ECE 445
Fall 2025

Final Report

Glove Controlled Quad-Copter

Team Members: Atsi Gupta (atsig2), Aneesh Nagalkar (aneeshn3), Zach Greening (zg29)
TA: Wenjing Song
Professor: Cunjiang Yu

Abstract

This project presents a gesture-controlled quadcopter system that replaces traditional joystick-style controllers with an intuitive wearable glove. The glove uses an MPU-6050 IMU to detect hand orientation and motion, and an ESP32 microcontroller converts these readings into directional commands. These commands are transmitted to the drone over a low-latency Wi-Fi UDP link, enabling real-time control without additional communication hardware. Three onboard buttons provide discrete functions such as takeoff, landing, and taking a picture.

A custom PCB and 3D-printed enclosure were designed for the glove to support stable power delivery, secure sensor mounting, and comfortable wearability. The drone subsystem was assembled, repaired, and integrated with both an ESP32-based flight controller and an onboard ESP32-CAM module. The mounted camera allowed the drone to capture images during flight and validated the system's ability to support additional peripherals over the same communication platform.

Testing showed that the system met its major functional goals: gesture commands were registered with 80% accuracy, communication latency remained below 200 ms, and the camera successfully captured low-resolution images. These results demonstrate the feasibility of an ergonomic, Wi-Fi-based gesture interface with expandable sensing capabilities for small drones.

Table of Contents

1 Introduction

1.1 Motivation and Problem Statement

Flying a drone usually requires a traditional handheld controller or a smartphone interface. While these tools work well for experienced pilots, they can feel awkward, difficult to learn, and physically demanding for new users. The steep learning curve discourages beginners, and it limits how easily drones can be incorporated into education, recreation, training, or assistive applications. Standard controllers also provide very little feedback to the user and offer only basic safety options, which increases the likelihood of accidental crashes or misuse.

These limitations motivate the need for a more intuitive and accessible control method. A gesture based interface removes many of the barriers created by joysticks, buttons, and touchscreen controls. It offers a more natural interaction style that aligns with how people already communicate movement through hand and arm motion. This approach has the potential to make drone piloting easier to learn, more ergonomic, and safer for a wider range of users including hobbyists, students, and individuals who would benefit from more accessible control systems.

1.2 Objective

The goal of this project is to create a wearable gesture control glove that allows a user to fly a quadcopter through natural hand movements. An IMU inside the glove measures orientation and motion and the ESP32 sends commands wirelessly to a drone equipped with a matching ESP32 flight controller. The glove provides an intuitive and ergonomic control method that reduces the difficulty of basic flight tasks. The system includes a gesture based emergency stop for safety and an onboard camera for capturing and storing photographs. The core objective is to achieve a stable and responsive control system with low latency and reliable gesture recognition.

1.3 System Overview

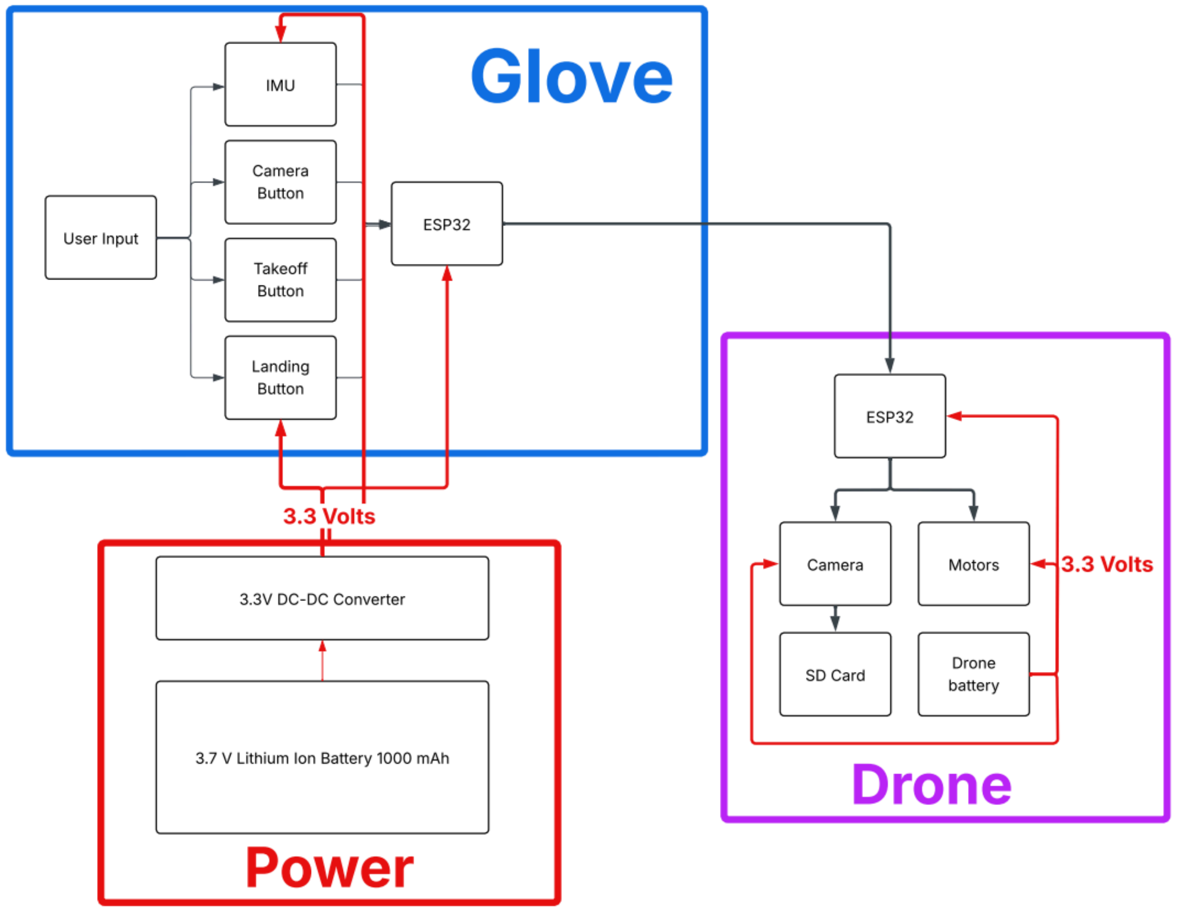


Figure 1: Block Diagram

The system consists of four main parts: the glove controller, the wireless communication link, the power system, and the drone. The glove uses an IMU and ESP32 to sense hand orientation and convert it into flight commands. These commands are transmitted to the drone, where a second ESP32 interprets them and sets motor outputs. The drone includes its own sensors, power system, and optional camera module. Together these components create a closed loop system that lets the user control the drone through natural motion.

1.4 High-Level Requirements

To demonstrate success, our project must meet the following measurable requirements:

1. The drone responds in real time to glove commands with minimal delay.
2. The buttons make the drone hover or land within 15 seconds of being pressed.
3. Directional commands (forward, back, left, right, up, down) work 80% of the time over 20 trials.

4. *(Stretch goal)* If the camera is integrated, the system should be able to store low-resolution images to the sd card.

The only changes since our design review is that we have removed one of our ‘stretch goals’ involving haptic feedback given to the user based on drone status. We decided to take this out as it is not critical to the performance of the project. Each of these requirements, aside from the camera, are essential for the project to succeed. First, we need the drone to respond quickly to glove commands. Without this, the drone would effectively be uncontrollable, as the user would not have a seamless experience. Second, the buttons need to call the correct functions for the drone. If this does not work, the user would not be able to make the drone take off and land. Lastly, and most importantly, the glove needs to successfully decipher the IMU metrics and transmit directional commands to the drone. This is the heart of the project.

2 Design

2.1 Design Details

2.1.1 Subsystem 1 – Glove (PCB + Power)

Design Description:
Our circuit (see Appendix B) was designed around a single universal 3.3 V power bus that supplies every subsystem on the glove. The entire system is powered by a 6 V 2400 mAh battery, which is stepped down to a regulated 3.3 V using an LM117C33 linear voltage regulator. Early testing revealed that the power stage was one of the most sensitive aspects of the design. While most components on the glove draw only modest current, the ESP32 can momentarily draw close to 200 mA whenever the Wi Fi antenna activates. Our original regulator was unable to provide this current reliably, causing the ESP32 to brown out and restart each time a packet was transmitted. After repeated debugging and oscilloscope measurements, we identified the power rail dip during antenna activation as the root cause of the failures. To correct this, we replaced the regulator with an LM117C, which provides a higher current rating of around 300 mA, giving the system enough headroom to handle transient surges without voltage collapse. Once upgraded, the ESP32 operated consistently with no unexpected resets, even under continuous Wi Fi transmission. In testing, the battery was able to power our circuit for far longer than an hour.

