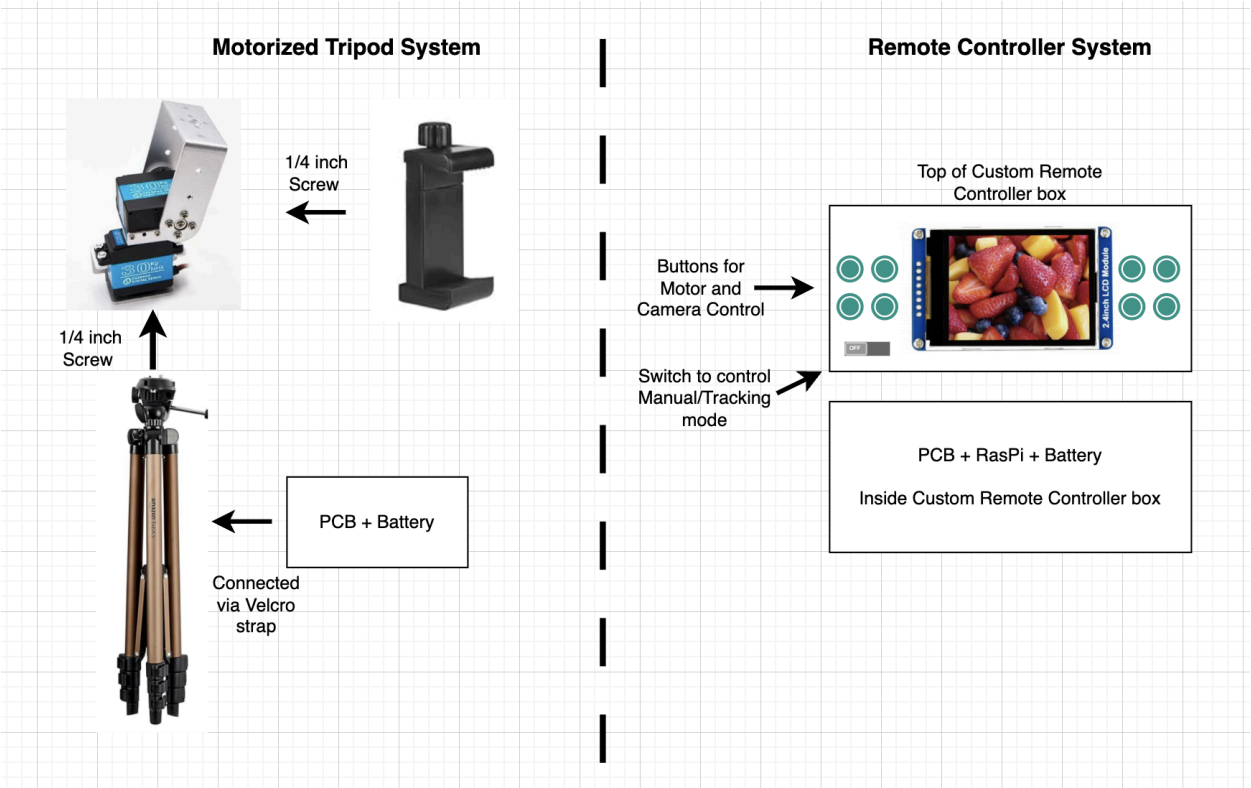
The Smart Tripod must provide low-latency, high-frame-rate live streaming from the iPhone to the external display via AirPlay. The video feed should maintain at least 24 FPS with <1 second of latency while streaming through a Raspberry Pi. Additionally, the system must perform subject detection via OpenCV 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

2.1 Block Diagram



2.2 Physical Design



The Motorized Tripod System, shown on the left, is built around a standard tripod with height-adjustable legs. A custom enclosure, fabricated by the machine shop, will house the PCB and battery for this section of the project. This enclosure will be securely fastened to the tripod using Velcro straps, allowing for easy removal when needed.

