University of California, Santa Cruz

Long Flight Time Buoyant Drone Final Report

Abstract

Research data collection with drones has become invaluable and enabled researchers to collect more accurate data faster than ever before, even in areas that could not normally be accessed. Unmanned multicopters are used to carry payloads and collect data, but they are limited to short flight times. The USGS collects magnetometer data with multicopters and are limited to 15-minute flights, limiting the amount of data they can collect on a trip. We have partnered with Jonathan Glenn of the USGS to identify the needs of researchers and have worked with them to develop a drone with an extended flight time. Since drones expend most of their energy counteracting their own weight, we implemented a helium lift bag to reduce the drone's and payload effective weight to 4N and increase flight time. The primary project goals developed with USGS were a minimum drone flight time of 30 minutes, a reduced magnetic field to decrease magnetometer interference to less than 10nT, a 20-mph minimum airspeed, and legal compliance with the FFA, with both Remote Control and Autonomous Control functionality. Magnetic interference was never verified, legal compliance is achievable, but needs a completed drone to fully apply, and Remote Control and some autonomous functions, but not the full autonomous system, were verified in MATLAB. In the end, due to manufacturing errors the drone test flight never made it off the ground, but power draw analysis verifies a 40-minute flight time can be reached with a 20-mph airspeed. The project was unsuccessful in meeting the USGS's needs, but simulation and power draw analysis show the design is possible and worth pursuing.

Dylan Harootunian
Chin Ming Ryan Wong
George Hernandez
Jeremy Germenis
Leonid Shuster
Isaac Szu

June 7th, 2021

Table of Contents

Chapter 1: Introduction **4**
1.1 Project Background and Motivation 4
1.2 Current Solutions-DJI Matrice 600 Pro 6
1.3 The Need for a Better Solution 9
1.4 What to Expect and Report Layout 9
Chapter 2: Design Considerations of a Buoyant Drone **11**
2.1 Project Goals and Requirements 11
2.2 Physics of a Buoyant Drone 14
2.3 General Design Overview 18
2.4 Conclusion 22
Chapter 3: Envelope and Drone Frame Design **23**
3.1 Barone's Mechanical System. 23
3.2 Fabrication 29
3.3 Mechanical System Verification. 33
3.4 Conclusion 40
Chapter 4: Propulsion Design and Actuator Interface **41**
4.1 Parts Selection of the Propulsion System 41
4.2 Propulsion System Implementation 44
4.3 Propulsion System Verification 50
4.4 Conclusion 51
Chapter 5: Sensor Array, State Estimation, and PCB Interface **53**
5.1 Sensor Array 53
5.2 State Estimation 73
5.3 PCB Interface 75
5.4 Conclusion 99
Chapter 6: System Behavioral Design **101**
6.1 Discussion of Drone Operating Environments 101
6.2 Day in the Life of The Barone 101
6.3 Detailed State Machine Design 103
6.4 Startup Procedures 105
6.5 Linearization of Equations of Motion 105
6.6 Open Loop Remote Control Design 109

6.7 Auxiliary Functions 112
6.8 Closed Loop Remote Control Design 118
6.9 Autonomous Design 124
6.10 Error Identification and Response 124
6.11 Conclusion 125

Chapter 7: Power Analysis and Flight Time 126
7.1 System Components 126
7.2 Powering & Distribution 140
7.3 Conclusion 144

Chapter 8: Full System Simulation and Validations of Drone Design 145
8.1 Choosing a Full System Simulation Environment for the Buoyant Drone 145
8.2 Necessary Building Blocks for a Practical Simulation Environment 146
8.3 Open Loop Remote Control Implementation 148
8.4 Sensor Array Implementation 151
8.5 Closed Loop Remote Control Implementation 152
8.6 Verification of the Prototype's Fabricated Dimensions 156
8.7 Conclusion 158

Chapter 9: Testing in a Controlled Environment 159
9.1 Tests to Conduct 159
9.2 Procedures and Results 160
9.3 Unconducted Test Procedures 169
9.4 Conclusion 170

Chapter 10: Testing in an Uncontrolled Environment 172
10.1 Variable Environment Tests 172
10.2 Conclusion 174

Chapter 11: Legal and Safety Requirements 175
11.1 Legal Requirements 175
11.2 Safety Requirements 177
11.3 Conclusion 179

Chapter 12: Costs and Funding 181
12.1 System Costs 181
12.2 Project Funding 182
12.3 Conclusion 183

Chapter 13: Conclusions and Next Steps 184

13.1 Review of Technical Requirements 184
13.2 Project Conclusions 186
13.3 Lessons Learned 186
13.4 Personal Reflections 188
13.5 Team Reflection 193
13.6 Next Steps 193
Chapter 14: Appendix 197
Bibliography 199

Chapter 1: Introduction

1.1 Project Background and Motivation

1.1.1 USGS Goals

Researchers at the United States Geological Survey (USGS) collect data of the Earth's Earth's Earth's magnetic field over an area, and use magnetic anomalies in order to identify underground geological formation such as geysers, rivers, and mineral deposits. The area covered in each survey is large and not always easy to travel across, so the USGS has adopted the use of drones to carry a magnetometer payload to collect data[2]. Jonathan Glenn, a researcher with the USGS, conducts magnetic field surveys with drones and has provided the background information for the limitations and difficulties of using drones as data collection tools. The primary issue Glenn conveyed was the limited flight time; the current drone setup Glenn uses to collect data is limited to 15-minute flights and the drone must make 7 trips to cover the usual area they survey in a day[12]. Even with limited flight time, data collection with drones is invaluable.

In 2010 magnetic data was collected at Yellowstone National Park using both aerial and land based collection methods, around geologic anomalies such as Lone Star Geyser. This survey provided significant evidence that "a significant decrease of the substratum total magnetization is observed within altered zones"[1]. It finds that these anomalies can be used to map areas of hydrothermal alteration by searching areas characterized by this lower magnetism. This survey was able to provide data on different types of anomalies that would allow researchers to determine characteristics of that anomaly such as whether it is liquid dominated or its depth. Another survey in 2012 by USGS built on this by using UAS to reveal a more than "35km long linear, intra basin magnetic high"[2]. The unprecedented level of detail from this survey allowed for the discovery of several major hot springs that could be used for geothermal research and energy. "These findings could never have been substantiated by ground based data"[2].

The USGS notes that beyond providing more substantive evidence, surveys conducted by UAS had a multitude of benefits. UAS surveys are highly adaptable, compared to high resolution commercial surveys, that are "relatively inflexible to the need to change survey specifications that may arise as data [is] collected"[2]. UAS surveys have additional safety benefits for surveys that require low altitude flight paths that would pose risks to pilots of manned aerial vehicles.

Due to the successes of USGS experiments with UAS based magnetometer data collection, Jonathan Glen has expressed interest in further developments in technology that could benefit this method of data collection.

Jonathan Glenn and the USGS are only one example of the effects drone technology has on the research community. Drone technology is superior to older data collection methods in safety, sensor accuracy, and adaptability, and cannot be replaced, but drones have their limitations and those limitations need to be improved to further help researchers. We worked with Jonathan Glenn to define the needs of researchers based on his specific needs as our first test case. Jonathan Glenn is a potential client who has agreed to pay for a system that meets his needs, but also, he helps provide detailed information on the needs of researchers who use drones as a whole.

1.1.2 Advantages of Drone Data Collection

Drones have several major advantages over other methods of data collection, including safety, accessibility, sensor accuracy, and real-time monitoring.

Drones help to collect data when it is dangerous for humans to do so. Many tasks performed by people can be hazardous and even result in death, 5333 deaths in the United States during 2019 alone[8]. Drones help mitigate the risks by removing the operators away from potential danger; although drones can not be applied in every situation, accepting their use can help protect workers from injury and death. Drones are already being employed by businesses, and can perform thousands of dangerous tasks even in a single company[9]. Drone safety is only the beginning to their advantages.

Drones also increase accessibility to places that can be hard to reach with other data collection methods. Limited accessibility applies to several situations such as the tops of mountains, the inside of a volcanic eruption site[10] or near people. Other forms of aircraft, such as planes or helicopters, could be used to access these areas, but a lot of time and resources have to be allocated in order to use them, and there are potential issues of safety for the pilots and nearby people. With the use of a drone, people can access places in the form of an expendable, small, and easily controllable flying aircraft, and this is imperative when wanting to investigate something close to the ground, where manned aircraft cannot safely go.

Drones can also be used for real-time data monitoring. With the attachment of sensors onto a drone, a drone can easily capture the scene of an environment and record it for further analysis. These drones are even more useful when they provide live data back to the user, allowing the person to be far away from the area allowing a much faster means of recording data, by decreasing data collection times by improving accessibility through drone flight, in some cases reducing days or months of data collection to a few hours[11]. In turn, this allows for a more efficient workflow. Workplaces have used drones for this reason as well, and have seen substantial improvement in productivity.

1.1.3 Disadvantages of Drone Data Collection

Drones have a limited weight capacity, so a limited amount of onboard power is available, limiting drone flight time. Propulsion systems and wireless communication systems are two of the largest power draws on a drone, so they should be optimized to increase flight time as much as possible[5]. The propulsion system must be as energy efficient as possible to carry the drone over large distances and maintain flight as long as possible. Given a drone's propulsion and electrical system, the maximum flight time will decrease as the drone carries heavier loads since more energy is required to maintain altitude. The communications system must be optimized due to its large power costs to transmit data, especially as the drone travels further from the user and further increases with increased amounts of data transmitted.

The USGS drone setup is designed to carry a magnetometer payload, which measures the magnetic field around it; precautions need to be made to reduce the magnetic interference. The greater the generated magnetic field of the motors is and the closer the motors are to the magnetometer, the greater the interference will be causing an error in the magnetic field readings.

Drones are also vulnerable to outside forces such as drag, and this is heavily dependent on the design . Given that the drone must be light to be able to have the largest possible flight time, this also must be balanced with the wind that may push it around if it is too light. The aerodynamics of the drone may also have a dampening effect on the air resistance. Another cause of a collision may be caused by a user error with the controls. If a drone's rotors are not sufficiently protected, any obstructions or collisions involving the rotors may cause the drone to cease to function and crash. A lighter drone will cause crashes to be more prevalent due to the drag force because of wind causing the drone to deviate from its desired path[6].

1.2 Current Solutions-DJI Matrice 600 Pro

The USGS currently uses the DJI Matrice 600 Pro for magnetometer data collection. The DJI Matrice is a hexacopter and is a drone designed to carry heavy payloads, up to 6 kg. The drone primarily suffers from a short flight time of 15 minutes\, due to the mass of the electronics, frame, and payload. To carry the large payload, the drone needs a powerful electrical system and large batteries, massively increasing the size of the drone, and a heavy frame to support the electronics, payload, and propulsion forces.

Table 1.1
DJI E2000 Powertrain Specifications [7]

Part Name	Use	Number	Power	Mass	Dimensions	Total mass
DJI 6010	Propulsion motor	6	5100 g (max thrust)	230g	66.7mm Diameter X 29.2mm	1380g
DJI 2170R	Propeller	6	N/A	58g	21 in Diameter	348g
ESC (Unnamed)	Motor Control	6	N/A	90g (with cables)	85 mm X 44 mm X 18mm	540g
TB27S	Battery	6	99.9 Wh (Storage)	595g	Not Listed	3570g
Total mass						5838 g

Table 1.2
DJI Matrice 600 Pro Specifications [7]

Specification	Description
Assembled Drone Dimensions	1668 mm X 1518 mm X 727 mm
Storage Drone Dimensions	437 mm X 402mm X 553 mm
Total Mass	9.5 kg
Wind Resistance	8 m/s
Max Speed (no wind)	40 mph
Max Payload	6 kg
USGS Payload (Info Given by Jonathan Glenn)	1 kg

Table 1.1 summarizes the specifications of the DJI E2000 powertrain system, the onboard propulsion system for the DJI Matrice Pro. The propulsion system alone weighs almost 6 kg of the drone mass. The total max thrust is 30.6 kg in order to move the drone and payload of up to 6 kg, and this consumes large amounts of power, six 100Wh batteries per flight.

Table 1.2 looks at several important system wide performance specifications for the drone. The overall drone size during flight is fairly large, but the storage size is far smaller, at 26.1%, 26.5%, and 76.1% of the assembled size for length, width and height respectively. The total drone mass is also 9.5 kg, sixty-percent of which is dedicated to the propulsion system.

The drone also has the ability to operate in wind conditions of up to 8 m/s, and has a top speed of 40 mph in no wind conditions. These values are provided by the website for the drone, but these values can change depending on the wind resistance and moments created by the payload mass, position, and drag.

The drone itself is very impressive, boasting a powerful and fast propulsion system that can support large payloads, however, there are several shortcomings. When used by the USGS for collecting magnetometer data, it can only make fifteen minute long flights. This is largely due to the mass and forces of the propulsion system. A large portion of the energy expended during flight is just for counteracting its own flight mass. The drone is also designed to carry payloads of up to 6 kg, but the magnetometer only weighs 1 kg, so the drone isn't designed to perform most efficiently for the magnetometer mass.

The drone is also very expensive, costing $6,599.00 at dronenerds.com. This is excluding spare batteries listed at $209.00 on the same site, and the drone needs six batteries for flight. The drone can also only make a 15 minute flight. The result is the drone must make around 7 flights in a day, requiring plenty of spare batteries, further increasing the cost.

Lastly, the drone has an extremely powerful propulsion system that generates large electromagnetic interference for the magnetometer payload.

The DJI Matrice Pro is a powerful drone, but it has a short flight time due to its weight and power consumption, and also generates magnetometer interference that interferes with the data collection that the USGS needs.

1.3 The Need for a Better Solution

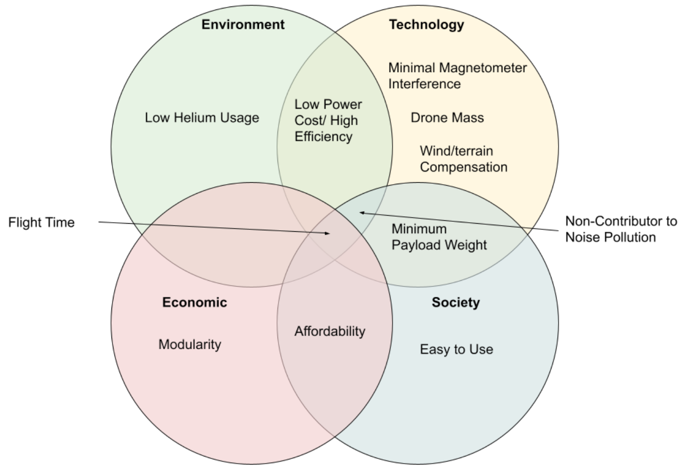


Fig. 1.1. Four Lenses Applied to the Project

The main goal of this project is to improve upon the total flight time tof drones and considers the four lenses perspective in Fig 1.1. To extend flight time, a lift bag is added to the drone to reduce the effective weight and energy needed to keep the drone afloat. The drag becomes a problem since the lift bag becomes large to compensate for any mass of the drone and payload but can be reduced by incorporating an aerodynamic design of the helium lift bag. The USGS's magnetometer payload is the primary sensor attachment for the project, and the design will be based around that weight. We wish to also balance these requirements for improvement with other criteria, such as a low magnetometer interference of the drone motors. This will be done by making sure the motors are sufficiently far away from the actual payload to ensure minimal remnant magnetic field, as well as taking advantage of the fact that smaller motors, with a lower magnetic field, can be used due to the lower power required to maintain height. Also, given that helium has had shortages before and is a non-renewable resource, we need to ensure that any helium is used efficiently, meaning helium leakage must be minimized. This goes with our affordability requirement because we want our drone to be more power efficient than the competitor drones, as well as more cost efficient with our components.