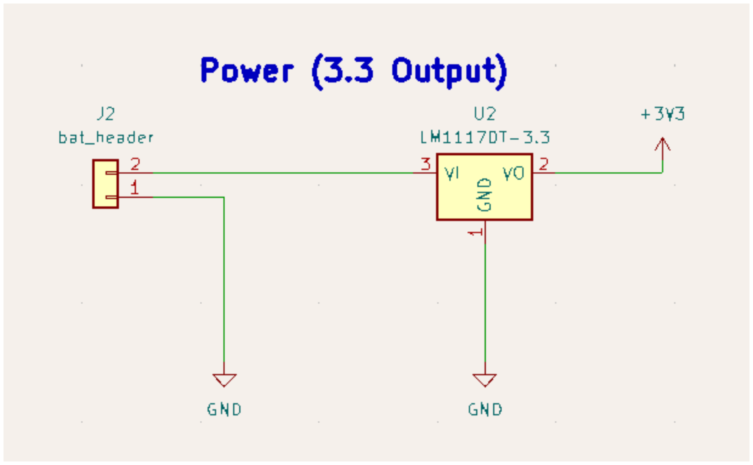


Figure 2: Power Subsystem

The PCB layout was created to keep all high current traces short and to minimize noise coupling into the MPU 6050 IMU. The IMU is connected through an 8 pin header that aligns with the onboard I²C bus and the shared 3.3 V rail. Special care was taken to route the I²C lines away from the ESP32 antenna region to avoid interference. Decoupling capacitors were placed close to both the ESP32 and the IMU to reduce voltage ripple and ensure stable sensor readings. The regulator section also includes both input and output capacitors following the manufacturer's recommendations to reduce oscillation and improve transient response. Ground planes were used on both layers of the PCB to provide stable reference points and to lower electromagnetic noise, which is particularly important for maintaining accurate IMU measurements.

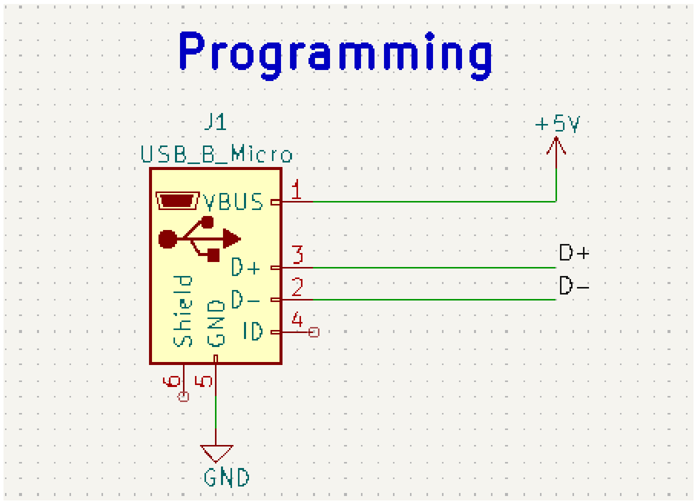


Figure 3: Programming Module

The programming of our PCB was fairly standard. It included a micro-usb header that was soldered onto the board with D+/- lines going directly to pins on the ESP32.

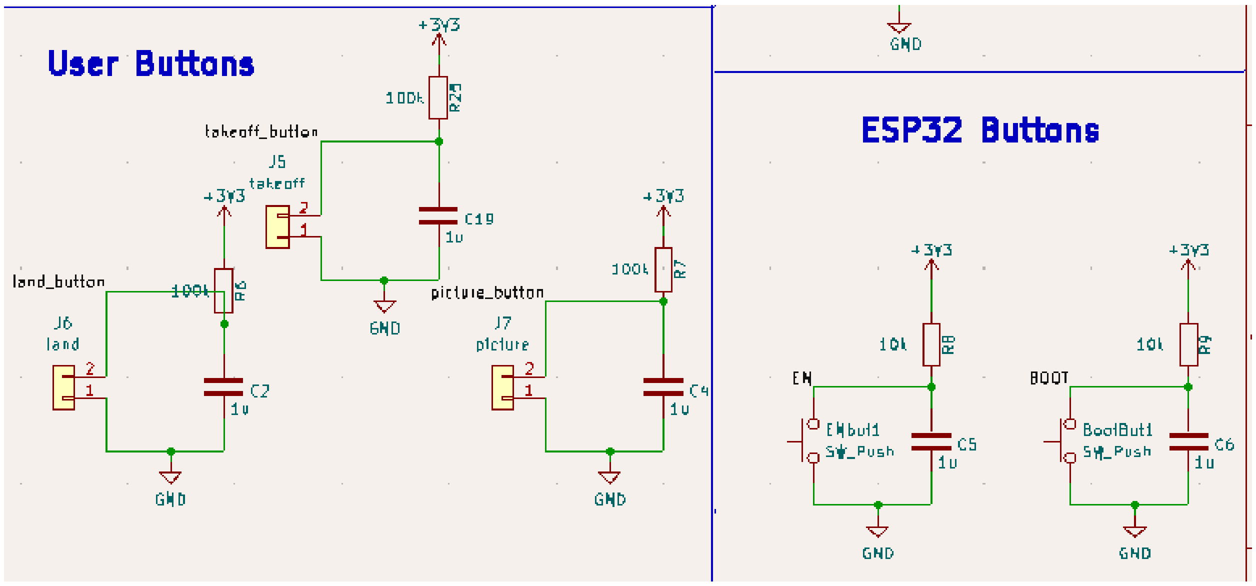


Figure 4: Debouncing Circuits

Lastly, the pcb was designed with five buttons: 3 for user input (takeoff, land, picture) and 2 for programming and reset. All of these buttons were designed with the same debouncing circuit to ensure that only one ACTIVE_LOW signal would get processed for every button press.

Future Improvements:

The power subsystem proved to be the biggest area of improvement in our project. As shown above, we have a battery going into a linear regulator. Linear regulators dissipate the overhead energy as heat, which can eventually burn the regulator out if the battery voltage is too high. As a result of this, our group burnt several regulators and also fried some components. To solve this problem, we would have liked to use a buck converter as opposed to the linear regulator, as it dissipates heat into inductors. Also, we would have put a fuse into our system after our regulator. This would have prevented components from getting fried when too much voltage was applied.

Results:

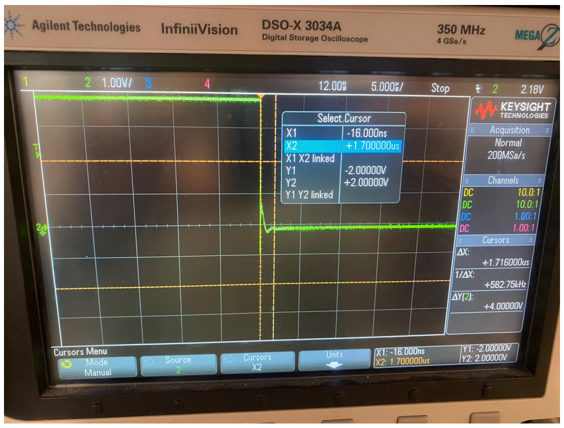


Figure 5 : Debouncing Waveform

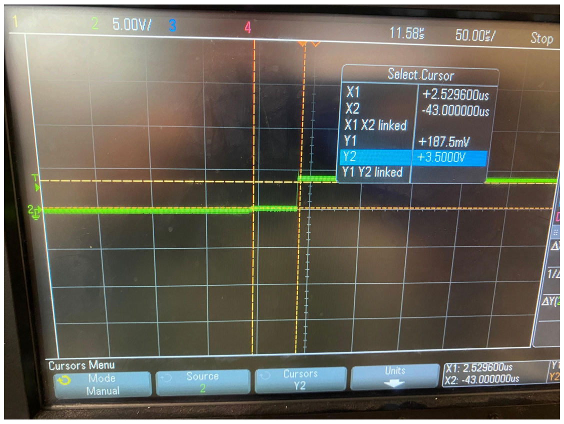


Figure 6: Stable 3.3V Waveform

As shown above on the left, the debouncing circuits that we used were a great success. The buttons are debounced in a matter of microseconds which is more than adequate for our needs. The waveform on the right was taken when the circuit was under an average load. This included sending wifi signals from the esp32 antenna, but in average increments. As shown in the waveform, our power subsystem successfully provided a stable 3.3V input to our components.

2.1.2 Subsystem 2 – IMU + Communication

To manipulate the drone's movements, the glove first needs to measure hand orientation using the MPU-6050 IMU and then transmit that processed data to the PyDrone over Wi-Fi. The IMU provides raw acceleration and gyroscope measurements, the ESP32 processes these readings into meaningful roll and pitch angles, and the built-in antenna on the ESP32 sends these commands to the drone with minimal latency.

As shown below, the MPU-6050 IMU uses an 8-pin header, with the SCL, SDA, and INT pins routed directly to the ESP32 for I²C communication and interrupt signaling. The VCC pin was supplied with a regulated 3.3 V source, while all unused pins—including AD0, AUX SDA/SCL, and the remaining configuration pins—were tied to ground to ensure stable operation. This wiring configuration minimized floating inputs, reduced noise susceptibility, and provided a clean digital interface between the IMU and the microcontroller.