Atop the tripod, a pan-and-tilt servo bracket mount will be attached to the ¼-inch mounting screw. This bracket will hold the servo motors, enabling smooth two-axis rotation. A custom clamp mount, modified by the machine shop, will then be installed onto the bracket. This clamp will securely hold the iPhone and adjust its orientation based on user input.

The Remote Controller System will feature a custom enclosure housing the PCB, Raspberry Pi, and battery. Mounted on top of this enclosure will be an LCD display along with physical buttons for motor and camera control. Additionally, a dedicated switch will allow users to toggle between manual mode and tracking mode for the motorized tripod.

2.3 Subsystem Overview and Requirements

2.3.1 Remote Display / Controller System

2.3.1.1 Power Subsystem

Purpose and Interactions:

The power subsystem of the remote display / controller system is responsible for the maintaining power input to the control/network subsystem, and the video feedback subsystem. It is only intended to power the electronics onboard the remote controller. The power subsystem will consist of a battery with an internal BMS capable of supplying proper current for the two step downs. These two step downs will be from 7.4v to 5v for the Video Feedback subsystem and 7.4v to 3.3v for the Control/Network subsystem.

Requirements	Verification
3.3V power supply - 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. Plug in the battery to the PCB. 2. Connect an oscilloscope channel pair across the output voltage of the buck converters inductor, and the ground plane. 3. Measure the average voltage as well as the peak to peak ripple. Compare these values to the requirement
5V power supply - The power subsystem should be capable of supplying power to the motor subsystem at 5V $\pm$ 100 mV, to ensure stable operation of the motors.	1. Plug in the battery to the PCB. 2. Connect an oscilloscope channel pair across the output voltage of the buck converters inductor, and the ground plane. 3. Measure the average voltage as well as the peak to peak ripple. Compare these values to the requirement

2.3.1.2 Control / Network Subsystem

Purpose and Interactions:

The ESP32-S3 serves as the central controller for the tripod's motorized adjustments and iPhone camera control. The microcontroller will offer both manual and automated control. It features physical buttons for direct control of azimuth (pan) and zenith (tilt) movement, as well as dedicated buttons for zoom, photo capture, and video recording. A switch allows users to toggle between manual mode and subject tracking mode for greater flexibility.

For automated operation, the ESP32-S3 receives subject tracking data from the Raspberry Pi via GPIO, dynamically adjusting the tripod's position using stepper motors. It also communicates over WiFi via WebSockets, enabling remote control of both the motors and iPhone camera functions. To ensure seamless connectivity, the ESP32-S3 establishes a dedicated 2.4GHz WiFi network for system communication. This WiFi network will maintain the Airplay and webSocket

connections used throughout all of our subsystems. The remote control interface integrates a custom PCB and a power management system for efficient operation.

Requirements	Verification
Motor Button Response and Tracking Control Latency – The ESP32-S3 must process manual button inputs and subject tracking data and send stepper motor commands with ≤125 ms latency, allowing ample time for motors to receive and react to commands within the 250ms high level requirement.	1. Ensure the ESP32S3 is plugged in via Micro USB. 2. Open a serial terminal to monitor the ESP32S3 output. 3. Press a motor button and verify the two timestamps(time at button pressed, time at websockets sent) are within 125ms of each other. Timestamps will be recorded within ESP32S3 flashed firmware.
Camera Button Response Latency – 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.	1. Ensure the ESP32S3 is plugged in via Micro USB. 2. Open a serial terminal to monitor the ESP32S3 output. 3. Press a camera button and verify the two timestamps(time at button pressed, time at websockets sent) are within 250ms of each other. Timestamps will be recorded within ESP32S3 flashed firmware.
Network Throughput Capacity – 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.	1. Ensure the ESP32S3 is plugged in via Micro USB. 2. Open a serial terminal to monitor the ESP32S3 output 3. Enter test_network into the terminal to check network statistics. The maximum bandwidth will be printed. 4. Now, begin running Airplay and enabling tracking mode on the tripod. 5. Enter the following command into the ESP32S3 terminal: network_status 6. Ensure that the output does not show network exceeding 80% bandwidth over normal operation ESP32S3 firmware will utilize the esp_wifi and WIFI C++ libraries to monitor network traffic.