1.4 What to Expect and Report Layout

The project was determined to be unsuccessful in meeting the requirements given by the USGS, but some analysis done shows it is possible and worth pursuing. The report covers the team's design decisions, progress made, and verification tests of technical requirements, as well as individual and team reflections at the end of the report.

To begin the report, Chapter 2 establishes the physics introduced by incorporating a buoyant element to the drone design and uses the physical analysis to begin the high level design needed for the drone in order to incorporate buoyancy properly. Next, Chapter 3 introduces the lift bag design and frame design in order to meet all mounting requirements and force requirements that were presented in Chapter 2. Chapter 4 builds further on this, by introducing the design of the propulsion system in order to have a controllable drone for the performance requirements provided by USGS. Chapter 5 then goes over the sensor array and state estimation needed for the autonomous system development. Then, Chapter 6 combines the physics and state estimation of the previous chapters in order to deliver controllable flight in both Remote Control and Autonomous setups. Chapter 7 then analyzes the power requirements of the drone during flight, and lays out power distribution and estimates the total flight time. Chapter 8 attempts to use simulation to validate the design set forth throughout the previous chapters to ensure we could move forward with physical testing. Chapters 9 and 10 test the drone for verification of our requirements. Chapter 11 summarizes the legal and safety requirements required by the school, the FAA, and Covid19 regulations. Chapter 12 covers product costs and project funding. Chapter 13 makes final conclusions about the project and also covers the next steps if the project is continued. Lastly, Chapter 14 is the appendix with reference charts, data, and sources that were useful in our project design, and will be useful for anyone else seeking to replicate or build upon our result.

Throughout the documentation, the System Technical requirements are referenced, and the full document is included in the appendix. Any reference to the System Technical Requirements is abbreviated by STR RequirementID, RequirementName for consistency due to their prevalence in a design report.

Chapter 2: Design Considerations of a Buoyant Drone

In this chapter, we first introduce the flight conditions and payload requirements. Next, we address the necessary physics to consider with a buoyant drone design, as well as the problem with drone controllability. Finally, with the high-level system understanding developed in this chapter, we introduce the general design of the drone.

2.1 Project Goals and Requirements

In order to best implement our solution to the drone flight time problem, goals were created in order to set parameters for what stakeholders would want in our final product. These goals were defined by the needs of Jonathan Glenn, our client at USGS, who expressed their needs for what they would want to see in a long flight time drone system. Additional goals based on what we determined were necessary to achieve requirements given by the USGS were also created. Both sets of goals were broken down into 11 system level requirements each having relevant subsystem and component level requirements related to that system. Each of the system level requirements and how we planned on addressing them will be analyzed and addressed through this section.

2.1.1 Minimum Flight Requirements

STR 1.0.0 Drone Flight Time: Flight time shall be at least 30 minutes with payload.

The first and most important requirement was that of flight time. Since USGS's current drone could only fly for 15 minutes with its payload, Jonathan Glenn expressed that a doubling of this flight time would be something that they would value in our drone. Additionally this flight time would need to be achieved under our drones maximum flight conditions, defined by STR2.0.0,Drone Speed. We aimed to meet this requirement by adding buoyancy to our system to reduce the power draw needed from the motors. This was verified via power draw testing, seen in Chapter 7.2.3.

STR 2.0.0 Minimum Drone Speed: Drone shall be able to fly at least 5 mph in winds up to 15 mph.

This requirement addressed the maximum performance condition that Flight Time STD1.0.0 must be met under. USGS expressed that they generally travel at least 5 MPH when conducting a survey and that 15mph would be the maximum wind speed that they would collect data in. To achieve this requirement we wanted to reduce the drag on the system caused by wind by adjusting our systems shape. This was verified in our V-REP simulation, seen in Chapter 8.2.2.

STR 3.0.0 RC Control: Drone shall respond to user commands with pitch and roll angles within ±0.1 radians and a height of 1±0.15m.

USGS expressed that they would want manual control over the drone, more specifically they would want the ability to switch to manual control if some data of interest or an obstacle appeared to its scientists. We additionally wanted to add closed loop control to the remote control to help maintain stable angles so that the system would be easy to pilot. This was verified in Matlab, seen in Chapter 6.8

STR 4.0.0, Autonomous Control: Drone shall maintain stable pitch/roll/height and follow a path.

USGS expressed the drone should have autonomous capabilities. They wanted to be able to use these capabilities so that the drone could fly a predetermined flight path. Additionally they wanted the system to be able to resume a given flight path if it had been interrupted in favor of manual control. This would be achieved by designing a closed loop system capable of terrain tracking by using ultrasonic sensors as well as waypoint navigation using GPS data. The design for autonomous was not completed, considerations can be viewed in Chapter 6.9

STR 5.0.0, Cost: Drone shall cost less than $10,000. Stretch goal of less then $6,600

USGS stated that the current drone they would pay up to $10,000 for a drone with the capabilities they requested. Additionally the drone they currently have costs $6,600 . We kept a bill of materials and made estimates as to what labor costs for manufacturing would be, but have not been able to fully verify whether this requirement would be fully met. For our analysis of system cost see Chapter 12, or our bill of materials in the appendix.

STR 6.0.0, Magnetometer Interference: Interference from drone shall be less than 10nT on payload

Since USGS collects magnetometer data with their drone they expressed that by reducing the interference from the drones motors too less than 10nT would be beneficial to their research, as their current drone causes high magnetometer interference. Since our drones motors would be drawing less power with our reduced effective weight our system would have a reduced electromagnetic field. We aimed to use the reduction of this field in order to achieve this requirement. This was not verified.

STR 7.0.0, Drone Safety: The drone, its usage, and build should be safe to all individuals involved.

As a team we needed the drone to be safe to use. This means that the drone would have a standard for safe operating procedures. Additionally the drone must be able to follow safety regulations while it is being operated. More information can be found in Chapter 11.

STR 8.0.0. Helium Leakage: The lift bag shall maintain 90% of its buoyancy over a one week period.

Since Helium is a limited resource and has a high price we wanted to make sure that our system would be able to retain helium over the course of a week, so that our client would be able to do all their testing over the course of several days without having to refill the system. We wanted to implement some sort of seal onto the lift bag so that it could be closed after filling with minimal leakage. Tests were conducted to measure this and it was found that we did not meet this requirement, can be seen in ChapterC 9.

STR 9.0.0, Legal Compliance: The drone and team shall abide by all applicable laws for drone flight.

This requirement was to ensure that our drone followed all laws and regulations so that when the system was finished no additional work would need to be done before our client could use it. To try and meet these requirements FAA regulations were checked and proper avenues for paperwork were looked into. Additionally regulations related to the Covid-19 pandemic were followed. More information can be found about legal compliance in Chapter 11.

STR 10.0.0, Noise Level: The drone should be quieter than 65dB at 5feet away.

The last requirement USGS gave us was that by reducing the noise level of the drone to 65dB. 65dB is the ambient sound level of an urban environment so at 65dB or lower USGS could use the drone closer to populated areas with less noise complaint. Tests were conducted to measure this and it was found that we did not meet this requirement, more information about the tests can be seen in Chapter 9.

STR 11.0.0, Manufacturability: The drone should be able to be manufactured with equipment within our access, further decomposed in subsystem requirements.

Our drone needs to be able to be built within the first 4 months of the senior capstone project parts II and III with an additional month and a half for testing and repairs. The requirement means that it had to be built with equipment within our team's work space, such as 3D printers, soldering equipment, and sewing machines. This requirement was to ensure that a prototype could be built. More information can be found in Chapter 12.

2.1.2 Minimum Flight Requirements

Our stakeholders' primary need for this project is to have a flight time longer than the many of the quadcopters and hexacopters currently used in research applications. Currently USGS uses a DJI Matrice 600 Pro, a drone that when carrying the MagArrow (a 1kg magnetometer used by USGS) has a 15 minute flight time. Due to the short flight time USGS needs to conduct 7 flights in order to conduct just one survey. USGS has expressed interest in a drone that could fly twice as long as the Matrice 600 Pro, at least 30 minutes. The 30 minutes would be achieved at the system's maximum performance specified in STR 2.0.0,Drone Speed, all motors providing maximum allowable force for the entire duration of the flight. We, however, theorised that motor usage could be lowered to the point that an even longer flight time could be achievable when the system is at minimal performance, when the motors are only exerting enough force to hover. So we set a stretch goal in STR 1.0.0,Flight Time of 1 hour. Since drones spend the majority of their power on motors providing force to counteract the system weight in order to stay airborne. Our primary method to achieve STR 1.0.0, Flight Time, was to implement a buoyant force to the drone which could reduce the effective weight of the drone. By reducing our effective weight, the motors on our drone would use significantly less power draw since the force needed to stay airborne generated from the propellers will be lower than a system with a higher effective weight. The decreased power draw would decrease the battery usage of our drone extending our flight time.

The next question to consider is what is a maximum performance scenario? According to USGS they usually fly their drone at least 5 MPH when conducting surveys. They also want to conduct surveys in mild to moderate winds up to 15MPH. These speeds give our maximum performance requirement. The drone must be able to fly at least 5MPH in a 15MPH head wind. Therefore our systems motors must be able to supply sufficient thrust to keep the drones speed and ability to maintain height even with the extra drag felt by the system in a headwind. Additionally a battery must be chosen in order to support the power draw needed by the drones motors to be at this throttle for 30 minutes as well as power the rest of the drones electrical systems for the full duration of the flight. For more information on the power draw of the systems see Chapter 7.

2.2 Physics of a Buoyant Drone

Adding a buoyant force to the drone comes with a variety of pros and cons due to the physics of buoyancy. First we decided to use helium as our lifting gas. Using Helium is standard practice in buoyant systems today since helium is inert while hydrogen, the only other commonly used lifting gas, is highly combustible and poses serious explosive and fire risks when used. Helium has a density of 0.164 g/L. Compared to air which has a density of 1.18g/L[13] both at STP. Since the lift force can be found with $F_l = (\rho_1 - \rho_2)V$. Where ρ_1 is the density of air, ρ_2 is the density of helium, and V is volume. It is calculated using the given densities that a cubic meter helium applies 1N of lift force. Thus the volume of our liftbag was decided based on the weight of the drone and the weight of the payload we need to carry. The addition of helium to our system causes a variety of other direct effects such as creating a buoyant moment, and indirect effects such as an increase in drag due our increased surface area from the helium lift bag, and reduced electromagnetic fields due to the motors not needing to compensate as much for the systems weight. In this section we will take a look at the pros and cons that adding buoyancy to our system will cause. Then Internal equations of motion are developed. The forces acting on the system can be seen in figure 2.1 at the end of the section.

2.2.1 Benefits of a Buoyant Force

The most straightforward impact buoyancy has on our system is a reduction in effective weight. Since the buoyant force is applied opposite the force of gravity, these forces cancel with each other resulting in a reduction in net effective weight. In the case of an object with buoyancy the effective weight is found simply by using $weight = F_{gravity} - F_{buoyancy}$(2.1). Since the buoyancy is reducing the effective weight of our system the motors will need to apply less force to lift the drone. Due to the smaller lift needed, the motors have a reduced power draw. The smaller power draw of the motors, which directly causes a smaller current at the same voltage, causes a smaller magnetic field and flux to be generated[15]. This smaller flux should help to reduce the magnetic interference on the payload carried by our system. The greatest effect of the reduction of the power drawn from the motors is a decrease in battery usage. Since power needed by the motor is equal to torque times angular velocity, a smaller torque applied by the motors will require less power. This will result in a longer flight time of the system since motors are the primary power drawing component in most drone systems.

2.2.2 Drawbacks of a Buoyant Force

The addition of a buoyant force also has its drawbacks. The buoyant force created by the helium causes a buoyant moment between the center of gravity and the center of buoyancy. This moment will cause the center of buoyancy to always be pushed to the top since buoyancy acts opposite of gravity. Because of the buoyant moment, the center of buoyancy will be designed to sit above the center of mass. With the center of buoyancy being on top the moment becomes self correcting and holds the lift bag upright. Although this has the benefit of preventing our system from capsizing, most drones steer by tilting which is not viable with the buoyant moment since it will hold our system upright, preventing it from tilting. In order to compensate for the buoyant moment, propulsion systems are placed equidistant around the body of the drone. Each of the propulsion systems consists of a motor with a propeller at the end of a shaft connected to a servo. These servos change the angle that the motors are applying their force. The rotation from the servos in combination with operating the motors at different speeds allow the system to be steered even with the buoyant moment. Additionally this method of moving will keep our system stable with pitch and roll angles always remaining close to 0 radians. By adding this propulsion system we were effectively able to turn this drawback into a benefit to our system.

Another constraint of a buoyant system is that a lift bag with a large volume is needed in order to contain sufficient helium to generate the required lift force. The large volume of the lift bag causes the drone to have a huge frontal surface area, resulting in a large drag force applied to the drone when in motion or in high headwind conditions. The drag causes a large issue especially in respect to meeting the requirement STR 2.0.0, Drone Speed. In order to compensate for the drag, the shape of the envelope(the enclosure surrounding the lift bag) was made into an ellipsoid rather than a sphere in order to reduce drag when moving parallel to the ground. Drag on an ellipsoid can be defined by the following equations 2.2-2.4 from *Applied Fluid Dynamics Handbook[14]* where ρ is Air density $1.225\frac{kg}{m^3}$at STP, U is Air Speed ,D is High radius, and L=Width radius.

$$Drag\ Force\ =\ F_d = \frac{C_d \rho A U^2}{2} \qquad (2.2)$$

$$Drag\ Coefficient\ =\ C_d =\ 0.44(\frac{D}{L})\ +\ 0.016(\frac{L}{D})\ +\ 0.016\sqrt{(\frac{D}{L})} \qquad (2.3)$$

$$Frontal/Attack\ Area\ =\ A\ = \frac{\Pi D^2}{4} \qquad (2.4)$$

From equation(2.3) it can be seen that by reducing the frontal attack area of the shape and the drag coefficient the force of drag will be reduced. From equations (2.3-2.4) it can be seen that reducing the height(D) is the most effective way to reduce both the frontal attack area of the shape and the drag coefficient. This is important since weight is a force that only acts in the axis of gravity; the reduced effective weight of the system does not give us any benefits in horizontal motion. Acceleration is defined as $a\ = \frac{F_m - F_d}{m}$(2.5)where F_m is the force from the propellers and F_d is the force of drag[16]. Combining this with the drag force found in equation 2.2, it can be seen that the larger the drag we experience the smaller our acceleration will be. This is critical because velocity is defined as $v\ =\ v_i\ \ +\ at$(2.6) where v_i is initial velocity, a is acceleration, and t is time[16]. Since our acceleration will be affected by drag, so will our velocity at any given time. Therefore in order to meet STR 2.0.0, Drone Speed it is critical that we reduce drag. The envelope therefore is designed to be short and wide to maintain its volume while reducing drag. The reduced height will reduce the force of drag experienced by our drone, the smaller drag from this will increase our possible velocity allowing us to meet STR 2.0.0,Drone Speed. The width of our system is made wider in inorder keep the volume of helium contained the same, keeping our effective weight low enough to meet STR 1.0.0,Flight Time. For More information about envelope design see Chapter 3. For more information on flight time see Chapter 7.

Another factor that constrains the use of a buoyant system is the cost of helium. Helium is a nonrenewable and rapidly depleting resource resulting in a high cost. Additionally, the user would need to have a tank capable of holding the required helium which would add additional cost. The high cost is important since it is important to meet our STR 5.0.0,Cost. In order to deal with this constraint, we aimed to minimize helium loss so it can perform flights for up to a week without needing to refill. This allows users to spend time collecting data over several days without needing to be concerned about refilling the drone. In the case of helium balloons deflate because helium atoms are small enough to slip between spaces in balloon material [17]. The helium loss is the case with any material, however can be reduced with materials like mylar which have smaller gaps in their structure when compared to nylon[17].Ultimately the issue was not solved in the duration of this project. For information on how this was tested, see Chapter 9.