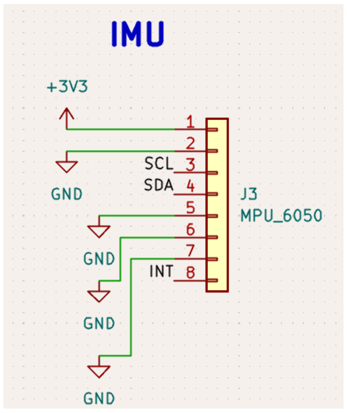
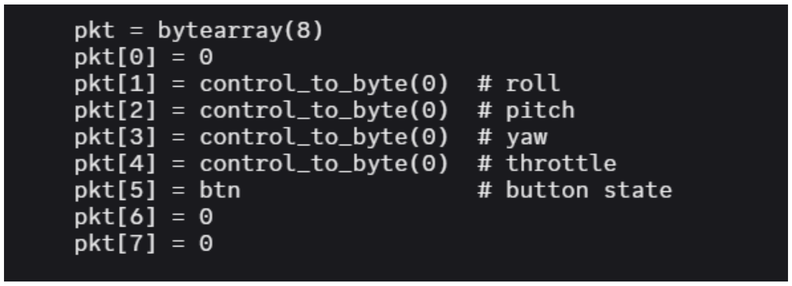


Figure 7: IMU Pinout

The onboard ESP32-S3 sampled the MPU-6050 IMU every 50 ms and used the accelerometer readings to compute the orientation of the user's hand. Before computing angles, calibration offsets were subtracted from each accelerometer axis to remove bias.

Using a flex sensor was considered as a design alternative. However, flex sensors measure finger bending rather than overall hand orientation, which did not make them a great fit for capturing the pitch and roll motions required for drone control. Also flex sensors were more likely to capture unnecessary noise, which would make it difficult to manage due to how much humans naturally shake their hand when holding it steady.

The glove controller formats those movements into an 8-byte control packet. When building that packet, each byte holds significance as shown in the figure below.



```
pkt = bytearray(8)
pkt[0] = 0
pkt[1] = control_to_byte(0)  # roll
pkt[2] = control_to_byte(0)  # pitch
pkt[3] = control_to_byte(0)  # yaw
pkt[4] = control_to_byte(0)  # throttle
pkt[5] = btn                 # button state
pkt[6] = 0
pkt[7] = 0
```

Figure 8: Control mappings

On the drone side, a MicroPython UDP server continuously listens on a fixed socket and updates motor commands immediately upon receiving packets. UDP was chosen over TCP because it provides significantly lower latency and does not require acknowledgment packets, making it better suited for real-time control loops.

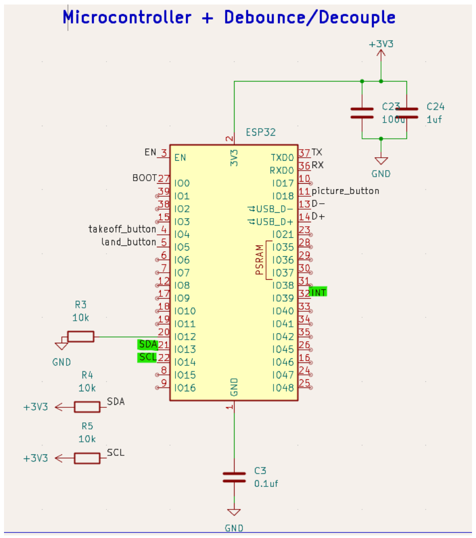


Figure 9: Microcontroller Circuit

2.1.3 Subsystem 3 – Drone + Camera

When selecting the drone to be used in our project, we had to check whether the drone had an ESP32 for communication, exposed GPIO pins to attach a camera if necessary, open source code and a reasonable price point.

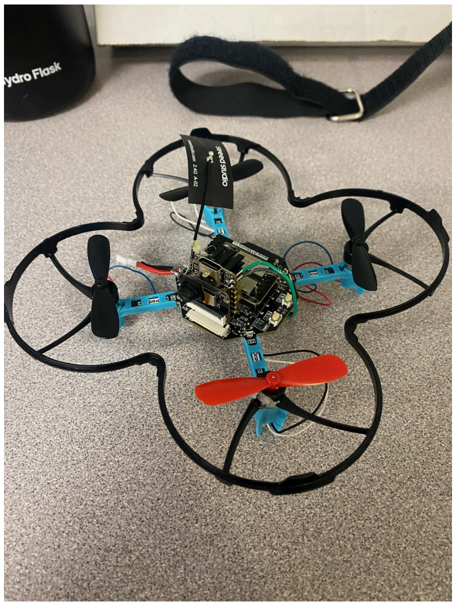


Figure 10: Pydrone with Camera Module Mounted

One challenge that arose was testing to see if the ESP32 was able to flash MicroPython code to the drone. Fortunately, we figured out that this was indeed possible because the PyDrone's ESP32-S3 could run MicroPython and still maintain a stable Wi-Fi connection, allowing us to send control packets from our glove's ESP32 in real time.

A design alternative here was to use an RF module for communication. However, the ESP32's built-in antenna could communicate over distances of 50 meters indoors and 200 meters outdoors, which is what is required from our high-level requirements.

Another challenge was finding the correct motors to use. Unfortunately, the drone was not working upon arrival. It was lifting off the ground, but violently moving in one direction. In testing, one of the motors burnt out. The replacement motors we ordered, although the same size, were not receiving enough current from the PyDrone battery to reach their rated RPM and achieve liftoff. To add on to this issue, we could not change the flight controller in the drone to tune it accordingly, as it was sent with an executable and not open source code. So, we were not able to get the drone functioning. For future work, we will design a custom flight controller for our drone. This way, we could individually tune each motor so it flies evenly. Designing our own flight controller was out of the scope for the project this semester.

For the camera module, we bought a *Seeed Studio XIAO ESP32S3 OV2640.* This module comes with an ESP32, SD card reader, and the camera module itself. We mounted this board to the drone and powered it with the drone's 3.3V power system. We programmed the board to connect to the LAN that the drone emits and listen to a button command from the glove. After receiving the photo command, it takes a low-resolution image and saves it to a micro SD card. Because this camera module works as intended, we have fulfilled the camera 'stretch goal' as outlined in the high-level requirements.

2.1.4 Subsystem 4 – Enclosure and Mechanical Design

The enclosure and mechanical subsystem underwent several iterations as the design requirements for comfort, accessibility, and testability became clearer throughout the semester. Initially, the plan was to mount the PCB directly onto a wearable glove, but this approach quickly revealed practical issues. The glove fit varied significantly between users, made rapid swapping difficult during testing, and restricted access to the PCB whenever debugging or re-flashing was required. Additionally, adding a battery housing directly onto the glove introduced too much weight for a flexible material to support. To resolve these constraints, we transitioned to an upper arm–mounted casing while keeping the IMU separately attached on the back of the hand.

Early CAD Development (Fusion Iteration 1)

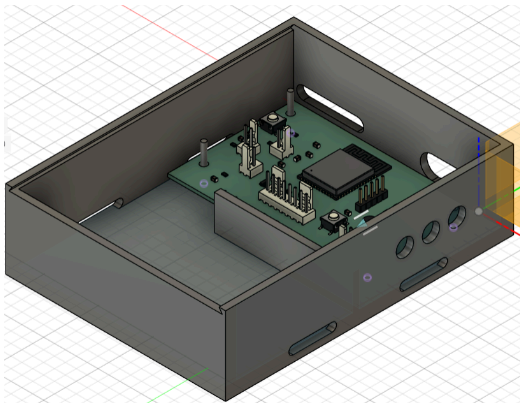


Figure 11: First iteration

The first mechanical iteration (Figure 11) focused on capturing the dimensions of the PCB and creating a simple rectangular enclosure around it. This version ensured accurate board outline, mounting hole alignment, and wall clearances but lacked internal organization and battery accommodation. Cutouts were added for cable access, and a dovetail-style sliding lid was incorporated to allow quick opening. This iteration established the baseline geometry and helped validate physical tolerances between the board and enclosure. Small issues were addressed in the lid being too large for the cutouts, the button holes being slightly too small, and the programming cutout being too high.

Refinement to Support PCB Revision 2 (Fusion Iteration 2)

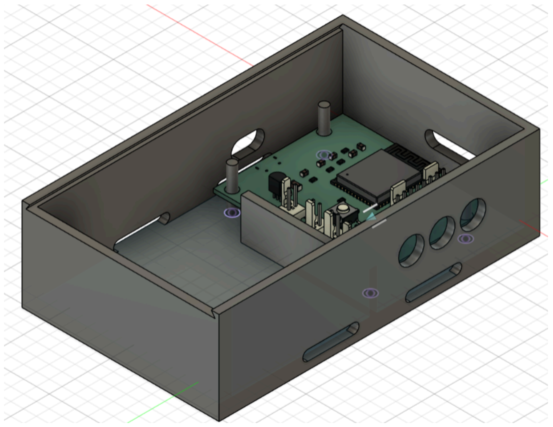


Figure 12: Second iteration

Following the fourth PCB revision, which included a new regulator layout, additional button circuitry, a smaller board footprint, and adjusted ESP32/IMU orientation, the enclosure was redesigned to match. Four vertical mounting posts were added beneath the board, with proper spacing for shock-absorbing foam. The battery compartment was reshaped to hold the 6 V battery securely without stressing wires, and dedicated wire-routing channels were introduced to prevent solder-joint strain during movement.