Network Stability – The ESP32-S3 must maintain a stable connection with ≤5% packet loss, ensuring reliable data transmission across all system components.	1. Ensure the ESP32S3 is powered. 2. Log in to the ESP32S3's WiFi network using an external computer. 3. Run this command: **ping 192.168.4.1 -n 100** (default IP address of ESP32S3) 4. Ensure that the output has packet loss less than or equal to 5%
Power Consumption – The ESP32-S3 must operate a ≤1.5W power budget to ensure efficiency in portable use and to prevent excessive overheating.	1. Unplug the battery from the remote control PCB. 2. Unplug the Raspberry Pi 5v power connector from the PCB, and power using the USBC port instead. 3. Plug in the MicroUSB cable to power the ESP32S3, but keep the USB/Battery switch on the Battery setting. 4. Connect the 3v3USB pin directly to the 3.3V pin on the ESP32S3(or equivalent trace) using a multimeter. 5. Wait for ESP32S3 to boot up and begin normal operation, including Airplay. Then measure the current draw(average over span). 6. Calculate power (P = V * I) and ensure the value is below 1.5W

2.3.1.3 Video Feedback Subsystem

Purpose and Interactions:

The Video Feedback Subsystem is responsible for collecting and processing the iPhone's video feed. It consists of a Raspberry Pi 4 and a Waveshare 2.4-inch SPI LCD screen for real-time monitoring. The Raspberry Pi runs RPiPlay to wirelessly receive the iPhone's camera feed via AirPlay, enabling smooth video streaming.

Using OpenCV, the Raspberry Pi processes the video stream to detect and track the subject's position. It then transmits tracking data to the ESP32-S3 via 3.3V GPIO communication, ensuring precise position updates. The ESP32-S3 interprets this data and dynamically adjusts the tripod's orientation to keep the subject centered in the frame. Operating in a continuous feedback loop, the system ensures real-time tracking and seamless camera adjustments.

Requirements	Verification
AirPlay Streaming Latency – The iPhone's screen must stream to the Raspberry Pi with	1. Ensure that the Raspberry Pi is powered, and ready to receive Airplay connection. 2. Connect the iPhone to the Airplay server

≤1s latency at ≥24 FPS for real-time monitoring.	and verify that the screen is mirrored on the display. 3. Ensure that the displayed FPS is at least 24 FPS 4. On the mirrored iPhone, begin a timer for ten seconds, and begin a secondary timer simultaneously with a separate device. 5. 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. 6. Repeat 5 - 10 times until certain latency is under 1 second.
OpenCV Tracking Accuracy - The OpenCV algorithm should process and send tracking commands to keep the Subject within a 190x160 centered rectangle on the display(340x240)	1. Power on the motorized tripod. 2. Plug into the Raspberry Pi via USBC, and log into the desktop with microHDMI. 3. Make sure the Raspberry Pi is unconnected from the PCB power source, and plug in the remote controllers battery to start the ESP32S3. 4. On the Raspberry Pi, run the following file: tracking_test.py 5. Connect your iPhone to the Airplay, and initiate a tracking scenario. 6. Verify that the algorithm is able to keep the subject within the 190x160 rectangle at a moderate speed.
OpenCV Processing Speed – The Raspberry Pi must process subject tracking data and send tracking commands with ≤500 ms latency to ensure smooth camera adjustments.	1. Ensure that the Raspberry Pi is powered, and ready to receive Airplay connection 2. Plug in a micro HDMI cable to the Raspberry Pi to view the desktop. 3. Run the following python file: processing_speed_test.py 4. Connect your phone to the Airplay server, and set up a tracking scenario with a subject. 5. 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.
Power Consumption – 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.	1. Ensure the remote control PCB is powered by the battery. 2. Remove the 5V Raspi Power connector and replace it with a multimeter. 3. Begin normal tracking operation of the system, and measure the current draw(average over time).