2.2.3 Equations of Motion for Internal Forces

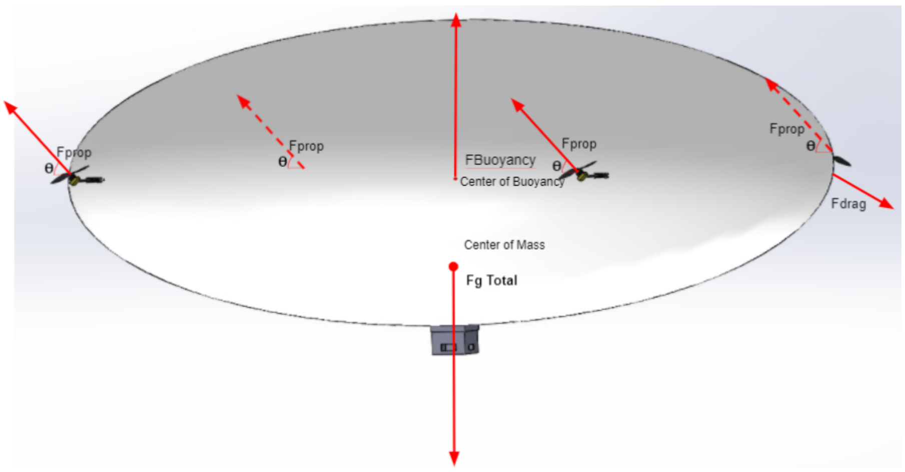


Fig. 2.1. Force Diagram of Full System

Now that the basic pros and cons of buoyancy are layed out we can start to define our full system equations of motion. These will be based on all the major force vectors in our drones body frame. The first major force vector is CM_x, the vector from the center of mass to the center of rotation for the motor, where x denotes the motor number. T_x will be the scalar throttle of each motor. M_xP is the unit vector from the center of rotation for the motor to the center of the propeller. CM_x will be a constant for each propeller decided on the mechanical design, and M_xP will vary from each motor depending on its angle and will have the value of $[\cos\Theta, 0, \sin\Theta]$ where $\Theta = 0$ when propeller angle is flat on xy plane. Finally CB is the vector from the center of mass to the center of buoyancy.

With the force vectors defined we can derive an equation for the net internal force on our system. Starting with the forces created by the propellers. The position of each of the four propellers placed around the drone can be given by the sum $M_xP*d + CM_x$ where d is the distance between the center of motor rotation and propeller, and the force direction is given by the M_xP vector. The force from each motor is given by M_xP*T_x. This results in the net propeller forces to be

$$F_{Prop} = \sum_{x=1}^{4} M_x P * T_x (2.7)$$

The next internal force we look at is the buoyant force, more specifically the lift force in the z axis left over after gravity and buoyancy cancel out. The force due to gravity is simply given by the mass of the drone multiplied by the acceleration due to gravity. The lift force is calculated by $F_B=\rho Vg$ where rho is the density of air and V is the volume of the lift bag. The mass of the bag and helium would be considered part of the drone mass, or the mass of helium could be included using the equation

$F_B=(\rho_{air}-\rho_{helium})Vg$. (2.8)
This results in net lift forces of
$F_{lift}=F_g+F_B$(2.9).
These two sets of forces defined in equations 2.7 and 2.9 can be combined in order to find the net internal force of our system

$$F_{internal} = F_{lift} + F_{Prop} \quad (2.10)$$

$$F_{internal}=(F_g+F_B)+(\sum_{x=1}^{4} M_x P * T_x) \quad (2.11)$$

Similar to forces we can derive an equation for the net moments created from internal forces. Again we start with the propellers. The moment caused by each motor can be calculated by taking the cross product of the force vector by its positional vector. This results in the moment from the motors to be defined as.

$$M_{mx} = \sum_{x=1}^{4} F_{mx} * (CM_x + M_x P_x * d) \quad (2.12)$$

Next buoyancy can be examined. A moment is created between the center of mass and the center of gravity. The buoyancy moment is caused by the buoyancy force wanting to be opposite the gravity force, and can be calculated with

$$M_{buoy} = F_B * (R * CB) \quad (2.13)$$

Using these two moments a total moment for the system can be found

$$M_{net} = M_{buoy} + M_{mx} \quad (2.14)$$

$$M_{net} = F_B * (R * CB) + \sum_{x=1}^{4} F_{mx} * (CM_x + M_x P_x * d) \quad (2.15)$$

With these total internal equations of motion derived for both internal forces and moments we were able to apply them when making design decisions through the process of creating the Barone. These equations were especially useful when creating the control system that would be created in order to meet STR 3.0.0,Remote Control and STR 4.0.0, Autonomous. For more information on the control system see Chapter 6.

2.3 General Design Overview

Based on the effects of the forces experienced by a buoyant system, as well as the high level system requirements defined by our team and USGS, we designed our system to utilize/compensate for the effects of the buoyancy while still aiming to meet the system requirements. This section will contain a general overview of this design and how it attempts to accomplish this.

2.3.1 System Size and Shape

As previously mentioned our primary method to achieve STR 1.0.0, Flight Time is to add buoyancy to the drone. This is accomplished by adding a lift bag to the drone's body. The goal of adding buoyancy is to reduce the effective weight of the system. In order to prevent loss of the system it was decided that the effective weight of the system should remain above 0N so it would not float away if control was lost. This became Weight Requirement STR 1.2.0: The drone shall have an effective weight of between 0 and 5 N.

In order to meet STR 2.0.0, Drone Speed it was deemed necessary to make the shape of the envelope an ellipsoid rather than a sphere. Based on the drag analysis in section 2.2.2., an ellipsoid shape would have a smaller drag force acting on it then a sphere. By keeping the drag and weight as small as possible, smaller motors could be used to control the drone. Since the motors would need to exert less force in order to counter the force of drag and weight. This would help us reach STR 1.0.0, Flight Time while also achieving STR 2.0.0, Drone Speed For more information about the drone frame system see Chapter 3.

2.3.2 Propulsion System

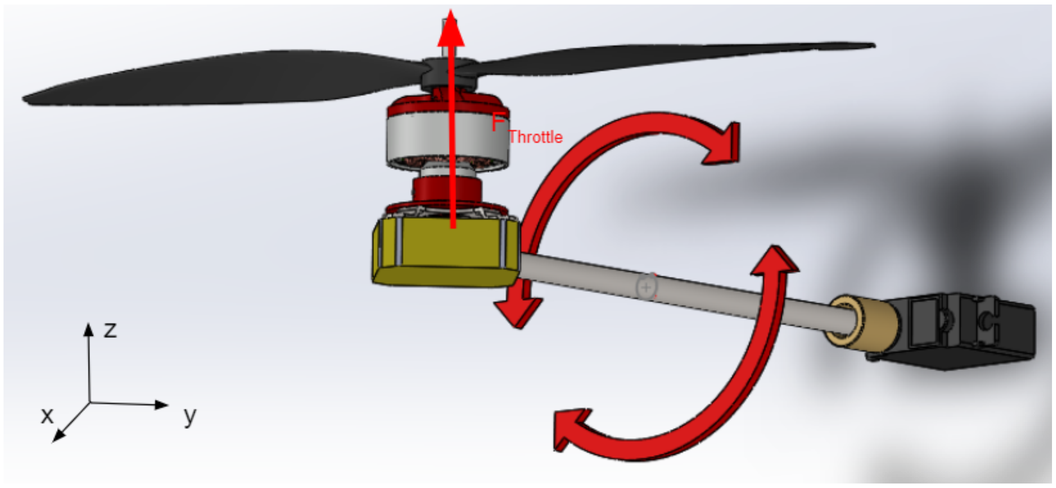


Fig. 2.2. Barone Propulsion System

In order to achieve STR 3.0.0, Remote Control we implement design features to overcome the buoyant moment discussed in section 2.2.2. Since our system cannot tilt like a normal drone due to the buoyant moment, our system uses propellers whose attack angles can be adjusted figure 2.2. We can rotate our propulsion force along the XZ axis using servos to change direction of forces output by the propellers figure 2.2; this allows us to have a controlled flight. Propellers can point up, down, forward, backwards, and anywhere in between. The propulsion system is implemented in order to be able to steer the drone even with its buoyant force keeping it upright. The propulsion system also has the additional benefit of keeping the entire system relatively stable with pitch and roll angles always remaining close to 0 radians compared to standard drones who tilt to steer. four of the propulsion systems will be mounted equidistantly around the envelope so that their forces would be applied evenly and maintain stable flight. For more information about the propulsion system see Chapter 4.

Additionally the motors are mounted far from the magnetometer payload this helps reduce magnetic interference and reach STR 6.0.0, Magnetometer Interference. Since the motors will be the largest source of magnetic interference and magnetic field is defined as $I = \frac{1}{r^2}$(2.16) where the intensity of the field[13], The motors and payload are placed far apart from each other in order to reduce the strength of the magnetic field from the motors at the location of the magnetometer payload. This was done in order to help achieve STR 6.0.0, Magnetic Interference.

2.3.3 Control System

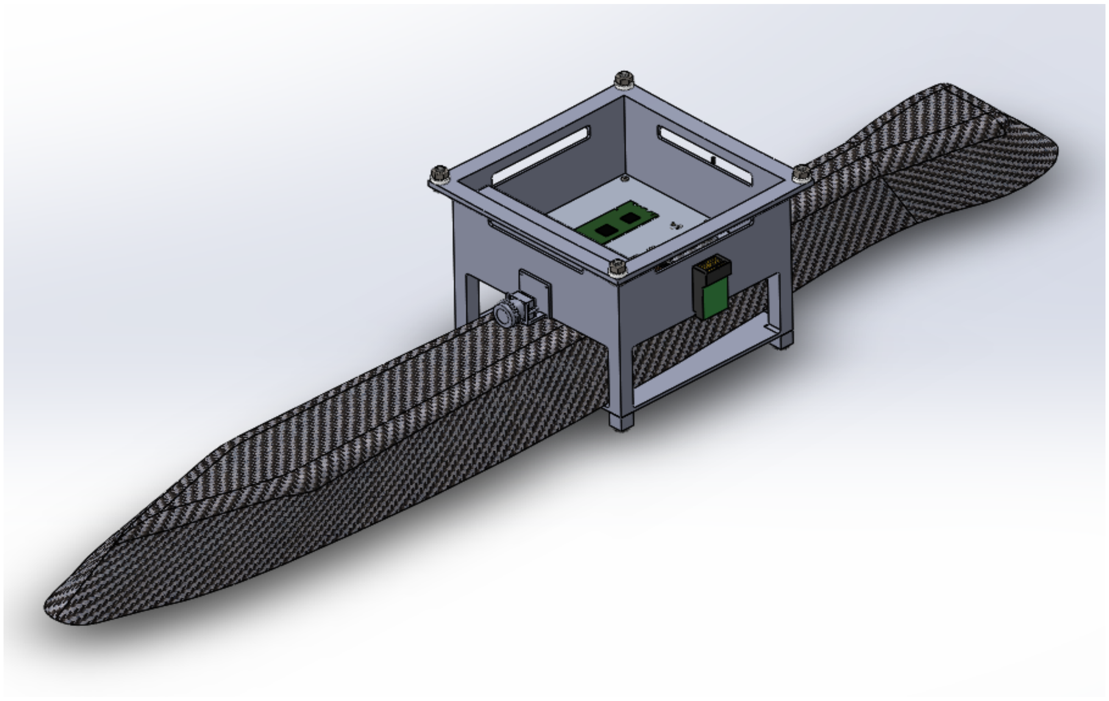


Fig. 2.3. Gondola for Electronic Housing

In order to achieve STR 4.0.0, Autonomous as well as aid with STR 3.0.0, Remote Control, we aimed to implement an autonomous as well as closed loop remote control system that would utilise various on-board sensors to maintain a stable flight in both autonomous and remote control modes. In the design of this implementation we created several subsystem and component level requirements some of which will be introduced here.

- System response STR 3.1.0: The drone shall respond to user input in < 0.5 seconds.

- RC Controller STR 3.1.1: The controller shall be capable of providing forward, turn, ascend, and descend commands
- Data Feedback STR 3.2.0: The drone shall be able to send feedback to the user
- Camera STR 3.2.1: The drone shall send camera feedback to assist in user controlled flight
- Low Battery STR 3.2.3: The drone shall send camera feedback to assist in user controlled flight
- Path Following STR 4.1.0: The drone shall be able to follow a path specified by the user in up to 15mph wind with a positional accuracy of 5m
- GPS Sensor STR 4.1.1: The sensor shall be accurate to within 5m of its location with sampling of at least 3Hz
- Terrain Tracking STR 4.2.0: The drone shall maintain a constant height above the ground, approximately 1m, and adjust height as needed. < 15% overshoot
- Ultrasonic Sensors STR4.2.1:The sensors shall be able to monitor the area in front of the drone in order to maintain constant height of 1 m.
- Barometric Sensor STR 4.2.2: The sensor shall be able to monitor altitudes above 4m for drone altitude awareness.
- Error Handling STR 4.3.0: The drone should be able to detect flight errors and compensate accordingly, specified in component section
- IMU Sensor 4.3.1: The IMU should be able to detect crashes and abnormal situations and feed the data back into the system

First an on board microcontroller needed to be implemented in order to communicate between the remote controller, sensors, and propulsion system to meet all of those subsystem and component STRs. This as well as camera, voltage alarm, GPS, barometer, IMU, remote control transceiver, battery and payload would be housed in a gondola attached to the bottom center of the envelope figure 2.3.

In order to meet STR 4.2.0, Terrain Tracking and STR 4.2.1, Ultrasonic, multiple ultrasonic sensors would be used to keep track of the distance the drone is from obstacles both in front of and below the drone. Two of these sensors; one on the bottom of the gondola, and one of the ones on the front of the drone will be tracking height. Two additional ultrasonic sensors at the front will monitor for obstacles directly in front of the drone for collision avoidance. The data from these ultrasonic sensors will be fed into the microcontroller in the gondola as inputs to the software control system to maintain stable height.

2.4 Conclusion

After requirements were defined based on the needs of USGS, a buoyant force was thought to be added to our drone system in order to reduce the effective weight. This reduction in effective weight allowed for the motors to create less power draw allowing us to achieve STR 1.0.0, Flight Time.

However it was revealed through force analysis that by adding a buoyant lift bag to our drone design drag would be increased, resulting in lower achievable velocities by the drone. Therefore efforts had to be made in order to reduce the drag experienced by our system in order to achieve STR 2.0.0, Drone Speed.

Chapter 3: Envelope and Drone Frame Design

In this chapter we discuss each of the major mechanical subsystems of the Barone's envelope and frame, the specifications they were designed to meet, and the reasoning behind those specifications. We will then be discussing the fabrication process and verification of each of these parts, with a full analysis of the verification results.

3.1 Barone's Mechanical System.

The mechanical parts in The Barone can be separated into four distinct subsystems: the Ultrasonic array, Propulsion System, Gondola, and Envelope. Certain systems had specific subsystem or comment level requirements that had to be considered related to the goals of that subsystem. Additionally STR 2.0.0, Weight had to be considered for all subsystems as well as STR 11.0.0, Manufacturability. In this section the specifications and design choices for each of the subsystems will be explored.

3.1.1 Ultrasonic Array

The first subsystem is the ultrasonic array. This subsystem aimed to help satisfy the STR 4.2.1, Terrain Tracking. The ultrasonic mount holds three ultrasonic sensors on the front of the drone to provide our control system with information about the distance of obstacles in front and below the drone. For more information about our sensor array see Chapter 5 or for more information about the controls system see Chapter 6.