Final Enclosure and Printed Design (Final Printed Design)

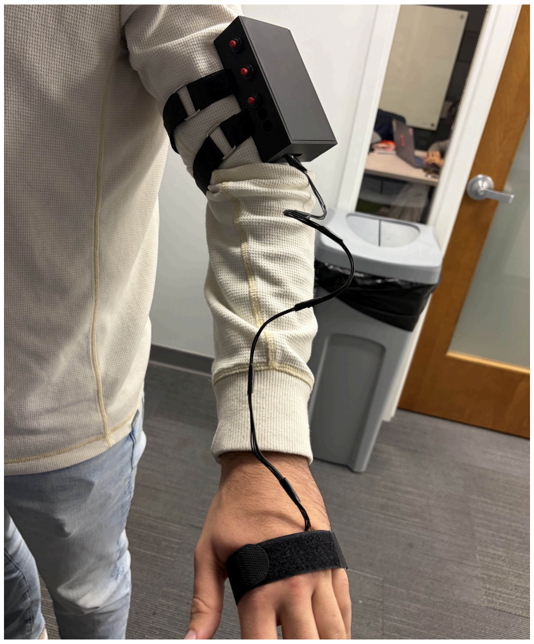


Figure 13: Final Design

The final printed enclosure integrates all functional constraints and mechanical improvements. The wall height was minimized while maintaining clearance from surface-mount components. Key design features include:

Sliding-lid mechanism:
Instead of screws or snap-fit tabs, the lid slides open on rails, allowing rapid access for debugging or reprogramming during demos. This dramatically improved test efficiency and kept the PCB accessible. This style of sliding lid is also fully 3D printable with no additional assembly.

Button recesses in the enclosure:
The three user-interface buttons (takeoff, land, picture) are accessible through recessed cylindrical holes in the side of the enclosure. The recess geometry protects the buttons against accidental presses while still allowing easy user input. This keeps the device intuitive.

Open IMU cable slot:
As we transitioned from a glove to an upper-arm mount, we added a dedicated exit port that allows the IMU cable to pass cleanly from the enclosure to the user's hand. This separation reduces interference with the ESP32 antenna and provides a more comfortable interaction for users.

Antenna clearance region:

The enclosure was designed to minimize material directly in front of the ESP32 antenna. While plastic is antenna-friendly, reducing unnecessary bulk ensures more consistent Wi-Fi signal strength during drone control.

Overall, the mechanical subsystem evolved from a simple protective case into a wearable device optimized for durability, usability, and rapid test iteration. The final design accommodates multiple users comfortably without requiring glove-size adjustment, while still allowing natural IMU placement on the hand.

Although the final enclosure design successfully met our mechanical and usability requirements, the resulting upper-arm mounted casing deviates from the original vision of a fully glove-mounted system. This redesign became necessary due to the size and weight of our PCB and battery, which were too large to be comfortably supported on the hand without compromising flexibility or stability. In future iterations, a key design alternative would be to significantly reduce both the PCB footprint and the battery form factor so the system can be returned to the glove itself.

2.1.5 Subsystem 5 - RC Car

Because the PyDrone did not fly in the end, our group put together a simple car as a proof of concept. This car could be controlled by the glove (forward, backward, left, and right) in the same manner that the drone was controlled. This worked successfully. The user was able to move their hand and see the movement reflected in the car's movement. This car was put together with an ESP32 development board, motor driver (L293D), 6V and 9V battery, two DC motors, and a car chassis. Because the car can successfully be controlled by the glove, we can conclude that we fulfilled the project's high-level requirements.

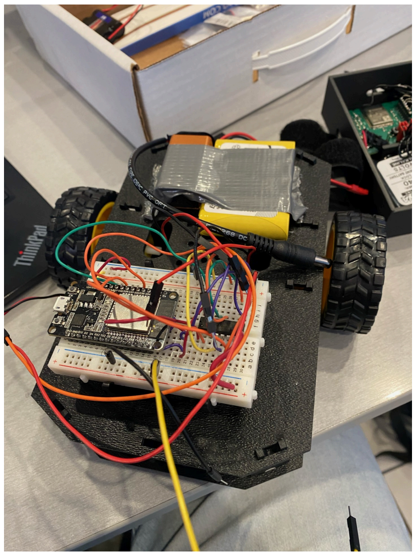


Figure14: RC Car

3 Requirements and Verifications

3.1 Complete Requirements

Requirements	Verification Procedure
1. Press registers a single event with debounce ≥10 ms	Capture button waveform + firmware logs for 20 different presses per button
2. Buttons use INPUT_PULLUP; idle = HIGH (~3.3 V), pressed = LOW (≤0.4 V)	Measure GPIO response using ESP32 output to confirm functionality
3. Ensure it can tolerate multiple button presses simultaneously in case of accident	Press multiple buttons at the same time and observe behavior (should do nothing)
4. Must be configurable to receive data via WI-FI using onboard ESP32	Verify that the drone can receive data via WI-FI from the ESP32 via debug logs from Pydrone
5. Must support real-time command reception from the glove with latency <200 ms	Measure command latency during test flights either visually or via program
6. Frequency band: 2.4 GHz Wi-Fi (802.11 b/g/n)	Stress test by introducing background Wi-Fi traffic to ensure drone commands remain prioritized.
7. Typical throughput: up to 65 Mbps, but only a few kbps required for control packets	Verify, through a different receiver (windows machine), that the glove is correctly sending UDP packets to the correct socket, with the correct wifi network.

8. Range: 30–50 m indoors, up to 100 m line-of-sight outdoors (depending on antenna quality and environment)	Verify that we can reliably control the drone from 30-50 meters away.
9. Interface: MicroPython socket libraries	All of this must be done using libraries native to MicroPython, or lightweight programs that can be flashed to the ESP32.
10. The power system must provide a stable, 3.3V output to the entire circuit.	Verify 3.3 V under idle, typical, and peak loads using oscilloscope
11. The power system must operate for more than an hour before needing to recharge.	Measure real-world runtime to ensure >1 hour per charge

Reference Appendix A for all proof of verifications and quantitative results.

4 Cost Analysis

4.1 Cost (Bill of Materials)

The total cost of the project was determined by the key components required for both the glove subsystem and the drone platform. The ESP32 Camera Board (Seeed Studio XIAO ESP32S3 OV2640), purchased from Amazon for $24.00, supported image capture on the drone. Power regulation for the glove was provided by five LM1117IMP-3.3/NOPB linear voltage regulators ordered from Digi-Key, totaling $6.35. To address issues with the defective drone hardware, we purchased a 716 coreless motor set with CW and CCW propellers from Amazon for $7.98 and later ordered an additional round of replacement motors, which added to the overall hardware expenditure. A standard 9 V alkaline battery, also from Amazon, cost $8.97 and was used during early testing phases. The primary drone platform, the PyDrone Development Kit, which includes the frame, integrated ESP32-S3 flight controller, and motors, was purchased from RCDRone for $79.00. Altogether, these purchases represent the core expenses associated with the development and testing of the gesture-controlled quadcopter system.
Total Purchased cost: $24.00 + $6.35 + $7.98 + $8.97 + $79.00 = $126.30
All resistors, capacitors, headers, JST connectors, prototyping wires, and other passive components used for the glove PCB and breadboard prototypes were obtained from the ECE Supply Center and therefore incur no additional cost.

4.2 Schedule

Located in Appendix D

5 Conclusion

5.1 Accomplishments

Our final system successfully showed the core high-level requirements. The glove reliably measured hand orientation using the MPU-6050 IMU, formatted control packets on the ESP32, and transmitted these commands over Wi-Fi with real-time responsiveness. Although the PyDrone could not stay in flight because of hardware constraints, we delivered a proof-of-concept RC car. This car responded consistently to glove inputs for forward, backward, left, and right motions.

The glove's power subsystem also met its goals: the regulator delivered a stable 3.3 V under idle, typical, and peak Wi-Fi loads. The camera stretch goal was fully achieved—our ESP32-S3 OV2640 module captured low-resolution images and saved them to an onboard SD card upon receiving a wireless command. These results demonstrate that the system meets its primary design objectives.

5.2 Uncertainties

Although the system met its major performance targets, there were some technical uncertainties.

Over long sessions, there was significant IMU drift, with an accumulated error of around 3-5 degree error per minute depending on motion. This led to some misclassification after using the glove for a long time.

Power sag was another big problem that we mitigated. With the improved LM117C regulator, rapid Wi-Fi burst activity caused transient dips of 40–60 mV, as shown in oscilloscope traces. This was fine for functionality, but it could explain occasional inconsistencies we saw in earlier prototypes.
The PyDrone experienced hardware degradation clearly shown by the failure of motors. The replacement motor we ordered seemed to operate only around 60-70% of the RPM of the other three causing uneven flight control.

These uncertainties are important engineering challenges that future iterations must address to support more demanding or safety-critical use cases.

5.3 Future Work & Design Alternatives

A smaller PCB and a lightweight Li-Po battery would allow the electronics to return to a true glove-mounted form factor, improving ergonomics and fulfilling the original design vision. Additionally, a redesigned drone frame with higher-quality brushless motors and a dedicated flight controller (e.g., STM32-based) would eliminate the RPM issues we were having.

Two important changes that could be made to the power subsystem to ensure integrity are adding a fuse and replacing the linear voltage regulator with a buck converter.