	4. Calculate the power (P = V * I) and ensure it is below 10W.

2.3.2 Motorized Tripod System

2.3.2.1 Power Subsystem

Purpose and Interactions:

The power subsystem of the tripod system is responsible for the maintaining power input to the tripod control subsystem, and the motor subsystem. It is only intended to power the tripod system and not the remote control. The power subsystem will consist of a battery with an internal BMS capable of supplying proper current for the motors, the step down, and the voltage regulation. The step down will be from 7.4V to 3.3v to power the ESP32S3 onboard the control system and a separate 5v step down to deliver power to the motor subsystem.. The power system will be chargeable through the battery pack connector, and will also contain a master power switch to ensure the battery is separated from the load when not in use.

Requirements	Verification
3.3V power supply - 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. Plug in the battery to the PCB. 2. Connect an oscilloscope channel pair across the output voltage of the buck converters inductor, and the ground plane. 3. Measure the average voltage as well as the peak to peak ripple. Compare these values to the requirement
5V power supply - The power subsystem should be capable of supplying power to the motor subsystem at 5V $\pm$ 100 mV, to ensure stable operation of the motors.	1. Plug in the battery to the PCB. 2. Connect an oscilloscope channel pair across the output voltage of the buck converters inductor, and the ground plane. 3. Measure the average voltage as well as the peak to peak ripple. Compare these values to the requirement

2.3.2.2 Control Subsystem

Purpose and interactions:

The control subsystem of the motorized tripod system will consist of an ESP32S3. It will interface with the ESP32S3 on the control/network subsystem of the remote control. The control subsystem of the tripod will communicate with the subsystem of the remote, to ensure the tripod is oriented properly. To orient the tripod properly, the control subsystem will interface with the

motor subsystem to translate azimuth and zenith angle instructions from the controller. It will receive power from the power subsystem of the tripod.

Requirements	Verification
Motor Button Response and Tracking Control Latency – The ESP32-S3 must send stepper motor commands after receiving with ≤125 ms latency, allowing accurate positioning with relatively small delay.	1. Ensure the ESP32S3 on the tripod PCB is plugged in via Micro USB, and that the remote controller is turned on and connected. 2. Open a serial terminal to monitor the ESP32S3(tripod PCB) output. 3. 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) are within 125ms of each other. Timestamps will be recorded within ESP32S3 flashed firmware.
Network Stability – The ESP32-S3 must maintain a stable connection with ≤5% packet loss, ensuring reliable data transmission between the controller and the tripod.	1. Ensure the ESP32S3 on the tripod PCB is plugged in with Micro USB, and connected to the remote controller network. 2. Open a serial terminal to monitor the ESP32S3(tripod PCB) output. 3. Type the following command into the terminal: network_status 4. 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** 5. Verify that the packet loss is less than or equal to 5%.
Power Consumption – The ESP32-S3 must operate a ≤1.5W power budget to ensure efficiency in portable use and to prevent excessive overheating.	1. Unplug the battery from the tripod PCB. 2. Plug in the MicroUSB cable to power the ESP32S3, but keep the USB/Battery switch on the Battery setting. 3. Connect the 3v3USB pin directly to the 3.3V pin on the ESP32S3(or equivalent trace) using a multimeter. 4. Wait for ESP32S3 to boot up and begin normal operation, including motor movements. Then measure the current draw(average over span). 5. Calculate power (P = V * I) and ensure the value is below 1.5W

2.3.2.3 Motor Subsystem

Purpose and interactions:
The motor subsystem is the physical interface between the tripod angling system and the control system. It is used to accurately position the zenith and azimuth angles of the phone, to allow for tracking or direction control based on user input. The motor subsystem will consist of two HS-311 servo motors. The servo motors will be capable of up to 270 degrees of rotation, allowing for full position control, while ensuring any external wires won't get crossed in any movement. They will receive power from the 5V buck converter, and will interface with the control system via a 3.3V PWM signal.