First, since the ultrasonic mount needed to hold 3 uniquely shaped objects on the curve of our envelope, it was decided that the part would be designed to be 3D printed in order to meet STR 11.0.0, Manufacturability. We decided that two ultrasonic sensors would face forward to assist with collision detection in front of the drone. The third ultrasonic sensor would be angled downward to look for obstacles in front of the drone that are below the line of sight of the two above to help with maintaining a stable height as well as obstacle avoidance. In order to best utilize the 15° field of view of the ultrasonic sensors. The two forward facing ultrasonic sensors were angled 7.5° in opposite directions so their field of views would not have overlap and we would be able to effectively double our forward facing field of view from 15° to 30°. The center ultrasonic sensor was angled downward at 45° so that its field of view would overlap with the ground while still looking forward. Each ultrasonic sensor was given its own 46x22mm(size of the ultrasonic sensor) mounting surface on the 3-D printed part which provided mounting holes for the sensors to be attached. The surfaces also had a cut out so that the soldered chips on the back of the ultrasonics would not be damaged. Finally the back of the part was given a curve to match that of the 54" radius of the envelope. A thin plate with the same height, width and curve was also printed that would be placed inside the envelope to help hold the part though . The ultrasonic sensor array is depicted in Figures 3.1 and 3.2.

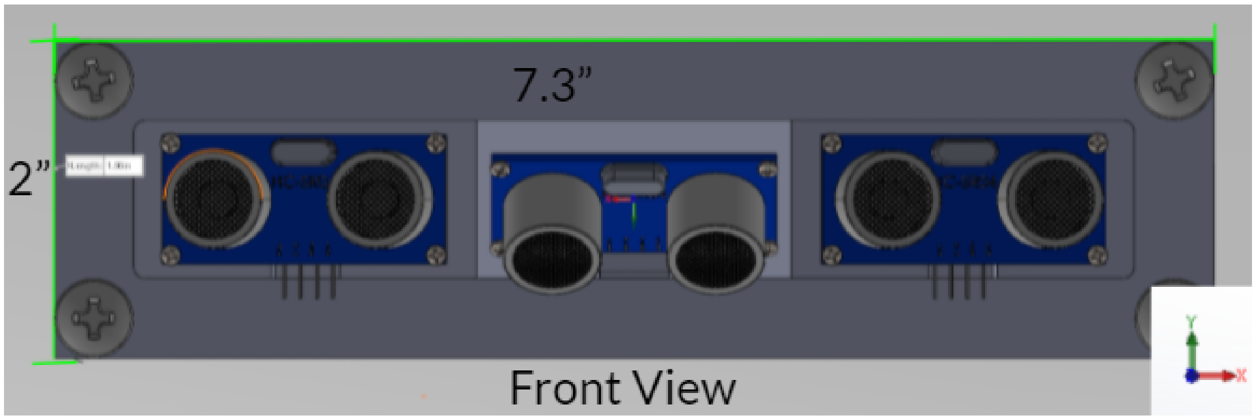


Fig. 3.1. Ultrasonic Array Front View

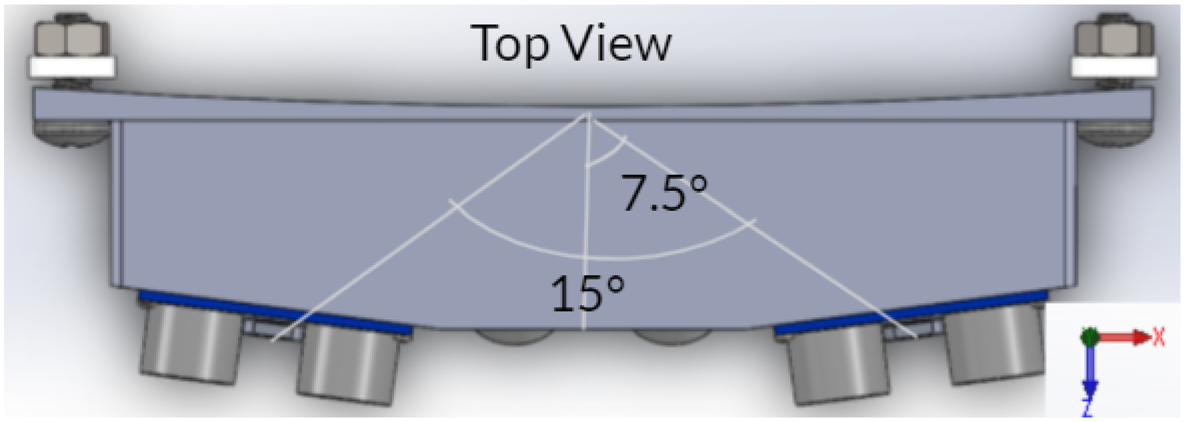


Fig. 3.2. Ultrasonic Array Top View

3.1.2 Propulsion System

The next subsystem is the propulsion system. This subsystem's goal was to hold the propulsion module discussed in Chapter 2 in order to fulfill STR 3.0.0, Remote Control and 4.0.0, Autonomous. To learn more about the actuators in the module see Chapter 4. For more information about the controls system see Chapter 6. The 4 propulsion modules would be placed equidistant from each other around the system so that the force they supply is balanced around the drone. The propellers need to be placed without posing a potential danger to the envelope.

Similar to the ultrasonic mount, the mounting block for the propulsion modules would be 3D printed in order to fit the shape of the servo as well as fit the curve of the 54” diameter envelope. To fit the servo into the mount, a hole was cut into the block shape with the exact dimensions of the servo, 20x40x30mm. The servo would be placed into this block and nuts screwed into holes placed to line up with the attachment bracket of the servo. Since the servo needed to be connected to the motor such that the servo's axis of rotation would be perpendicular to the force axis of the propellers, a carbon steel shaft was attached to the servo using a 25T servo coupler. The 25T represents the 25 teeth the servo has on its rotation gear. The shaft itself was made to be 6" to avoid the 4.5" radius propeller from making contact with the envelope. At the end of the shaft a 3D printed mounting block was to be attached. This mounting block would be the size of the motor stand so that the motor would be able to be mounted to the block perpendicular to the servo shaft. Lastly, a mounting plate with the same dimensions of the bracket was printed to go inside of the envelope similar to the ultrasonic plate. This system is depicted in figure 3.3.

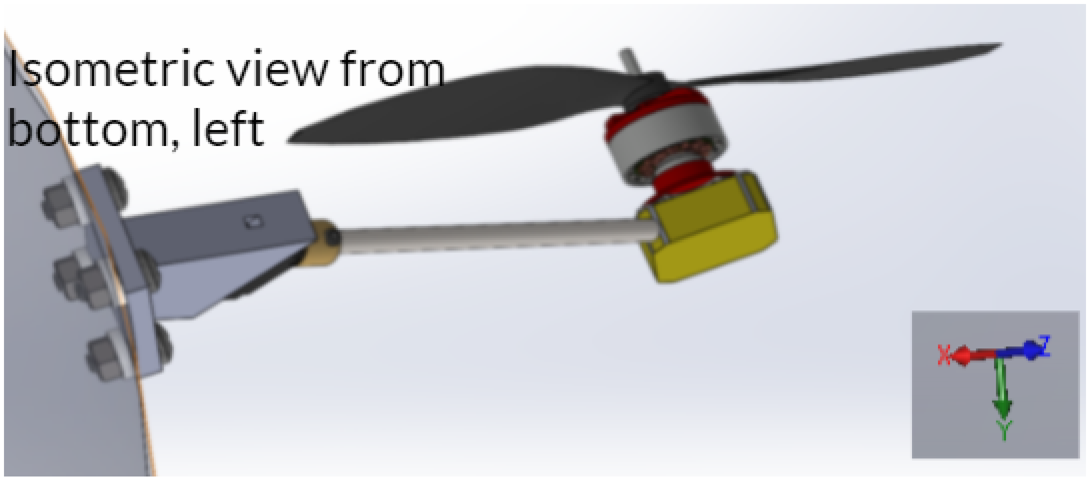


Fig. 3.3. CAD of Propulsion Subsystem

3.1.3 Gondola

The next subsystem is the gondola. This subsystem was designed to hold all the remaining onboard electronics needed for the drone that are not contained within other subsystems. The gondola can be broken into 3 distinct sections. The primary housing basket, the payload bay, and the outside attachments. Each of these subsystems were used to hold different electronic components. To learn more about the housed electronics see Chapter 5.

First, the primary housing basket, this section holds the battery, the voltage regulators, the ESCs, and the PCB used to house the sensors and microcontroller. The primary housing basket had holes added to the sides so that wires could be passed from the inside of the gondola to the propulsion system, external computer, or other electronics. The dimensions of the housing basket, 7x7x5", were chosen in order to house all the necessary parts, the largest of which was the battery whose length of 6.8" defined the length and width requirements of the gondola. The bottom of the housing basket was made of a thin aluminum plate that the PCB, ESC, and switching regulators would be attached to with standoffs. An aluminium plate was chosen to help regulate any heat inside the gondola due to its high specific heat capacity compared to most medals.

The second section is the payload bay. This section was primarily designed to house the MagAero, the magnetometer used by the USGS to collect their data. Since the MagAero has a height and width of 50x150mm, the height and width of the payload bay were allocated extra room to be 53x158mm to allow the payload to be tied down or tethered to the payload bay. On the bottom of the payload was the final ultrasonic sensor of our system. One ultrasonic sensor was mounted directly to the bottom of the payload bay and extensions were added so that the gondola would rest on the extensions rather than the ultrasonic sensor. Since the extensions were also going to be 3D printed, the overall height of the whole system needs to be 7” or less in order to fit within the printer we had access to. In order to comply with STR 11.0.0, manufacturability, the 7” height limit leaves the payload bay and housing basket little room for error in the vertical dimension. The extensions were also designed to be printed separately so that they would not take away from the possible height of the gondola.

The final section of the gondola are the parts attached to the outside. This section includes the remote control transceiver, the camera, the voltage alarm, and the power switch. These parts all fit onto the sides of the housing basket with minimal modifications to the design. All of these parts were able to simply be attached to the side of the gondola with command strips. Going into the housing basket, wires are run to the power switch, remote control transceiver, and voltage alarm. The gondola with all attached sensors is illustrated in figure 3.4.

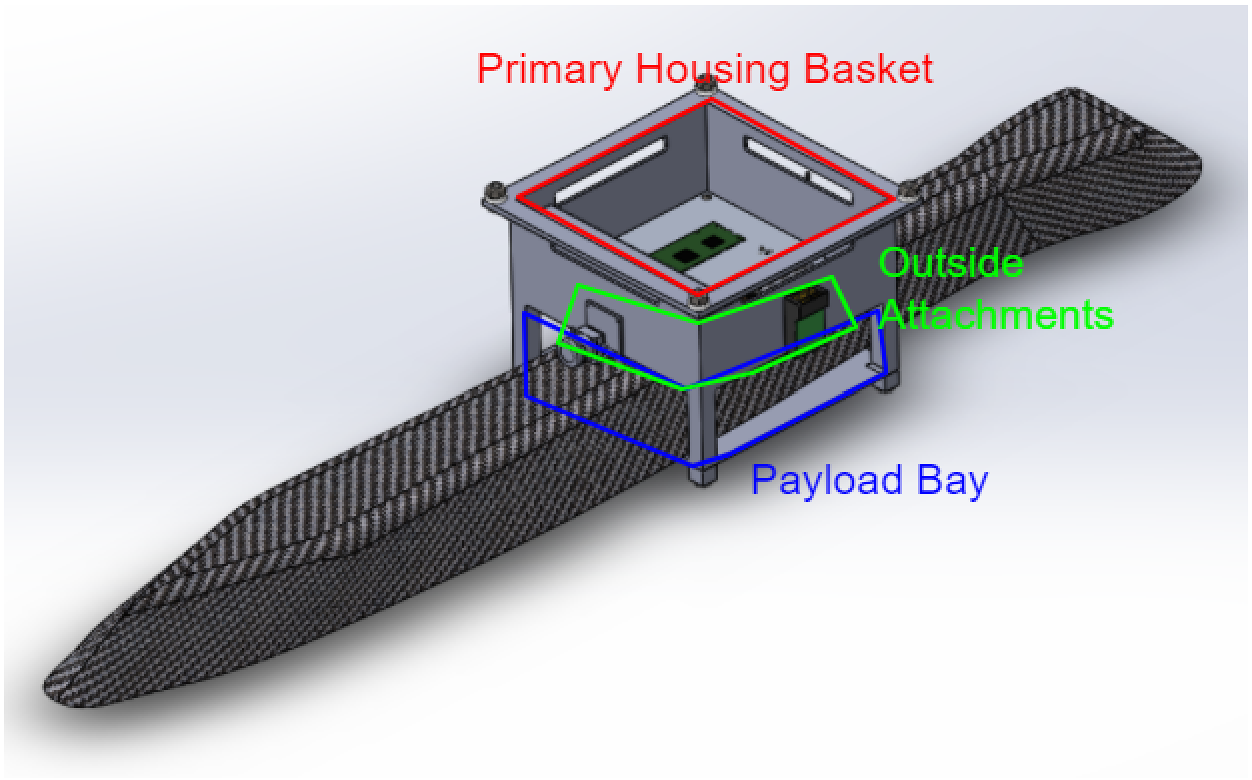


Fig. 3.4. Gondola CAD Housing all Electronic Components.

3.1.4 Envelope and Lift Bag

In order to meet STR 1.2.0, Weight, as well as STR 2.0.0, Drone Speed, the final subsystem designed was the envelope and lift bag.

After work began on the project it became clear that our drone would weigh about 45N without helium. The breakdown of this weight can be seen in our weight budget which can be found in the Chapter 14 appendix. To bring our weight down to the range specified in STR 1.2.0, Weight, we would need to add 4m^3 of helium to apply 40N of buoyant force opposite gravity. Therefore 4m^3 became the volume of Helium in the lift bag in order to prove the necessary buoyancy to meet STR 1.2.0, Weight.

In order to accomplish the STR 2.0.0, Drone Speed, we adjusted the shape of the envelope around the lift bag to be an ellipsoid. Using the drag equation introduced in Chapter 2, a basic drag analysis was done in MATLAB on ellipsoids of different sizes to find the drag our drone's envelope would be experiencing. Based on the analysis shown in figure 3.5, it can be seen that by reducing the height of the envelope, horizontal drag would be reduced. We chose an ellipsoid which would experience 4.6N of drag force in 20 MPH forward airspeed. The reduction in drag would be accomplished by fitting a 108" diameter balloon inside a 108"x108"x40" envelope; these dimensions are as shown in figure 3.6. Additionally, 4.6N was chosen as it was the lowest drag achievable with the 108" Lift Bag and also less than the effective thrust of our propulsion system. To learn more about the propulsion system see Chapter 4. The envelope additionally functions as a surface to attach parts to the system without damaging the lift bag.