A fuse would protect the board from too much current being supplied, which would prevent accidental shorts or unexpected load conditions from damaging sensitive components such as the ESP32, IMU, or onboard buttons. During development, several regulators and components were burned because we didn’t have a fuse, which could have broken the circuit before causing all this trouble.

Replacing the LM117 linear regulator with a buck converter would improve the system even more. Linear regulators dissipate excess voltage as heat and they become inefficient when stepping down from a 6 V battery to 3.3 V. A buck converter uses an inductor to step down voltage efficiently with little heat generated. This allows it to supply higher peak currents without thermal stress.

3. Ethics and Safety

Our project follows the IEEE and ACM Codes of Ethics, prioritizing safety, honesty, and responsible design. Drones raise ethical concerns related to misuse, privacy, and airspace regulations. To address this, we limited our system to hobbyist-level drones, comply with FAA rules (flying under 400 ft in uncontrolled airspace)

Electrical Safety: All glove-mounted circuits were insulated and tested to prevent shorts. Li-Po batteries followed IEEE battery safety standards, using proper charging, protection circuitry, and enclosures to reduce risks of overheating or puncture.

Mechanical Safety: Propellers were guarded, and flights limited to controlled test areas. A gesture-based emergency shutoff ensures immediate motor disablement in unsafe conditions.

Wireless Safety: ESP32 Wi-Fi communication was tested for reliability. A fail-safe mode will cut motors if signals are lost.

Lab Safety: Work followed UIUC lab policies and OSHA guidelines, including PPE use, safe soldering practices, and risk assessments during flight tests.

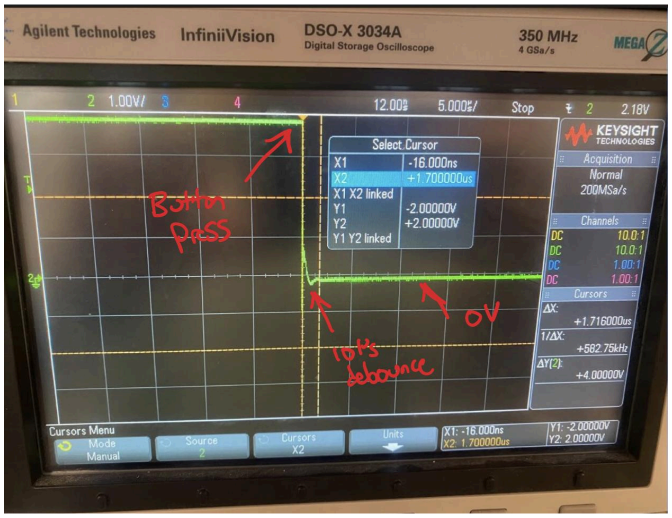
4. References

[1]Electrical Safety Standards." 2023. https://www.nfpa.org/ University of Illinois. "Division of Research Safety." 2023. https://www.drs.illinois.edu/
[2] Espressif Systems, "ESP32 Series Datasheet," Version 3.9, 2024. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
[3] M. A. Khan and S. W. Kim, "Performance analysis of 2.4 GHz wireless communication under varying environmental conditions," *IEEE Access*, vol. 9, pp. 14201–14210, 2021.
[4] IEEE Std 802.11-2020, *IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements*, IEEE, 2020.
[5] D. Halperin, B. Greenstein, A. Sheth, and D. Wetherall, "Demystifying 802.11n power consumption," in *Proc. USENIX HotPower Workshop*, Berkeley, CA, USA, 2010.
[6] Espressif Systems, "ESP-NOW: Low Power Wireless Communication Protocol," Technical Documentation, 2023. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html
[7] IEEE. "IEEE Code of Ethics." 2023. https://www.ieee.org/about/corporate/governance/p7-8.html
[8] ACM. "ACM Code of Ethics and Professional Conduct." 2023. https://www.acm.org/code-of-ethics
[9] OSHA. "Occupational Safety and Health Administration Regulations." 2023. https://www.osha.gov/
[11] Division of Research Safety." 2023. https://www.drs.illinois.edu/
[12] "ESP32-S3-WROOM-1 ESP32-S3-WROOM-1U Datasheet 2.4 GHz Wi-Fi (802.11 b/g/n) and Bluetooth ® 5 (LE) module Built around ESP32-S3 series of SoCs, Xtensa ® dual-core 32-bit LX7 microprocessor Flash up to 16 MB, PSRAM up to 8 MB 36 GPIOs, rich set of peripherals On-board PCB antenna." Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

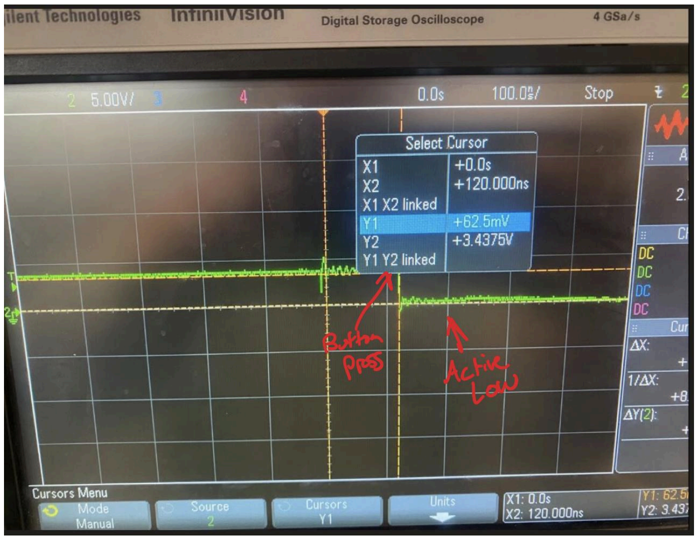
5. Appendices

Appendix A – Requirement and Verification Evidence

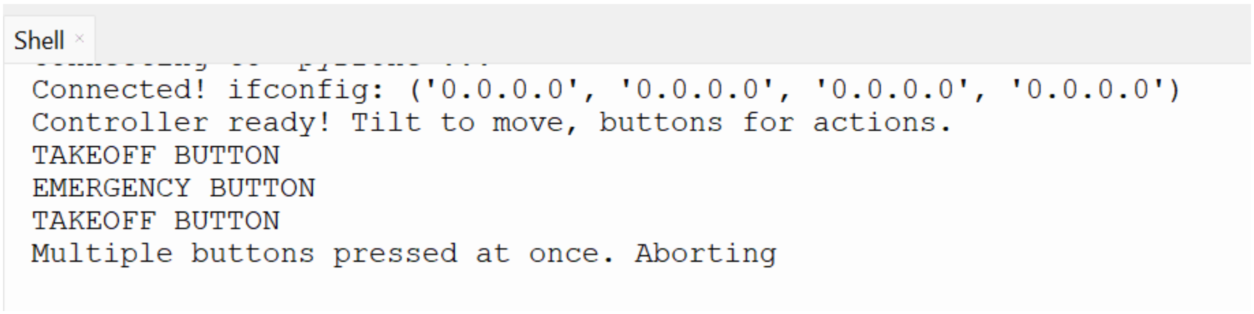
Requirement 1:



Requirement 2:



Requirement 3:

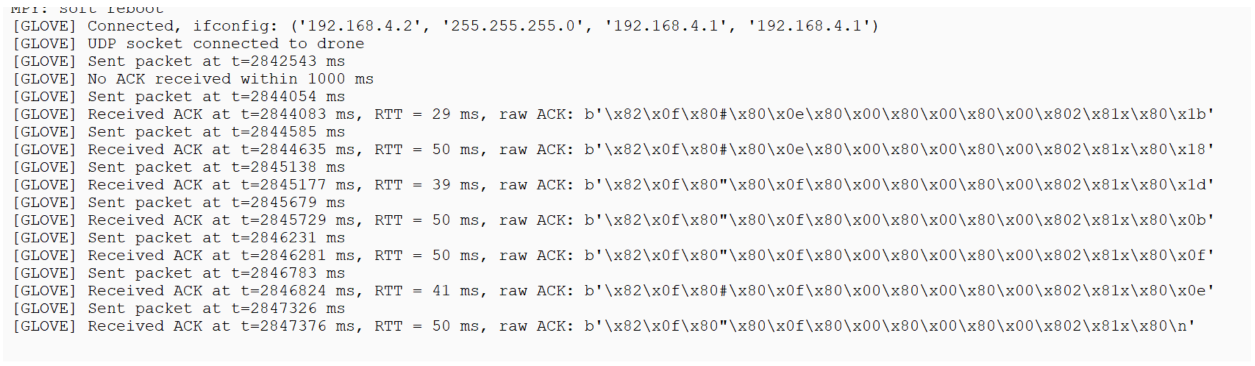


```
Connected! ifconfig: ('0.0.0.0', '0.0.0.0', '0.0.0.0', '0.0.0.0')
Controller ready! Tilt to move, buttons for actions.
TAKEOFF BUTTON
EMERGENCY BUTTON
TAKEOFF BUTTON
Multiple buttons pressed at once. Aborting
```

1. **Must be configurable to receive data via WIFI using onboard ESP32**

Refer to Verification 7 to prove this works

2. **Must support real-time command reception from the glove with latency <200 ms**