Requirements	Verification
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.	1. Disconnect the motors from the PCB. 2. Connect an ammeter in series with the ground wire of the motor. Connect a voltmeter between the power and ground terminals of the motor. 3. Start the motorized tripod and use it under normal operating conditions. 4. Watch the two meters while running under normal conditions, and record the maximum values that appear across both the meters. 5. Calculate power (P = V * I) and ensure the value is below 7.5 W.
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.	1. Ensure the ESP32S3 on the tripod PCB is plugged in with Micro USB, and connected to the remote controller network. 2. Place a phone on the tripod mount. Open a gyroscope app on the iphone. 2. Open a serial terminal to the ESP32S3(tripod PCB)3. Type the following command into the terminal: turn_right. 4. Record the difference in position. Compare this to the requirement.

2.3.3 iPhone App Integration System

2.3.3.1 App Subsystem

Purpose and Interactions:

A custom app will use WebSockets to receive commands from the ESP32-S3 over WiFi, allowing control of iPhone camera functions via AVFoundation, such as starting/stopping video recording,

capturing photos, and adjusting zoom. The live camera feed will be transmitted via AirPlay for real-time viewing on the external display. The app will also provide feedback regarding connection issues experienced during use to ensure a better user experience.

Requirements	Verification
Camera Command Execution Latency – 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.	1. Ensure the remote controller system is turned on and running. 2. Connect your iPhone to the remote controller's network, and plug into a computer via USB. 3. Open and run the application through Xcode and view the console. 4. Initiate a camera action by pressing one of the camera buttons on the remote control. 5. View the timestamps in the console(when websockets command is received, and when the camera action is initiated) and ensure they are within 250ms.
Power Consumption – The iPhone application and Airplay action must operate within a ≤6W power budget to ensure efficiency in portable use and to prevent excessive overheating.	1. Ensure the remote controller system is turned on and running. 2. Connect your iPhone to the remote controller's network, and plug into a computer via USB. 3. Open and run the application through Xcode. 4. Initiate Airplay connection to the Raspberry Pi. 5. 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.

2.4 Tolerance Analysis

Since our system relies on fast and precise communication between the iPhone, remote display/controller, and motorized tripod, it is crucial to minimize overall system delay. To ensure smooth operation, we must analyze and optimize response times at every stage, including command transmission, processing, and motor adjustments. Reducing latency in automatic mode is especially critical for real-time tracking, ensuring the tripod can continuously adjust to keep the subject centered without noticeable lag.

Seamless communication between devices is essential for ensuring the overall responsiveness of our system. Given the importance of minimizing delay, we must identify and optimize any sources of latency that could impact performance. This includes transmission delays, processing times, and synchronization

across all components. To effectively analyze these potential bottlenecks, we categorize them into three critical areas:

1. WiFi WebSockets Delay between ESP32-S3 to iPhone / Motorized Tripod.
2. AirPlay Streaming Delay from iPhone to Raspberry Pi.
3. OpenCV Processing Delay and its impact on tracking responsiveness.

In all areas the intention is to utilize delay equations to be able to estimate and quantify these potential delays.

2.4.1 WiFi Websockets Delay (ESP32-S3 to iPhone / Motorized Tripod)

Our plan is to have the ESP32 microcontroller send WebSockets commands over WiFi to the iPhone and the ESP32 microcontroller on the tripod.