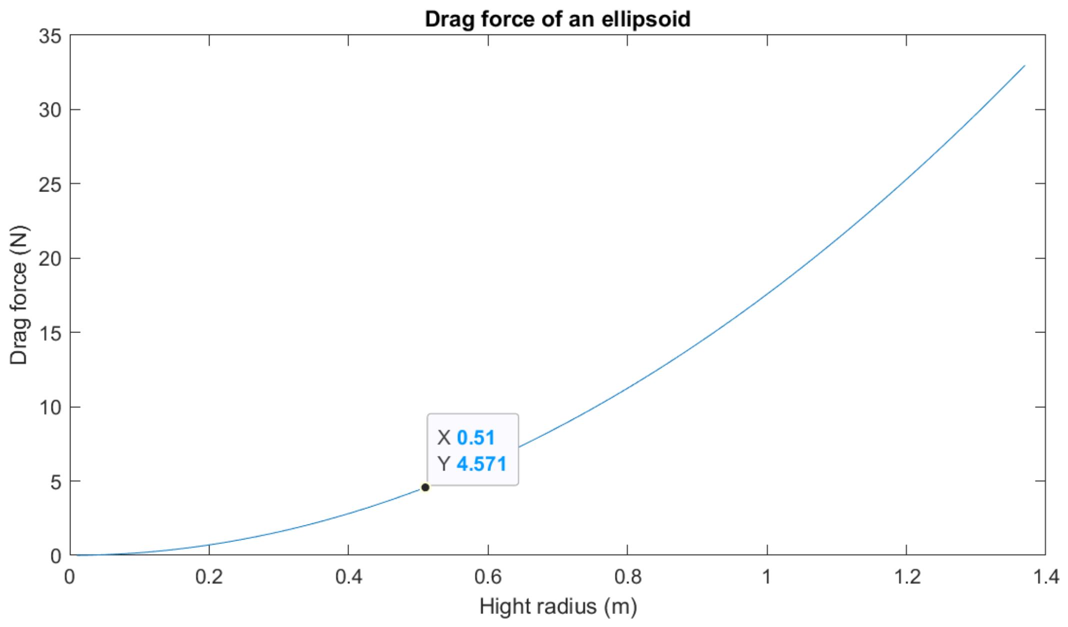


Fig. 3.5. Matlab Graph of Initial Drag Analysis on the Drone

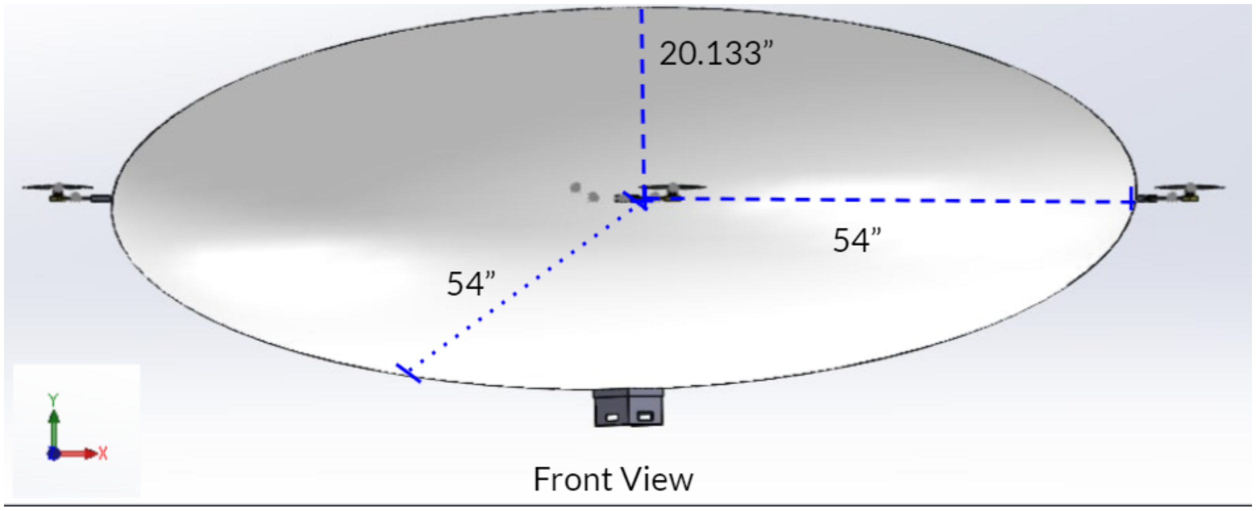


Fig. 3.6. Envelope CAD Design

Once the shape was decided, a material had to be chosen, this material needed to be able to withstand the largest forces applied to it from the attached parts. We found that the largest applied force to the envelope during a flight comes from the weight of the propulsion system falling after the motor turns off. For more information on the propulsion system see Chapter 4. Based on the force diagram in figure 3.7. The following equation was derived $F_e = F_m \frac{r_s}{r_b} \sin\theta^2$ (3.1)[16]. By plugging the maximums of our known values into equation 3.1 it was found that the worst case pressure applied to the envelope during flight would be 6.63psi. Once the maximum pressure value was known, a Pugh chart, as seen in figure 3.8, was created with various potential materials. 1.0 HyperD nylon was chosen as it could both withstand the worst-case pressure as well as maintain a low weight and price point to help meet STRs 1.2.0, Weight, and 5.0.0, Cost.

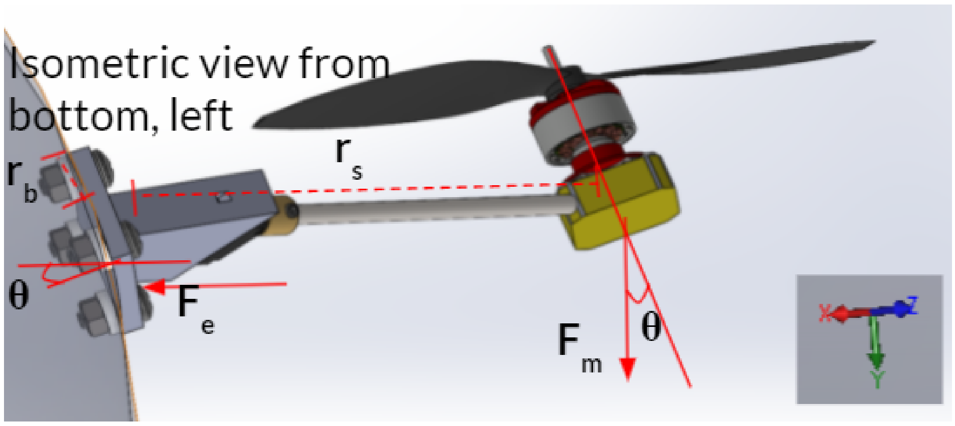
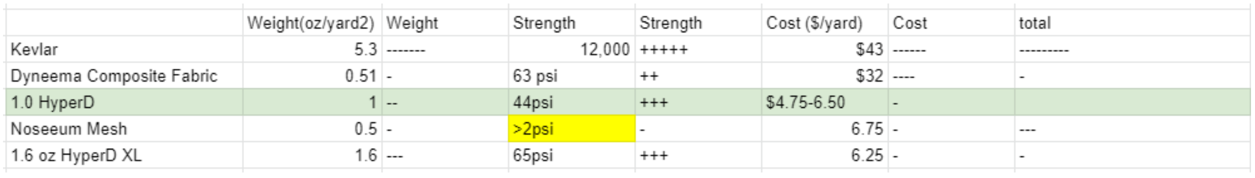


Fig. 3.7. Force Diagram of Propulsion System Applying Lever Force to Envelope



	Weight(oz/yard2)	Weight	Strength	Strength	Cost ($/yard)	Cost	total
Kevlar	5.3	-------	12,000	+++++	$43	------	---------
Dyneema Composite Fabric	0.51	-	63 psi	++	$32	----	-
1.0 HyperD	1	--	44psi	+++	$4.75-6.50	-	
Noseeum Mesh	0.5	-	>2psi	-	6.75	-	---
1.6 oz HyperD XL	1.6	---	65psi	+++	6.25	-	-

Fig. 3.8. Pugh Chart of Potential Envelope Materials

The HyperD nylon would be sewn together into the shape of the envelope using nylon string of similar material. As our tool of fabrication, the sewing machine was chosen, since it was something we had access to when fulfilling STR 11.0.0, Manufacturability.

3.2 Fabrication

This section will go over the fabrication process of the mechanical subsystems and will address the procedures, issues, and successes of the fabrication process. Additionally, before fabrication of the system was started, a CAD model was created with all subsystems designed to their stated specifications for both expected dimensions and weight. This can be seen in figure 3.9. This CAD was used to get early estimates for various force vectors used in various analyses, such as the forces fed into the control system, seen in Chapter 6, or the masses in the simulation, seen in Chapter 8. Once the CAD was finished, fabrication was ready to begin.

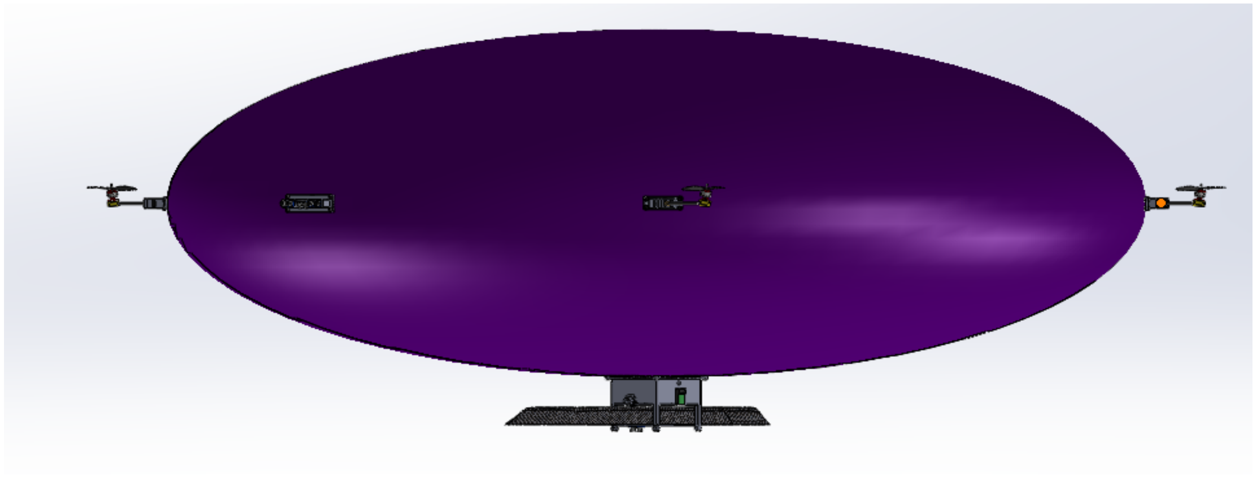
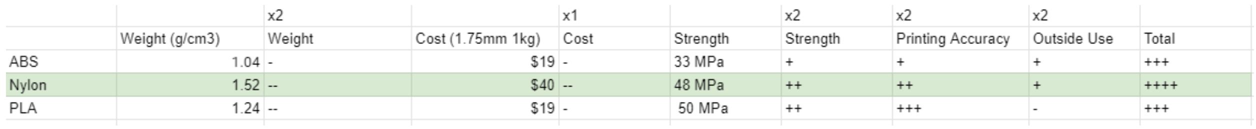


Fig. 3.9. Full Barone CAD

3.2.1 3D Printing

Since the ultrasonic array, propulsion mount, and gondola were all decided to be 3D printed. The 3D printing process needed to meet STR 11.1.0, 3D Printing: All 3d printed components should fit within dimensions of the Lulzbot Taz4 or Lulzbot Miniprinter beds. These printers were specified in the requirements as they were the printers we had access to. A Pugh chart was created with 3D printing materials shown in figure 3.10 to decide on a material to use for 3D printing. We found that all of the materials were more than strong enough to withstand the forces that the system would be encountering. Nylon was ultimately chosen due to its high print accuracy and resistance to sun damage.



		x2		x1		x2	x2	x2	
	Weight (g/cm3)	Weight	Cost (1.75mm 1kg)	Cost	Strength	Strength	Printing Accuracy	Outside Use	Total
ABS	1.04	-	$19	-	33 MPa	+	+	+	+++
Nylon	1.52	--	$40	--	48 MPa	++	++	+	++++
PLA	1.24	--	$19	-	50 MPa	++	+++	-	+++

Fig. 3.10. 3D Printing Materials Pugh Chart

The process of 3D printing was riddled with errors. First the relevant CAD models for the mounting brackets and plates were converted to STL files and placed into 3D printing software, Cura Lulzbot edition. This was the software we used due to its compatibility with the printers we had access to. Originally we were using the Lulzbot Taz4, to print our parts. This printer had a large bed and was able to print all our parts in one piece. However, multiple problems arose in the process. First, the nozzle repeatedly clogged due to previous users using incompatible printing materials. Once the nozzle was cleared, we tried to print our first parts; however, the parts would repeatedly get knocked off of the printer bed or begin to curl half-way through a print. We later found out that the printing bed was not level. We tested the evenness of the bed by using the 'paper test', where the nozzle would move across the entire area of a printing bed without making contact with a single sheet of paper between the bed and nozzle. The 'paper test' showed that on some corners of the bed the nozzle did not make any contact with the paper, but on other corners the paper was dragged across the surface of the bed by the nozzle. The results of the 'paper test' confirmed that the bed was not level. Although many hours were spent trying to level the printer bed, The Taz4's outdated manual leveling system, which had been broken at some point to our use of it, prevented us from fully leveling the bed with any further precision. Since we could not fix the unleveled bed a stronger glue was used to adhere the printed parts to the bed. With this quick fix,the majority of parts were printed on the Taz4, so the accuracy of the prints was not great, likely due to the unlevel bed and old nozzle. The ultrasonic mount, motor mounts, gondola, and all mounting plates were able to be printed. However only one of the propulsion mounts was able to be printed before the thermistor on the Taz4 hot end broke. This was confirmed by checking the resistance of the thermistor. At room temperature the thermuster should have a resistance of 90kΩ-110kΩ, when tested the resistance was found to be only 18Ω showing that it was broken, since we did not want to wait for a new hot end to arrive, the prints the remaining prints where done on a Lulzbot Mini that we had access too. Although the Lulzbot Mini's bed was smaller, it had a self bed leveling feature and a much cleaner nozzle, which resulted in high print accuracy. The Lulzbot

Mini was able to do the reminder prints without fail since the remaining parts to be printed were all small enough to fit onto its bed.

Despite many hardware bugs with the 3D printing process and low print accuracy on many of the parts, all parts were successfully able to be printed and we were able to move onto the verification process.

3.2.2 Envelope

The fabrication process of the envelope was one of the trickiest and most time consuming parts of this project. First, we needed to design a method to sew the fabric into the correct shape. This method needed to meet STR 11.3.0, Envelope. The envelope must be able to be cut with standard scissors and sewed with sewing machine Dylan owns, for dimensions to be within 5% of design when inflated. This was a difficult process as turning a 2D shape into a 3D one has many challenges. During research the process of goring was found. This is the process used to make globes out of many elliptical shapes. I was able to find out that the ellipse with our dimensions could be made with 12 gores each 122.5” tall and 28.5” wide. The curve of these gores would be the same as the curve of a 138.25” radius circle. These numbers were calculated using the trigonometry used in goring processes[18]. The dimensions of the gores are shown in figure 3.11.