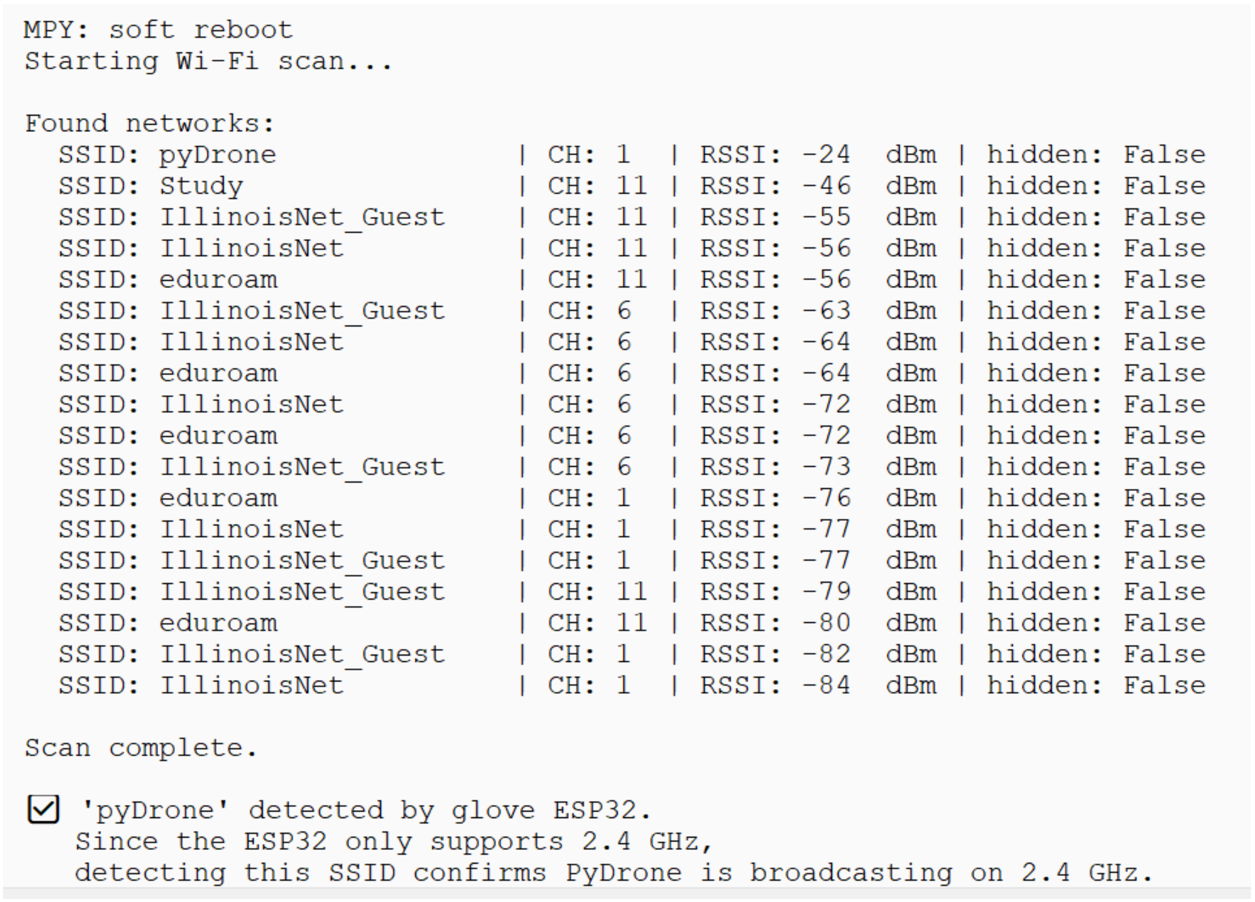


```
MPY: soft reboot
[GLOVE] Connected, ifconfig: ('192.168.4.2', '255.255.255.0', '192.168.4.1', '192.168.4.1')
[GLOVE] UDP socket connected to drone
[GLOVE] Sent packet at t=2842543 ms
[GLOVE] No ACK received within 1000 ms
[GLOVE] Sent packet at t=2844054 ms
[GLOVE] Received ACK at t=2844083 ms, RTT = 29 ms, raw ACK: b'\x82\x0f\x80#\x80\x0e\x80\x00\x80\x00\x80\x00\x802\x81x\x80\x1b'
[GLOVE] Sent packet at t=2844585 ms
[GLOVE] Received ACK at t=2844635 ms, RTT = 50 ms, raw ACK: b'\x82\x0f\x80#\x80\x0e\x80\x00\x80\x00\x80\x00\x802\x81x\x80\x18'
[GLOVE] Sent packet at t=2845138 ms
[GLOVE] Received ACK at t=2845177 ms, RTT = 39 ms, raw ACK: b'\x82\x0f\x80"\x80\x0f\x80\x00\x80\x00\x80\x00\x802\x81x\x80\x1d'
[GLOVE] Sent packet at t=2845679 ms
[GLOVE] Received ACK at t=2845729 ms, RTT = 50 ms, raw ACK: b'\x82\x0f\x80"\x80\x0f\x80\x00\x80\x00\x80\x00\x802\x81x\x80\x0b'
[GLOVE] Sent packet at t=2846231 ms
[GLOVE] Received ACK at t=2846281 ms, RTT = 50 ms, raw ACK: b'\x82\x0f\x80"\x80\x0f\x80\x00\x80\x00\x80\x00\x802\x81x\x80\x0f'
[GLOVE] Sent packet at t=2846783 ms
[GLOVE] Received ACK at t=2846824 ms, RTT = 41 ms, raw ACK: b'\x82\x0f\x80#\x80\x0f\x80\x00\x80\x00\x80\x00\x802\x81x\x80\x0e'
[GLOVE] Sent packet at t=2847326 ms
[GLOVE] Received ACK at t=2847376 ms, RTT = 50 ms, raw ACK: b'\x82\x0f\x80"\x80\x0f\x80\x00\x80\x00\x80\x00\x802\x81x\x80\n'
```

RTT (Round-Trip Time) is clearly below 200 ms here

3. **Frequency band: 2.4 GHz Wi-Fi (802.11 b/g/n)**

Run the code found in appendix



```
MPY: soft reboot
Starting Wi-Fi scan...

Found networks:
  SSID: pyDrone                | CH: 1  | RSSI: -24  dBm | hidden: False
  SSID: Study                  | CH: 11 | RSSI: -46  dBm | hidden: False
  SSID: IllinoisNet_Guest      | CH: 11 | RSSI: -55  dBm | hidden: False
  SSID: IllinoisNet            | CH: 11 | RSSI: -56  dBm | hidden: False
  SSID: eduroam                | CH: 11 | RSSI: -56  dBm | hidden: False
  SSID: IllinoisNet_Guest      | CH: 6  | RSSI: -63  dBm | hidden: False
  SSID: IllinoisNet            | CH: 6  | RSSI: -64  dBm | hidden: False
  SSID: eduroam                | CH: 6  | RSSI: -64  dBm | hidden: False
  SSID: IllinoisNet            | CH: 6  | RSSI: -72  dBm | hidden: False
  SSID: eduroam                | CH: 6  | RSSI: -72  dBm | hidden: False
  SSID: IllinoisNet_Guest      | CH: 6  | RSSI: -73  dBm | hidden: False
  SSID: eduroam                | CH: 1  | RSSI: -76  dBm | hidden: False
  SSID: IllinoisNet            | CH: 1  | RSSI: -77  dBm | hidden: False
  SSID: IllinoisNet_Guest      | CH: 1  | RSSI: -77  dBm | hidden: False
  SSID: IllinoisNet_Guest      | CH: 11 | RSSI: -79  dBm | hidden: False
  SSID: eduroam                | CH: 11 | RSSI: -80  dBm | hidden: False
  SSID: IllinoisNet_Guest      | CH: 1  | RSSI: -82  dBm | hidden: False
  SSID: IllinoisNet            | CH: 1  | RSSI: -84  dBm | hidden: False

Scan complete.

☑ 'pyDrone' detected by glove ESP32.
   Since the ESP32 only supports 2.4 GHz,
   detecting this SSID confirms PyDrone is broadcasting on 2.4 GHz.
```

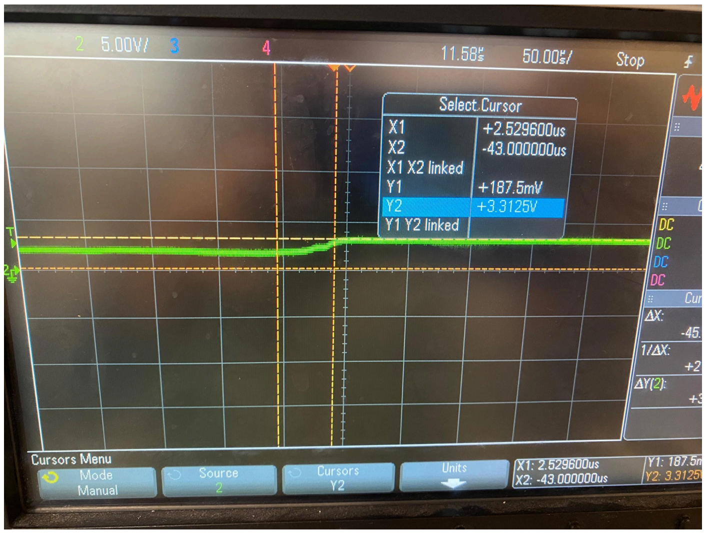
4. **Typical throughput: up to 65 Mbps, but only a few kbps required for control packets**

```
MPY: soft reboot
Connecting to 'pyDrone'...
Connected! ifconfig: ('192.168.4.2', '255.255.255.0', '192.168.4.1', '192.168.4.1')
Controller ready! TAKEOFF arms motors. LAND/EMERGENCY turn motors off.
Control packet payload size = 8 bytes.
[THROUGHPUT] interval ~1010 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1007 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1007 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1009 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1008 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1007 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1011 ms | packets: 21 | bytes: 168 | ~166 B/s (0.162 kB/s)
[THROUGHPUT] interval ~1008 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
[THROUGHPUT] interval ~1007 ms | packets: 20 | bytes: 160 | ~158 B/s (0.154 kB/s)
```