In any network, the delay is made up of multiple components that sum up to the total delay between sending and receiving between two different devices. In sending WebSockets commands the main delay components are:

- **Processing Delay**: Time for the microcontroller to process and send a command
- **Queueing Delay**: time waiting in the network for transmission
- **Transmission Delay**: time it takes to transfer each bit onto the network
- **Propagation Delay**: time it takes for signal to travel over the WiFi channel

Dtot = Dproc + Dq + Dt+ Dp

Through our research thus far, we've been able to find that on average the **Processing delay** of the ESP32 is roughly **~2.5ms**. The **Queuing delay** for a lightly loaded wifi network is roughly **>1ms**.

<u>Transmission Delay</u>

Transmission depends highly on the size of the packets needed to be sent as well as the wifi speed. A WebSockets command is 256 bytes or 2048 bits. From research we've found that in the best circumstances the ESP32 is able to transmit at roughly 30Mbps.

Transmission Delay can then be calculated using:

Dt = L / R, where L = length of packet (bits) and R = Link bandwidth (bps)

Dt = 2048 / (30 * 10^6) = 0.00006826666 = **0.0682 ms**

<u>Propagation Delay</u>

This delay as mentioned above refers to the time it takes for the signal to travel over the WiFi channel. It is as simple as:

Dp = d / s, where d = distance and s = speed of propagation.

We will for now calculate the propagation delay for d = 5 meters , which should be the average distance that this tripod will be used from.

For wifi, the signal propagates roughly the speed of light (~3*10^8 m/s).

So in terms of our calculations Dp = 5 / (3*10^8) = **16.67 nanoseconds.** This is essentially negligible for our system.

Total Delay for WebSocket Commands

With our calculations thus far, we can now calculate the total delay we might experience from each WebSocket command being sent to the app.

Dt = Dproc + Dq + Dt + Dp

Dt = 2.5 ms + 1 ms + 0.0682 ms + 1.6 ns = **3.5862 ms**

This value refers to an estimated value of the delay that will incur from ESP32 when sending and receiving commands.

ESP32 to Motor

WebSocket Commands have a huge role in sending the necessary data to the motors for adjustment. The calculation above is crucial as that is the amount of time it takes for WebSocket commands to be sent and processed.

We've been able to estimate the time it'll take for a stepper motor to make small adjustments. In our research we've found that in typical motor applications, stepper motors require around 50-150 ms to complete small adjustments. This means that **100ms** would be a reasonable estimate for quantification purposes.

With the addition of having to send the WebSockets command and processing first we can find the total delay for the motors actually moving by summing up these two values.

Dmotor = 3.5862 + 100 = 103.586 ms

2.4.2 AirPlay Streaming Delay from iPhone to Raspberry Pi

The other area where we want to examine performance is how the AirPlay streaming from our iPhone to the Raspberry Pi will work. The AirPlay streaming latency introduces four main components:

- **Encoding delay (Dencode): Time it takes to send the stream**
- **Decoding delay (Ddecode): Time it takes to decompress video**
- **Rendering delay (Drender): Time it takes to display the frame**

- **Transmission delay (Dt): Time it takes to send the stream**

Equation: Dairplay = Dencode + Ddecode + Render + Dt

Encoding Delay

The iPhone typically compresses videos into H.264 before sending the videos over AirPlay. We found that the typical encoding times for H.264 at 720p/30FPS can range from 50-150ms. These times depend on the video resolution, the iPhone's hardware acceleration, and as well as the Codec used, H.264 should be efficient enough.

Apple's hardware encoder has the ability to encode a 720p video in **~100ms**.

Transmission Delay

When the video is encoded, it will have to be transmitted over WiFi. With the assumption of a 10 Mbps throughput from the ESP32, the transmission time for a 720p frame (~1.5Mbits) would be:

1.5/10 = 0.15 sec = **15ms.**

Decoding Delay

Once the video has been transmitted to the Raspberry Pi, the Pi must now utilize its hardware decoder to decompress the video stream. We've found that this typically takes around 30-60ms. We will choose a value closer to 60ms in order to account for potentially buffering and synchronization issues.