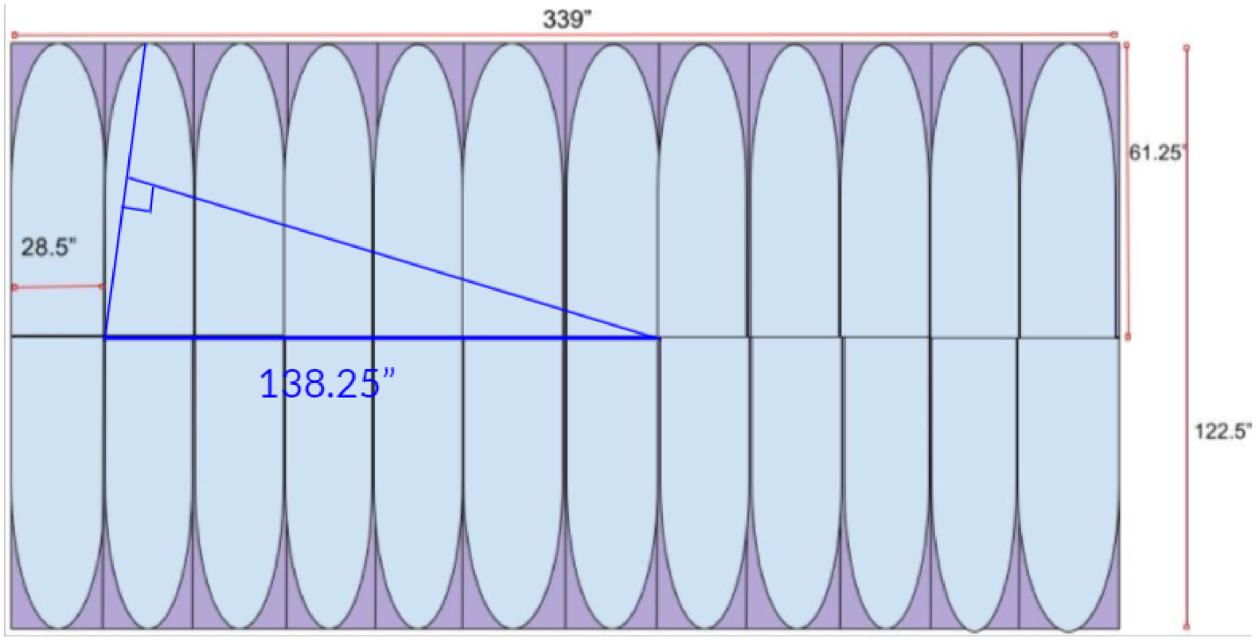


Fig. 3.11. Gore Dimensions for 108”x108”x40” Ellipsoid

Once the dimensions were found, pieces needed to be cut out from the 60'' tall roll of nylon. Since the sheet was only 60” tall, it was decided that 24 half gores of height 60” would be made rather than the 12 122.5” gores. The haves would then be attached together resulting in 12 120” full gores. The shorter height of the gores would result in a hole in both the top and bottom of the envelope. The hole on top would be patched using a fitted sheet of nylon and the hole on the bottom would be left in order to have access to the inside of the envelope for the purpose of attaching mounting plates and putting in the lift bag. Additionally an extra ¼" was left on either side of the sheets to account for loss of width in the sewing process. First,24 rectangular sheets were cut out. Then, a string of length 138.25" was used to trace the curve onto one of the sheets. This first sheet was used then cut out along the curves, and was then used as the trace for the rest of the sheets. The same trace was used for all the gores as to keep any error there may have been consistent. It would be better to have gores that are all wrong in the same way rather than gores with different errors. Once all 24 half gores were cut out 12 of the halves were sewn to the other 12. This resulted in 12 full gores that were the expected dimensions picture in figure 3.12.

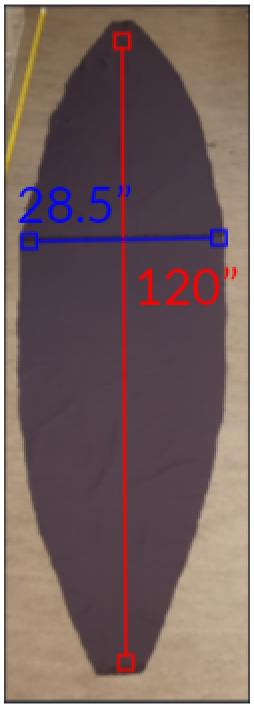


Fig. 3.12. One of 12 Gores.

Once all the gores were cut out the sewing process began. Since the team was inexperienced at sewing this part was riddled with mistakes. First it took many hours to figure out how to consistently properly thread the sewing machine. However, after many hours working with the machine multiple members of the team became proficient at threading the machine. The same was the case with the switch quality, although early on the quality of the stitches was questionable at best, resulting in uneven stitching. The quality of the stitches improved over the fabrication process. Once all twelve gores were sewn together the envelope seemed roughly the expected shape picture in figure 3.13. However the holes in the top and bottom were far bigger than expected. This error was thought to be caused by the poor stitch quality on some of the gores. Nevertheless the hole on the top was patched anyway and the envelope was ready to be tested.

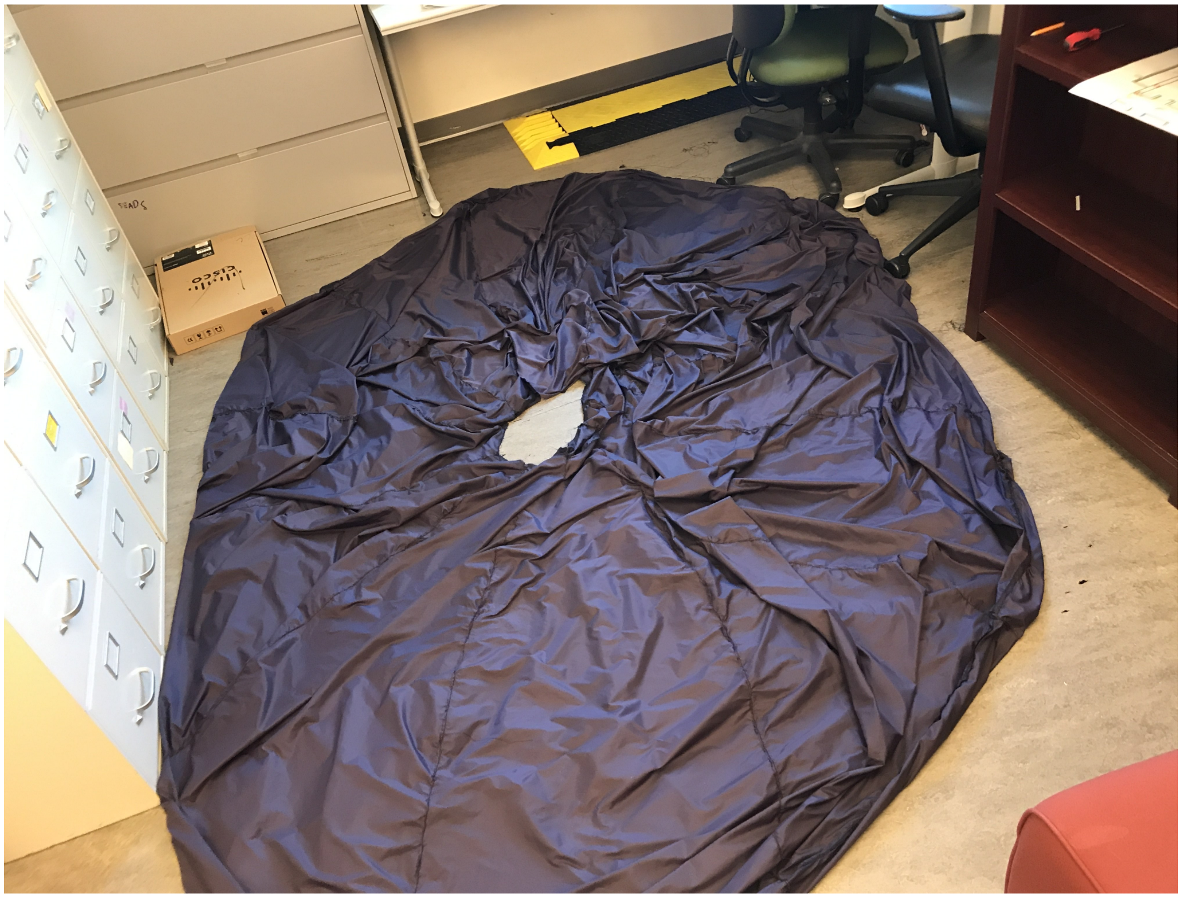


Fig. 3.13. All Twelve Gores Sewn Together Without Patching.

3.3 Mechanical System Verification.

Once the fabrication process was finished for both the frame related parts and envelope verification needed to be done to assure that they meet their intended specifications. Various tests were performed to analyse if they did meet their requirements. This section contains the test procedures and results for the verification processes.

3.3.1 Brackets and Mounting Plates

The mounting brackets for the ultrasonic array and propulsion system were easy to verify. Since the expected dimensions were known the parts were measured to see if they met specifications using rulers and calipers. Although both the ultrasonic mount and propulsion system mout were initially found to meet the expected dimensions shown in figures 14 and 15, it was found that the propulsion bracket did not account for the space needed for the servos wires. A second iteration was printed with an extra slot added to account for the wires. This new iteration was successfully able to hold the servo snugly picture in figure 3.16.

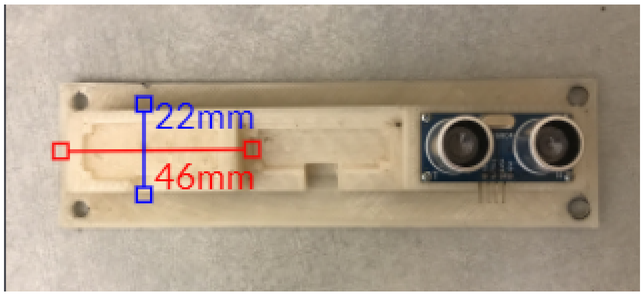


Fig. 3.14. Ultrasonic Array

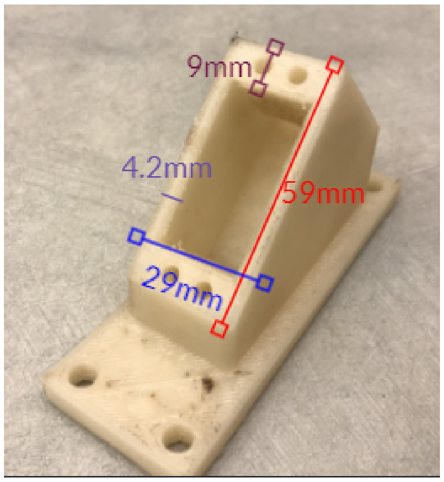


Fig. 3.15. First Iteration of Propulsion System Bracket

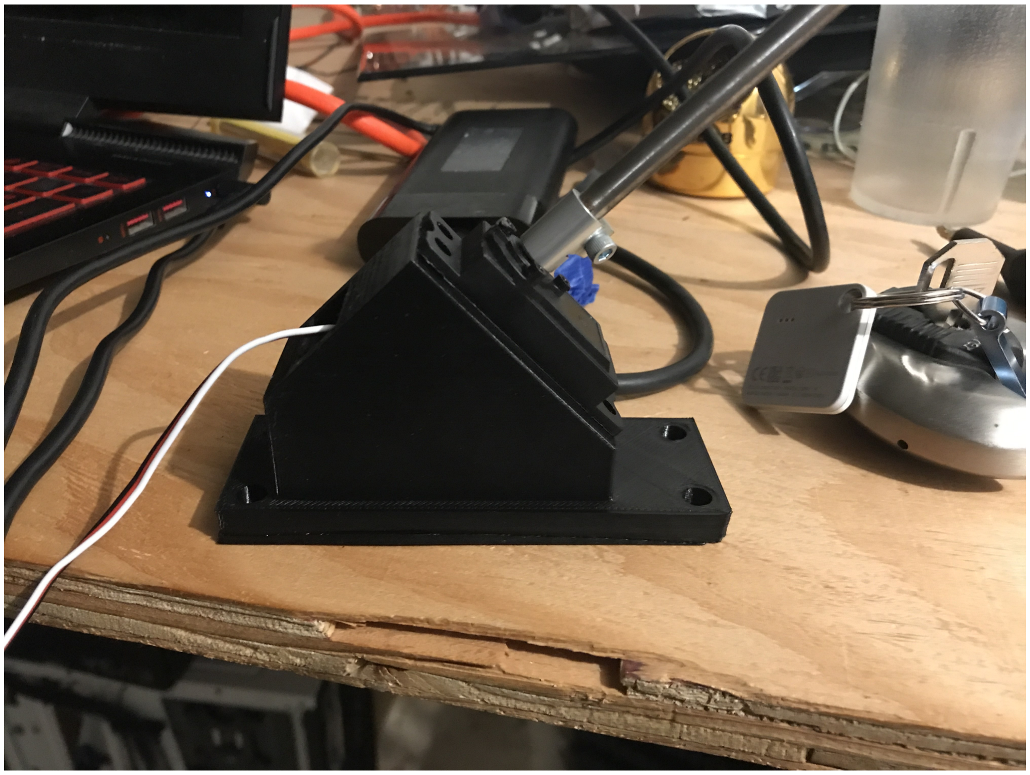


Fig. 3.16. Second Iteration of Propulsion System Bracket Holding Propulsion System

The results of the bracket measurements verified both the ultrasonic array subsystems ability to hold all three ultrasonic sensors and the propulsion subsystems ability to hold the servo. Additionally the servo coupler was successfully able to hold the shaft with the motor mounting block and motor attached to the servo, verifying the manufacturability of the propulsion subsystem. The dimensional accuracy of all the 3D printed parts verified STR 11.1.0, 3D printing.

3.3.2 Gondola

The next subsystem to be verified was the gondola. First, similar to the ultrasonic array and propulsion subsystems measurements were taken with a ruler and calipers to see that it met dimensional accuracy. Both the payload bay and housing basket were mesure to meet dimensional specifications shown in figure 3.17. During the barones first flight test the gondola was successfully able to hold a majority of the electrical system as intended further verifying its dimensions. The dimensional accuracy of the gondola further verified STR 11.1.0, 3D printing. Certain parts were not included in this test as they were not ready to be implemented into the system yet.

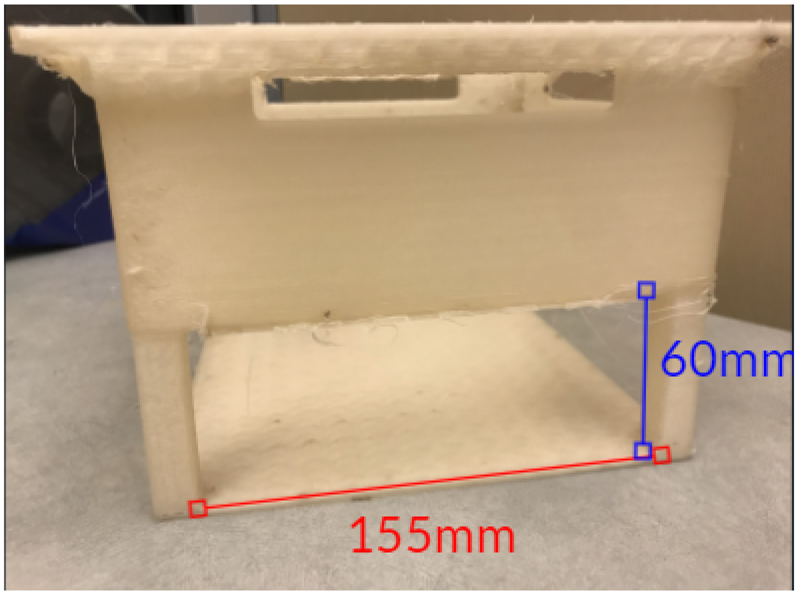


Fig. 3.17. Printed Gondola with Payload Bay Dimensions

Additionally, to verify the gondola, a heat analysis was performed in Solidworks to verify that the electronic components enclosed within the gondola did not break their maximum operating temperature. The results of the heat analysis showed a worst case maximum steady state temperature of 353° K or 80°C figure 3.18, below the maximum operating temperature of 100°C that most of the electronics on the board had. This heat analysis served as verification that the gondola could successfully house the electronics in its enclosed basket without the system overheating.