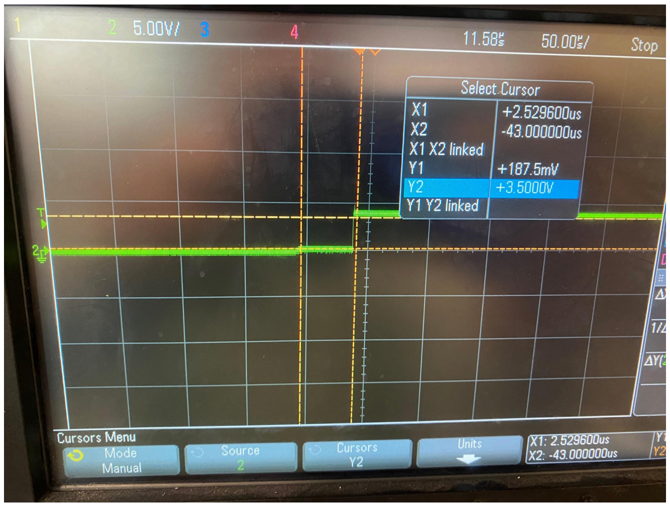
The glove's control traffic is only around 1–5 kB/s, while the Wi-Fi link supports up to 65 Mbps (≈8125 kB/s). This means the actual data rate is thousands of times smaller than the available capacity, providing a very large safety margin. Because the control packets require so little bandwidth, the Wi-Fi link can deliver them with very low latency and without risk of congestion or packet loss. Therefore, the measured throughput far below the maximum capacity demonstrates that the wireless link easily satisfies the communication requirement.

5. Recorded video
6. All execution of code was done through micropython firmware
7.

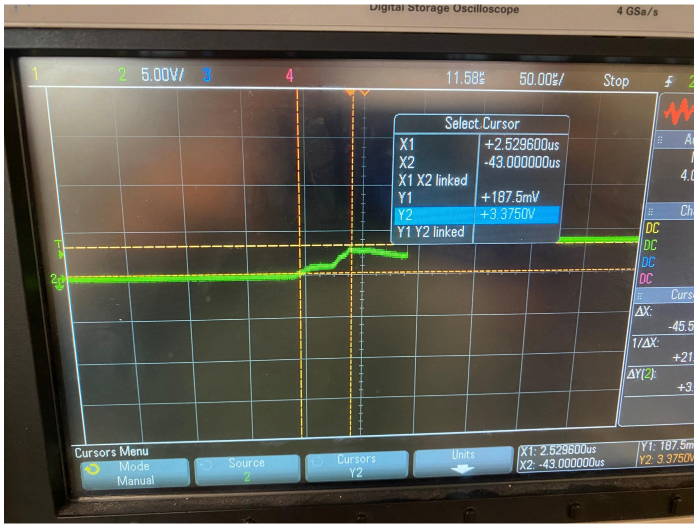
Below is proof of stable 3.3V at average load



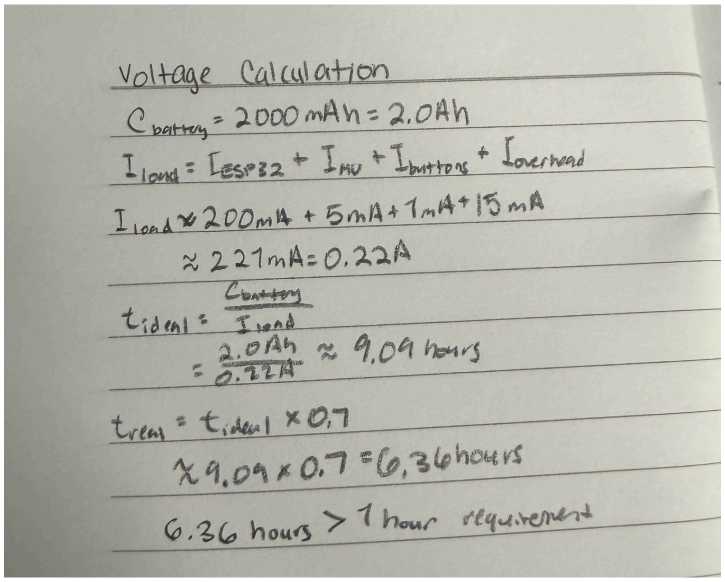
Below is proof of 3.3V at idle load



Below is proof of 3.3V at peak load



Requirement 11:



Voltage Calculation

$C_{battery} = 2000\,mAh = 2.0\,Ah$

$I_{load} = I_{ESP32} + I_{IMU} + I_{buttons} + I_{overhead}$

$I_{load} \approx 200mA + 5mA + 1mA + 15mA$

$\approx 221mA = 0.22A$

$t_{ideal} = \frac{C_{battery}}{I_{load}}$

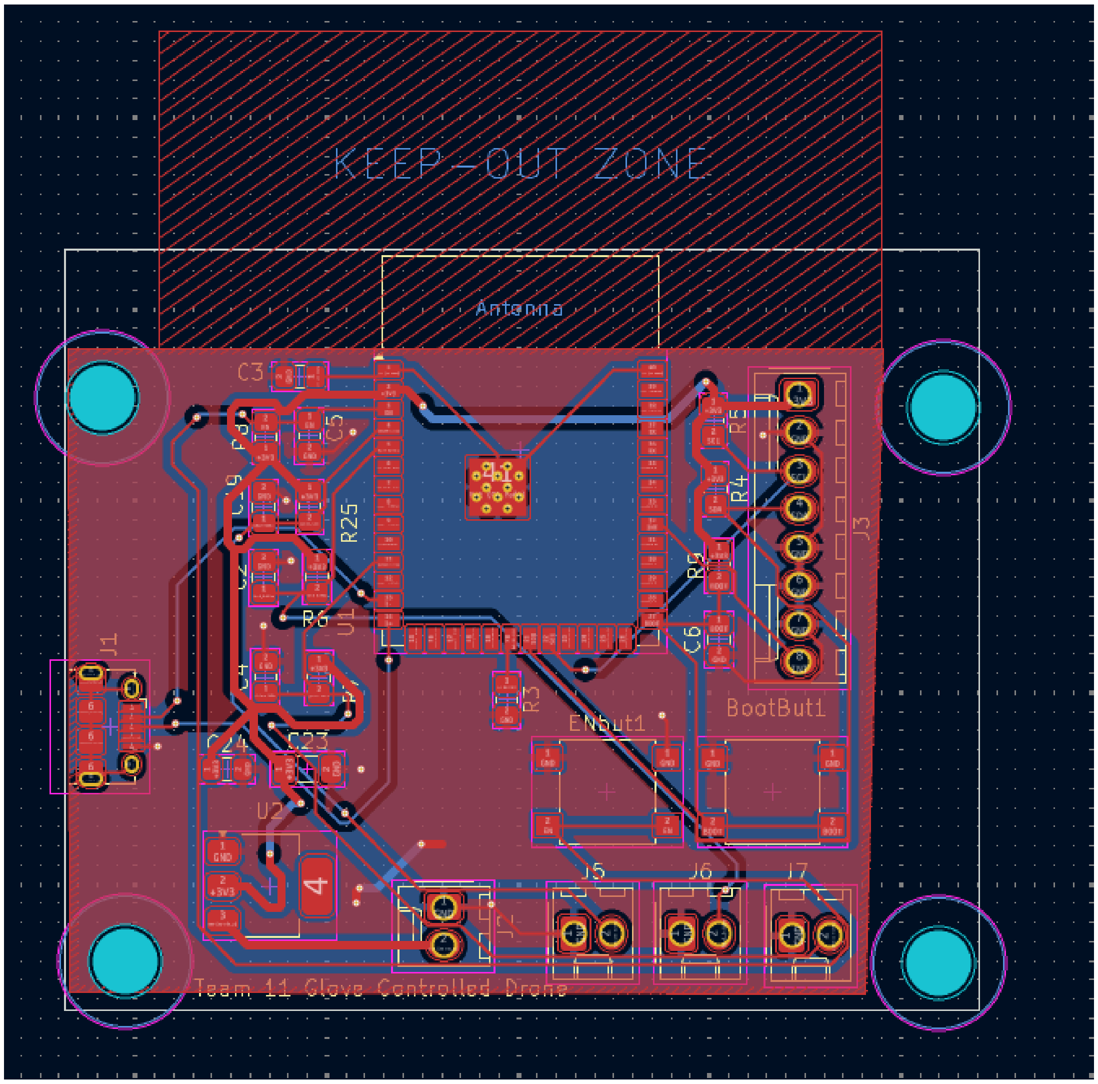
$= \frac{2.0Ah}{0.22A} \approx 9.09$ hours