Ddecode = **50ms**

Rending Delay

Once decoded, we must then render frame-by-frame on the display. For quantification purposes, we find that a 60Hz display refreshes every 16.7ms, so rendering will typically take one frame cycle roughly **20ms**.

Total AirPlay Latency

Our total AirPlay Latency estimate will be the sum of all the main components in our delay calculations.

Dairplay = Dencode + Ddecode + Render + Dt

Dairplay = 100 + 15 + 50 + 20 = **185ms**

A value of 185ms is reasonable as our research shows that Apple TV's AirPlay latency is reported to be around 150-200 ms for our goal resolution and frame rate. We also take into account the

overhead that comes from the Raspberry Pi, so all in all this would be very reasonable for our intended purpose.

2.4.3 OpenCV Processing Delay and Impact on Tracking Responsiveness

This portion of the latency calculations is the latency regarding what will be our OpenCV algorithms that will be processed once the Raspberry Pi receives video from iPhone AirPlay. OpenCV is crucial for the ability to detect and track objects in real time. With this, it introduces three main components:

- **Capture Delay (Dcapture): Frame capture time from AirPlay video input**
- **Processing Delay (Dprocess): Time it takes for OpenCV to process and track objects**
- **Transmission Delay (Dt): Time it takes to transmit tracking data via WebSockets**

Equation:
Dopen_cv = Dcapture + Dprocess + Dsend

Frame Capture Delay
This essentially just takes into account the kind of delay we'll perceive depending on the chosen frame rate for our real-time video. As stated in our requirements, we want the video to be at least output at >24fps.

This essentially means that the time for each frame would be:
Dcapture = 1 / 24 = 0.042 seconds = **42ms**

OpenCV Processing Delay
OpenCV processing delay relies heavily on the tracking algorithm of our choice. In our research we've found that there are many choices for us to choose from when developing our tracking algorithm. Our research shows that some of the fastest can process **~100 to 300 ms**. This is reasonable for now, but a lot of factors definitely affect it such as the complexity of calculations and how we'll handle frame resolution and processing.

A choice of Dproc = **200ms** should be sufficient enough to at least give us some idea to quantify the OpenCV delay.

Tracking Data Transmission
There is an extra delay when sending the tracking commands through the GPIO pins between the Raspberry pi and the ESP32.

We can calculate this delay by taking the length of the wired connection (~30mm) and dividing by the speed of electricity through copper (~3*10^8m/s).

Dsend = .03 / (3*10^8) = **0.1ns**

This value is small, and can be neglected.

There is an additional delay due to the reading of the GPIO pin state, and for an ESP32 this takes typically between 0.1-1ms, which again is negligible for our system.

Total OpenCV Processing Delay

With all our previous calculations, we can now quantify a reasonable estimate for the delay we can expect from our OpenCV algorithm when creating motor data from our tracking and processing of the AirPlay input.

Dopen_cv = Dcapture + Dprocess
Dopen_cv = 42 + 200 = **242 ms**

This is a reasonable estimate for our OpenCV since it heavily relies on our tracking algorithm. Many factors go into reducing this number and we'll have to take this into account when developing our OpenCV tracking algorithm for our project.

2.4.4 Total Delay of System and Conclusion

Total Estimated Delay of System
With all necessary calculations and delay estimates quantified we can sum all values up to get an estimate for the total delay of the entire system in real-time.

For further analysis we have decided to calculate the Worst-Case Total Latency for the system. We are choosing to add 100ms to account for network queuing with the ESP32. This means that our final equation becomes:

Dtotal = Dairplay + Dopencv + Dweb_socket + Dnetwork + Dmotor

Dtotal = 3.5862 + 103.5862 + 185 + 242 + 100 = **634.172ms**

This remains well within our target total round-trip time of under 1.75 seconds (1 second for AirPlay transmission, 500 ms for OpenCV processing and tracking commands, and 250 ms for motor response). When the tripod is operating in manual mode, the delay should be far smaller as there will not be an OpenCV processing element and Airplay latency contributing to the total delay. Additionally, we can explore using a more efficient codec for encoding the stream before sending it through AirPlay to further reduce latency. These estimates provide a solid expectation for overall system delay as we begin developing the network communication between our core components.