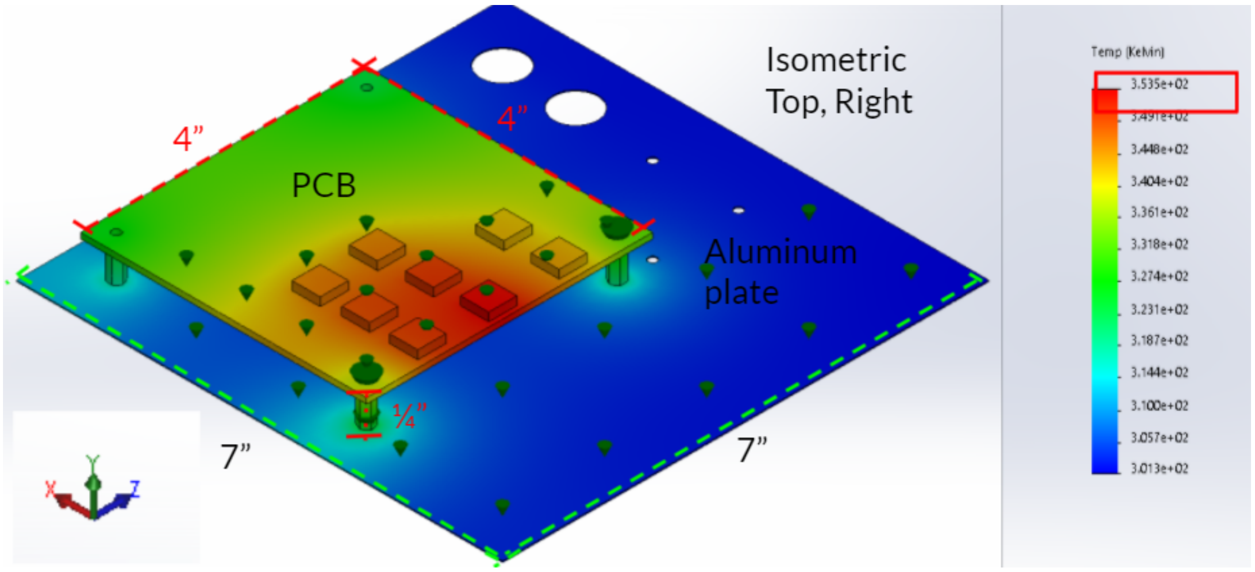


Fig. 3.18. Heat Analysis Test in Solidworks

3.3.3 Envelope and Lift Bag

The envelope and lift bag was the hardest subsystem to verify. To test the accuracy of its shape as well as usability to hold the propulsion system and ultrasonic array taut when inflated. Multiple inflation tests were performed with the procedure in figure 3.19 to verify it.

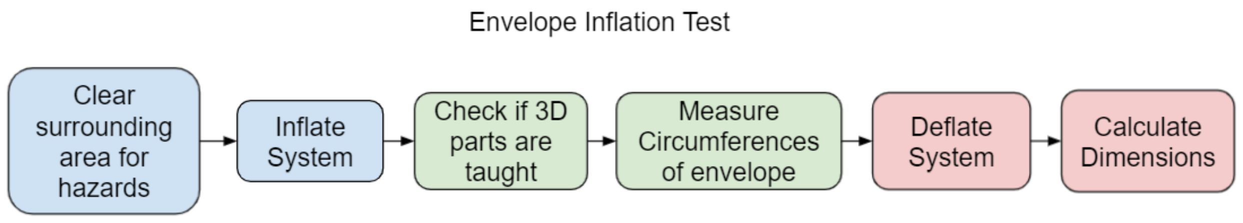

Envelope Inflation Test
Clear surrounding area for hazards
Inflate System
Check if 3D parts are taught
Measure Circumferences of envelope
Deflate System
Calculate Dimensions

Fig. 3.19. Procedure Flow Chart of Inflation Test

The dimensional results of the test revealed that the shape of the envelope was not correct. We believe this to be due to two factors. First, the errors in sewing resulted in an incorrect shape, and secondly, the stitching was not successfully able to force the balloon into an elliptical shape, since oftentimes in testing the stitching would begin to break rather than the balloon being forced down into the correct shape. Seven iterations of the test were done with the envelope patched in various ways, with the results of significant versions shown in table 3.1. V7 which was the best and final version is pictured in figure 3.20. Although the envelope was able to show some improvement in shape across multiple versions, we believe that a new envelope would have to be made from scratch in order to meet the dimensional requirements. In its current form the envelope will create 44N of drag in a 20MPH airspeed and will not be able to meet STR 2.0.0, Drone Speed. Additionally the failure to meet the dimensional specifications within 5% caused STR 11.3.0, Envelope, to fail.

Table 3.1.
Results of Inflation Test on Significant Versions.

Version	Circumferences	Diameters
Goal	255” & 339”	108x108x40”
V1	284” & 309”	98.4x98.4x81.7”
V5	269” & 308”	98x98x71.1”
V7	276” & 322”	102x102x70”

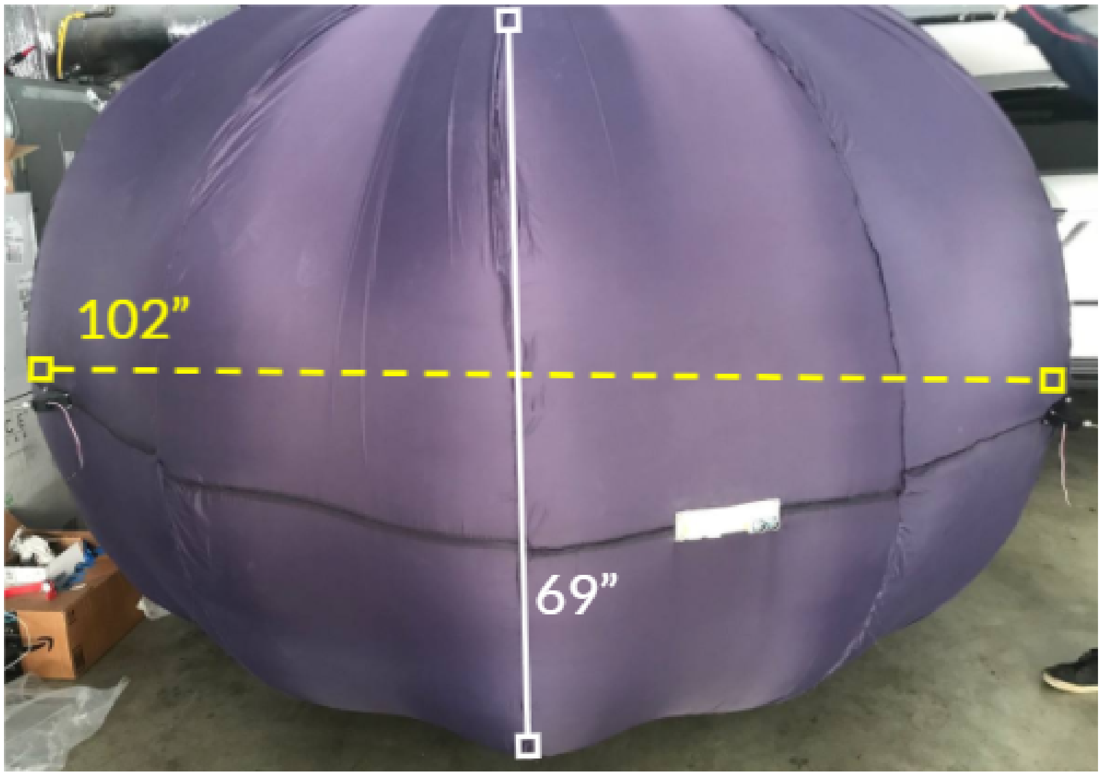


Fig.3.20. Inflated Envelope V7.

The inflation tests also checked to see if the brackets were successfully able to be attached to the envelope. Since the shape of the envelope was fabricated incorrectly the volume was estimated to be close to 6m^3 rather than the intended 4m^3 . In order to solve this problem air could be added before the helium so that the correct buoyant force was created while still filling the envelope until taut. In the first attempt to inflate the lift bag with the mounting plates and brackets attached. The lift bag was punctured when connecting with a sharp edge of one of the brackets. In order to fix this bubble wrap was applied to the inside mounting plate on the envelope and this was found to fix the puncturing problem found in the first test. Another issue found was that one of the propulsion system brackets was not correctly held taut, even when fully inflated; this is shown in figure 3.21. This slack on the propulsion system was believed to have caused more fabrication errors causing an area of the envelope to have excess material so that even when fully inflated the section pictured in figure 3.21 was not held taut like the other 3 propulsion systems.

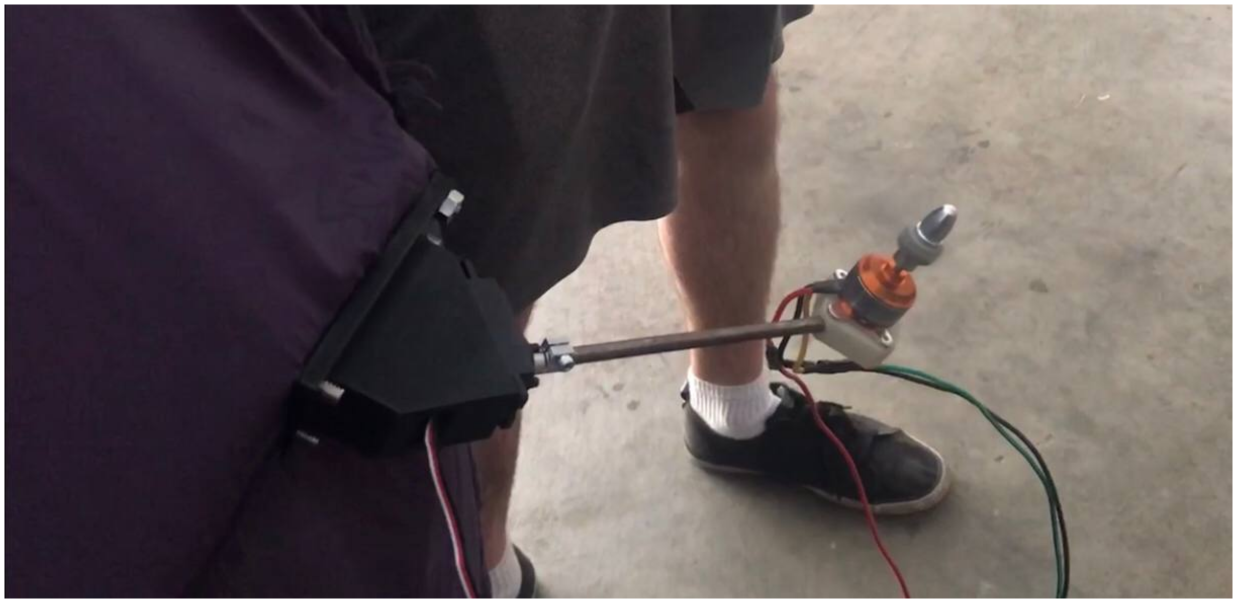


Fig.3.21. Propulsion System With Slack.

It was shown in inflation tests that when fully inflated three of the four propulsion systems as well as the ultrasonic array were held taut, an example of a taut propulsion system can be seen in figure 3.22. The ability to attach all subsystems together was also verified during an initial flight test as seen in figure 3.23. For more information about the flight test see Chapter 9.

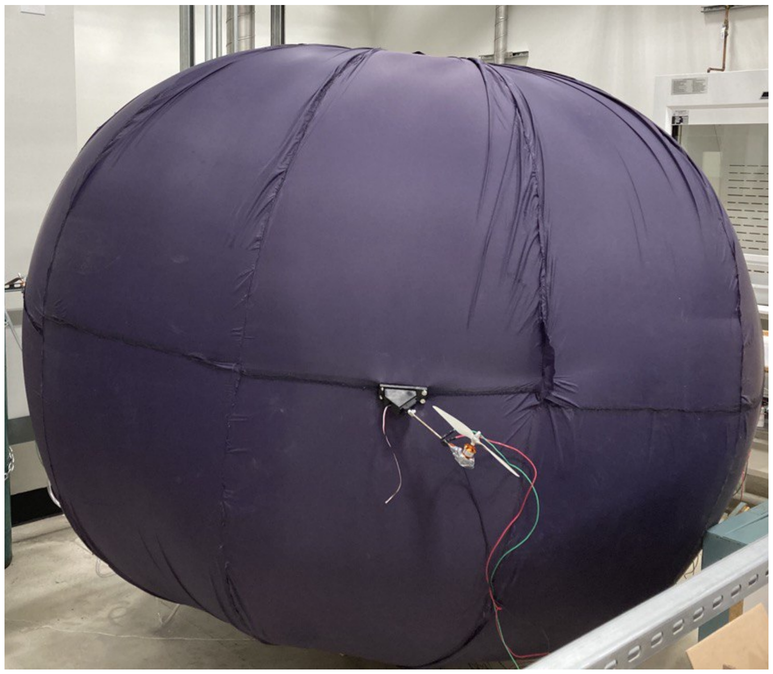


Fig.3.22. Propulsion System Held Taut During Inflation

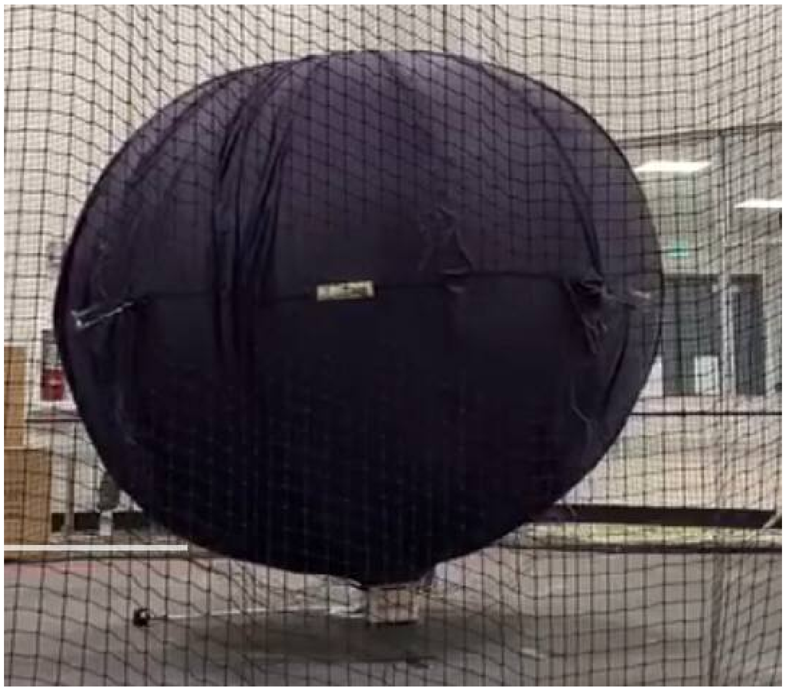


Fig.3.23. Full system assembled during first flight test.

3.4 Conclusion

After designing the various parts of the mechanical systems of The Barone, based on the technical requirements, Fabrication was carried out on all parts of the mechanical system. Then verification of these mechanical subsystems was carried out. The verification found the ultrasonic array, propulsion system, and gondola all meet their required dimensional specifications. This verified STR 11.1.0, 3D printing. The verification process of the envelope found that the envelope shape was not met, with the envelope being far too tall. Since the error in the envelope dimensions was greater than 5% STR 11.3.0, Envelope, failed to be met. The failure of the envelope to meet specification increased the drag our system experienced making it able to meet STR 2.0.0, Drone Speed.

Chapter 4: Propulsion Design and Actuator Interface

In this chapter we first discuss each of the major parts of the propulsion system and its interface, the specifications they were designed to meet, and the reasoning behind those specifications. Then the process of implementing the interface. Finally the process of verifying these parts and the results of the verification will be analysed.

4.1 Parts Selection of the Propulsion System

In order to control the barone, a propulsion system capable of meeting STR 2.0.0, Drone Speed, STR 3.0.0, Remote Control, and STR 4.0.0, Autonomous, needed to be created. In order to steer the drone effectively with its buoyant moment created by the helium lift bag the propulsion system picture in figure 4.1 was designed. For more information about the buoyant moment see section 2.2. The propulsion system consisted of a motor holding a propeller at the end of a shaft connected to a servo. This servo would turn the motor in order to change the direction of force created by the propulsion system. The motor is also connected to an ESC housed in the gondola of the Barone. This ESC sends the necessary signal to the motors to control their speeds. Both the ESCs and the servos are also connected to a microcontroller also housed in the gondola; the microcontroller provides output signals to both the ESCs and Servos based on input from the remote control transceiver. The output signal from the microcontroller controls the angle of the servo as well as the signal the ESC is sending the motor. In this section the selection process and design decisions around the parts in the propulsion system will be discussed.