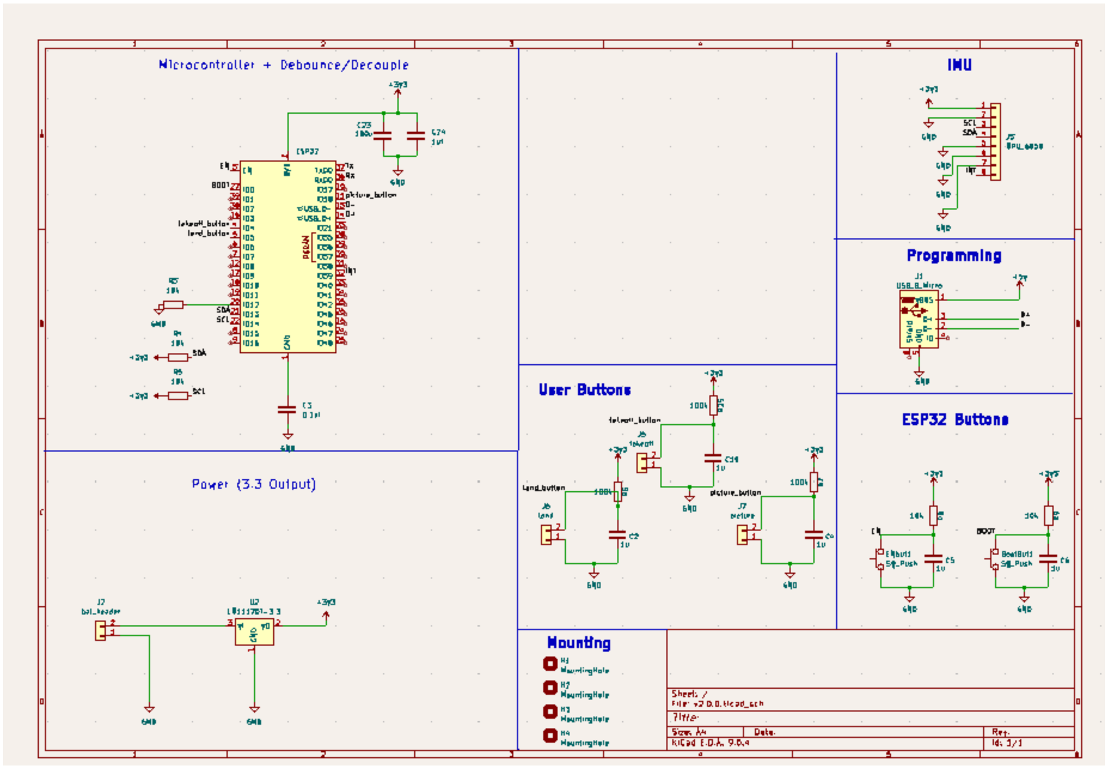
$t_{real} = t_{ideal} \times 0.7$

$\approx 9.09 \times 0.7 = 6.36$ hours

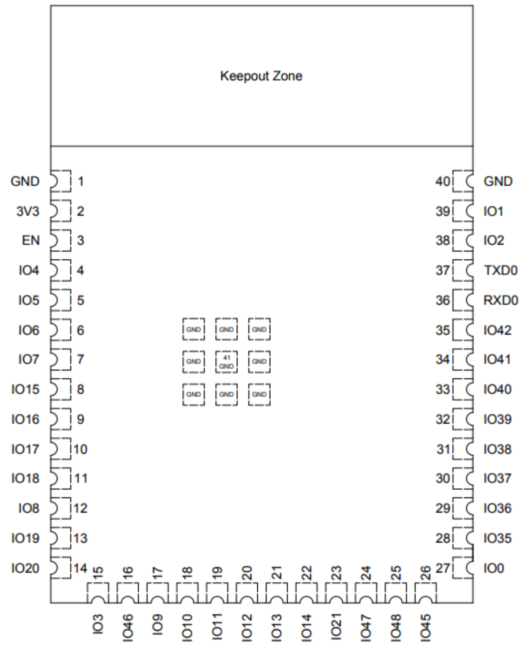
6.36 hours > 1 hour requirement

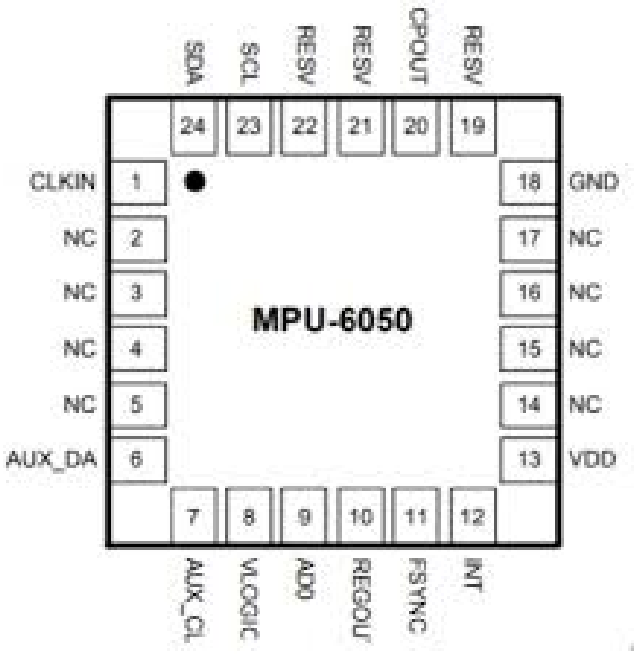
Appendix B – Full Schematics





Appendix C – Component Pinouts





Appendix D – Full Schedule

Week of 9/16

- Atsi: Research IMUs, ESP32 options, drone platforms; begin component list.
- Zach: Begin drafting initial KiCad schematic.
- Aneesh: Set up ESP32 environment; test MicroPython basics.

Week of 9/23

- Atsi: Order remaining breadboard & PCB components.
- Zach: Complete first-pass KiCad schematic for PCB Rev 1.
- Aneesh: Write ESP32–IMU test code; verify I^2C detection and raw data.

Week of 9/30

- All: Attend 10/3 PCB Review and document feedback.
- Zach + Aneesh: Add buttons to breadboard; program INPUT_PULLUP logic.
- Atsi: Write Design Document sections.

Week of 10/7

- All: Submit teamwork evaluation; finalize PCB edits; submit PCB order (10/6).
- All: Prepare and run Breadboard Demo #1; incorporate TA feedback.
- Atsi: Continue writing Design Document.
- Zach: Schematic & routing cleanup.
- Aneesh: IMU calibration + early gesture mapping tests.

Week of 10/14

- Zach: Apply edits for PCB Round 2.
- Atsi: Assemble PyDrone; test flight controller behavior.
- Aneesh: Begin glove-to-drone Wi-Fi communication testing.

Week of 10/21

- Zach: Document Wi-Fi packet testing results.
- Atsi + Aneesh: Update breadboard wiring per Demo #1 feedback.
- All: Prepare for Demo #2 (10/28).

Week of 10/28

- All: Conduct Breadboard Demo #2; log verification results.
- All: Start Final Paper introduction.
- All: Begin Final Presentation slides.
- Atsi: Begin CAD modeling of enclosure (Fusion Iteration 1).

Week of 11/4

- All: Order PCB Round 3 if needed.
- Atsi: Attempt drone repairs; test replacement motors.
- Zach: Refine regulator footprint; confirm capacitor sizing for ESP32 Wi-Fi bursts.
- Aneesh: Debug Wi-Fi packet loss; improve IMU smoothing.
- All: Update lab notebook with debugging work.

Week of 11/11

- All: Order PCB Round 4 if necessary.

- Atsi: Validate glove–drone communication on updated hardware; CAD Iteration 2.
- Zach: Assemble PCB Rev 2; perform continuity checks and flashing tests.
- Aneesh: Integrate smoothing algorithm; refine gesture thresholds.
- All: Update Final Paper notes.

Week of 11/18

- All: Run Mock Demo (11/18); gather TA feedback.
- Atsi: Print final enclosure; finalize battery compartment + routing.
- Zach: Test power stability under Wi-Fi load; measure 3.3V ripple.
- Aneesh: Finalize firmware + UDP packet structure.
- All: Update Final Presentation slides.

Week of 11/25

- Atsi: Drone repair attempt #2; finalize enclosure fit & mounting.
- Zach: Final PCB assembly and soldering; assist drone debugging.
- Aneesh: Finish gesture logic; test IMU drift mitigation; tighten communication loop.
- All: Expand Results & Verification sections of Final Paper.

Week of 12/2

- All: Final Demo (12/1).
- All: Final Presentation (12/8).
- Atsi: Complete enclosure documentation & cost analysis.
- Zach: Prepare schematic + PCB layout for appendices.
- Aneesh: Finalize firmware appendix + communication verification figures.
- All: Submit Final Paper (12/10) & Lab Notebook (12/11).