These calculations also give us valuable insight into optimizing our OpenCV algorithm. Our goal is to significantly reduce processing time through algorithmic improvements and optimizations. The 1.75-second upper bound is based on the necessity of aligning frame updates with the

display's refresh cycle—if our timing is off, the stream may not display correctly, resulting in inaccurate feedback for the user, and unresponsive automatic tracking.

3. Cost and Schedule

3.1 Cost Analysis

Labor

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 = (# members)*(hours per week)*(weeks)*(cost per hour) = (3)*(12)*(8)*(49.25) = $14,184

Parts

Below is a list of parts and the cost of each part:

Part	Quantity	Cost Per ($)	Total Cost ($)	Store
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 - IHLP3232DZER3 R3M01	4	1.1	4.4	digikey
Low Vf Schottky DIode	5	0.66	3.3	digikey
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
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
2SJ652 PCH Mosfet	1	1.63	1.63	lcsc

Total Cost of all parts = $248.7

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

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 $14,184 + $841.8 + $248.7 = $15,274.50.

3.2 Schedule

Below is the tentative schedule for the project for the rest of the semester.

- *Week of 3/3; Week 7*
    - First round of PCB orders (Kadin and Henry)
    - Ordering parts (All)
    - Teamwork Evaluation I (All)
    - Design Documentation (All)
    - Custom App Development (Basic UI and Camera Control/Websockets) (Miguel)
    - Design and prepare for Breadboard Demo (All)

- *Week of 3/10; Week 8*
    - Breadboard demo /w instructor and ta (All)
    - Finalize design and parts (All)

    - Second Round of PCB orders (if needed) (Kadin and Henry)
    - Finalize physical design with Machine shop for tripod phone mount (All)

- *Week of 3/17; Week 9*
    - *Spring Break*

- *Week of 3/24; Week 10*
    - Begin PCB soldering and testing functionality (Kadin and Henry)
        - Power, Control, Motor subsystem
    - Network and App Development (ESP32 and WebSocket) (Miguel)
        - Video Feedback, iPhone App subsystem
    - Begin development of OpenCV auto-tracking algorithm (Henry and Miguel)

- *Week of 3/31; Week 11*
    - Third Round of PCB orders (if needed) (All)
    - PCB soldering and testing functionality (Kadin and Henry)
        - Power, Control, Motor subsystem
    - Network and App Development (Airplay, video propagation) (Miguel and Henry)
        - Video Feedback, iPhone App subsystem
    - Individual Progress Report (All)

- *Week of 4/7; Week 12*
    - Fourth Round of PCB orders (if needed) (Kadin and Henry)
    - PCB soldering and testing functionality (Kadin and Henry)
        - Finalizing Power, Control, and Motor subsystem
    - Test network connections and latency (Airplay, openCV, video) (Miguel and Henry)
        - Testing functionality as well as latency and delay
    - Assemble main housing for tripod and controller (All)
        - Motors, buttons for controllers, LCD screen

- *Week of 4/14; Week 13*
    - Testing hardware and software for any issues (All)
        - Ensure that all subsystems work
    - Prepare for first round of demos (All)
    - Team Contract Assessment (All)

- *Week of 4/21; Week 14*
    - Mock Demo with TAs during weekly meeting (All)
    - Begin Final Paper (All)

- *Week of 4/28; Week 15*
    - Final Demos with Instructor and TAs (All)
    - Mock Presentation with Comm and ECE TAs (All)

- *Week of 5/5; Week 16*
    - Final presentation (All)
    - Submit Final Paper (All)
    - Turn in all checked out materials and Lab Notebooks (All)

4. Ethics and Safety

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.

4.1 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.

4.2 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.

4.3 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.

4.4 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. 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