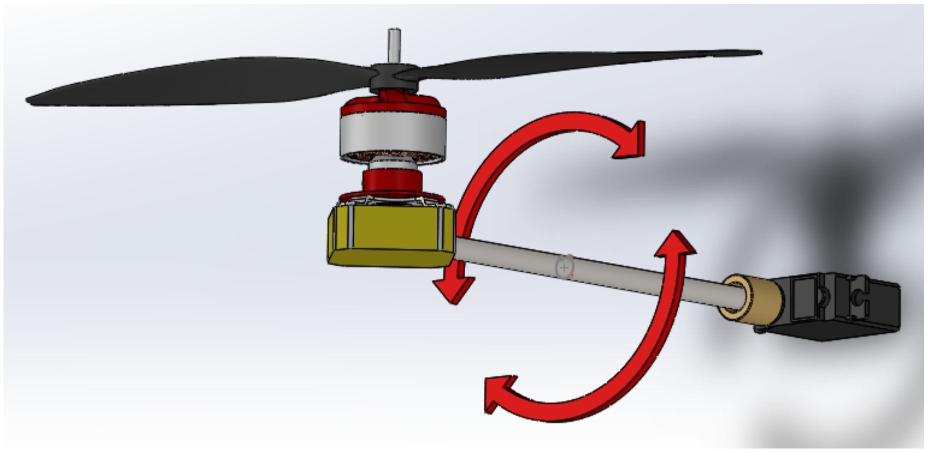


Fig.4.1 The Barone's Propulsion System

4.1.1 Actuator selection

First a 9" propeller was chosen, since the energy the motor needs to put into spinning a larger propeller to produce the same amount of thrust as a smaller one is significantly less, and thus larger propellers are more efficient than smaller ones. This is because the kinetic energy of a rotation object is equal to $\frac{I\omega^2}{2}$ [16]Additionally kinetic energy of the displaced air can be represented by $\frac{mv^2}{2}$[14]. We choose not to go above 9" however in order to keep the weight of the propeller relatively low. Both to meet STR 1.2.0, Weight, and reduce the force from the lever arm on the envelope, to learn more about the envelope see Chapter 3.

Next a motor was chosen based on the required throttle needed. First the required RPM needed from the motors was calculated. Since the maximum expected weight of the system was to be 5N, this was the required minimum force all four propellers needed to provide in order to get the system airborne. Using the Force equation for a propeller $F_x = \rho n^2 D^4 c_T$(4.1)where ρ is density, n is RPM, D is Diameter of Propeller, C_T is the Thrust Coefficient, and J is Advanced Ratio[19]. C_T was found using software from aerodynamics4students.com[20] suggested to us by Professor Gabrial Elkaim. Equation 4.1 was placed into a Matlab simulation the result of which is shown in figure 4.2 and it can be seen that in order for our propeller to produce 1.25N of thrust they must be spinning at 7400 RPM.

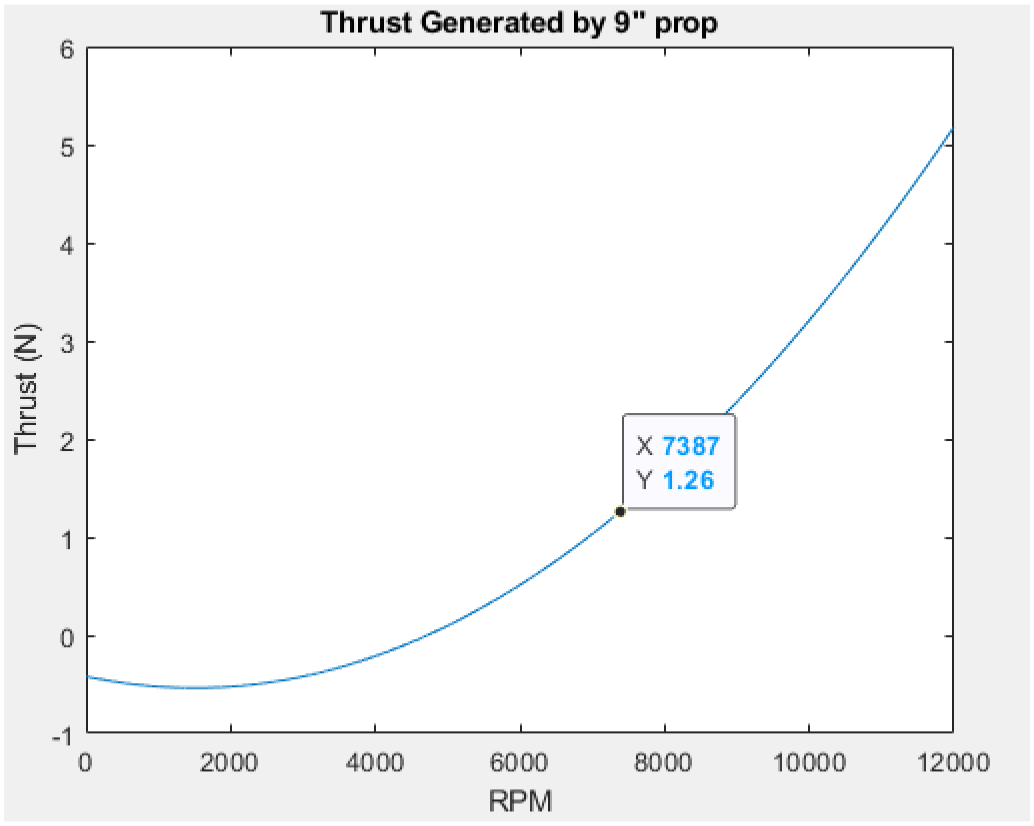


Fig.4.2 Matlab Graph of RPM vs Thrust of Our Propeller

Now that the required RPM was known a motor could be selected. A motor with a low Kv was chosen as lower KV motors have higher torque ratings and can spin larger loads with less RPM loss. Since our propeller was expected to weigh around 10g this made sense. We then needed to see that our motor would not hit its stall current while spinning the propeller at the required 7400 RPM. The expected torque needed from the motor to reach the required RPM with the propeller was found using equation $M_x = \rho n^2 D^5 C_Q$ (4.2)where ρ is density, n is RPM, D is Diameter of Propeller,C_Q is the Torque Coefficient[19]. C_Q was found using the same software as C_T in the previous section [4]. Equation 4.2 was used to find that in order to spin our propellers at 7400, 77.43Nm of torque would be required from the motor. This torque was multiplied by the K_T rating of the motor which represents the Nm of torque it can provide per amp of current drawn. The SK3 2822-1275kv Brushless motor was selected as based on its $7.49\frac{Nm}{A}e - 3$$K_T$ rating. The expected current draw of the motor with the propeller was 0.6A at 7400 RPM with our propeller well below its 8A stall current. Additionally it had a weight of only 30g, relatively low for a motor of this size, which helped meet STR 1.2.0, Weight, as well STR 2.0.0, Drone Speed by providing the required force in order to move our system.

Racerstar RS20A BLheli_S 4-in-1 ESC was the ESC chosen to control our motor. The Racerstar 4 in 1 ESC was chosen as each of its 4 ESCs were rated for 20A, the same rating suggested by the motors datasheet. Additionally the 4 in 1 ESC was only 25g much lighter than 4 individual ESCs or even other 4 in 1 ESCs, this also helped to meet STR 1.2.0, Weight.

Finally the servo was selected. The servo needed to be able to provide the required torque to move the rest of the propulsion system without hitting its stall torque, Additionally the servo needed to take in positional data for more than one full rotation or 360° based on the needs of our control system. See Chapter 6 for more information on the controls system. First the required torque needed to be found. This was done by using $\tau = la$a standard torque equation. Where l is the moment of inertia and a is the angular acceleration. Using Solidworks the moment of inertia was estimated to be 2.12kgm2 x 10-4. Then the angular acceleration of the servo was found using the power and RPM of the motor by using $a = \frac{\tau}{l}$where l was the moment of inertia and the τ was the stall torque of the motor found with $\tau = \frac{30p}{\Pi n}$, using the servos max power specs the RedCon 360 Degree Digital Metal Gear HV Servo was initially chosen since it had an angular acceleration of $687\frac{Rads}{s^2}$. The angular acceleration gave us the expected worst case toque the servo would need to apply as 0.143NM. Since this servo had a stall torque of 0.2NM it was deemed as an acceptable choice of servo. Once the servo was received and tested it was revealed that it did not take positional data as intended and was a continuous motion servo that functioned more like a motor that takes angular velocity data rather than positional data. A new servo The RC Sail Winch Servo 25T was chosen as it had a much higher stall torque of 1NM found using the same technique as the original.This servo also was able to take positional data over multiple rotations, up to 5.5 full turns.

4.1.2 Interface Selection

In order to fulfill STR 3.1.1, RC Controller, a remote controller had to be chosen that is capable of providing all necessary commands including forward, turn, ascend, and descend. It had to control the servos and motors, a remote controller and remote controller receiver were chosen. The remote controller had to transmit at least four channels for throttle, yaw, pitch, and roll commands, and the receiver had to receive at least four channels. The remote controller chosen is a 10-channel FlySky FS-i6, while the receiver is a 6-channel FlySky FS-iA6, so at most the controller could send out ten unique commands, but the receiver could only receive six of them, but this was sufficient as we only needed a minimum of four channels for throttle, yaw, pitch and roll commands. This fulfills the requirement for the remote controller.

4.2 Propulsion System Implementation

4.2.1 Remote Controller and Receiver Integration

In order to meet STR 3.1.2, Software for System Response of RC Control, the inputs from the remote controller and receiver had to be integrated with the PIC32.

Each channel from the receiver was outputted as a 50Hz signal with duty cycle ranging from 1ms - 2ms depending on the tilt of the joystick on the controller, and thus could be read by the PIC32MX340F512H (referenced as PIC32 for convenience) microcontroller's Timers and Input Capture functionality in order to read the signals from the remote controller receiver.

The PIC32's Timers are essential in time-sensitive events that can be detected by Input Capture, and can be used to set timers and trigger interrupts when those timers go off. The most important parts of the Timer are its pre-scalar and period register, or PR. Each Timer uses the Peripheral Clock of the PIC32, which is 40MHz, in order to increment its counter, and the user can configure how many cycles of the Peripheral Clock must go by in order to increment the Timer counter by one, also known as the pre-scalar. The user can configure between 1:1 pre-scalar, which means the Timer counts along with the Peripheral Clock at 40MHz, or a max of 1:256 pre-scalar, which means the Timer only increments by one after 256 cycles of the 40MHz Peripheral Clock go by. Then the user sets what the max value of the Timer should be before it goes off, which is known as the period register. For example, if a user wanted to set a 20ms/50Hz timer, you would first need to decide what pre-scalar you would like, which determines your max PR, and then you would calculate the PR.

For a 20ms/50Hz timer, with a pre-scalar of 1:256, the Timer's PR would be (20ms / 40MHz) / 256, which would give a PR of 3125[67].

Input Capture is also one of the pieces of hardware on the PIC32, and can play a very important role in measuring the length of a signal. Essentially, an Input Capture pin tracks the state of a pin, triggers an interrupt when the pin reaches a certain condition, and "captures" the value of the Timer it is connected to. In the case of the remote controller receiver, the Input Capture pin would need to capture the value of a Timer when the signal pin from a channel went high, and then once again when it went low, and subtract the Timer values to find the total amount of time that had gone by[68].

Figure 4.3 is a sample flowchart of how a standard run with the PIC32 and remote controller receiver goes:

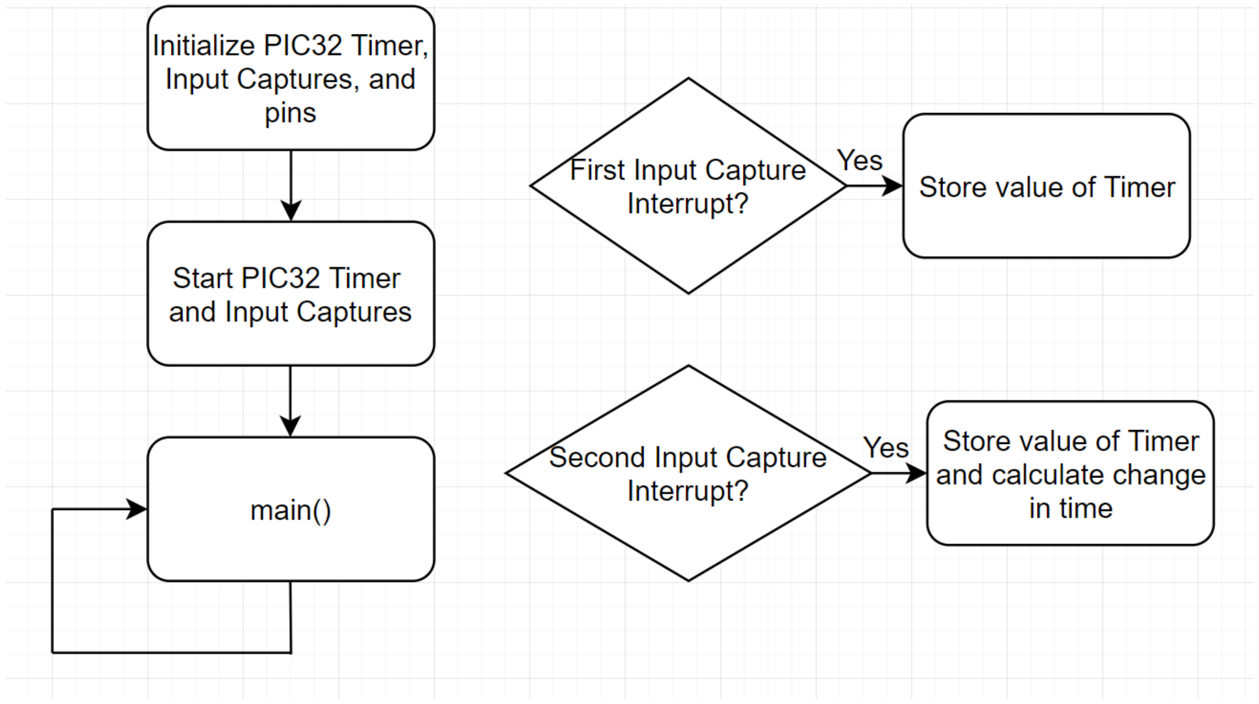

Initialize PIC32 Timer, Input Captures, and pins
Start PIC32 Timer and Input Captures
main()
First Input Capture Interrupt?
Yes
Store value of Timer
Second Input Capture Interrupt?
Yes
Store value of Timer and calculate change in time

Fig. 4.3. Sample Run with PIC32 and Remote Controller Receiver

In the initialization phase, Timer 2 is initialized to be 20ms/50Hz by setting Timer 2's pre-scalar and PR to 256 and 3125 respectively. Four Input Captures are used for the four channels from the receiver with Timer 2 as their Timer source, and configured so that each triggers an interrupt on every rising and falling edge of their respective channel pin. Each pin is configured as an input, with a starting value of 0. The Timer and Input Captures are then started, and the program enters the main loop. When a signal from the remote controller is sent to the receiver, the receiver outputs a signal with a rising edge and triggers the first Input Capture interrupt, and the user reads the value of the Timer the Input Capture captured. When the signal goes back low, the second Input Capture interrupt triggers, and the user once again reads the value of the Timer, and subtracts the first Timer value from the second to find the change in time of the signal. However, the values of the Timer are not equivalent to real-time, since they run off of the Peripheral Clock. To get this time in real-time, you need to multiply by the real-time you set the Timer to, which would be 20ms/50Hz in this case, and divide by the PR of the Timer, 3125, to get this time in milliseconds.

4.2.2 Servos, ESC, and Motors Integration

The servos and ESC, which took in a 50Hz signal and converted to a 24kHz signal for the motors, used the same input signal shown in the timing diagram in Figure 4.4:

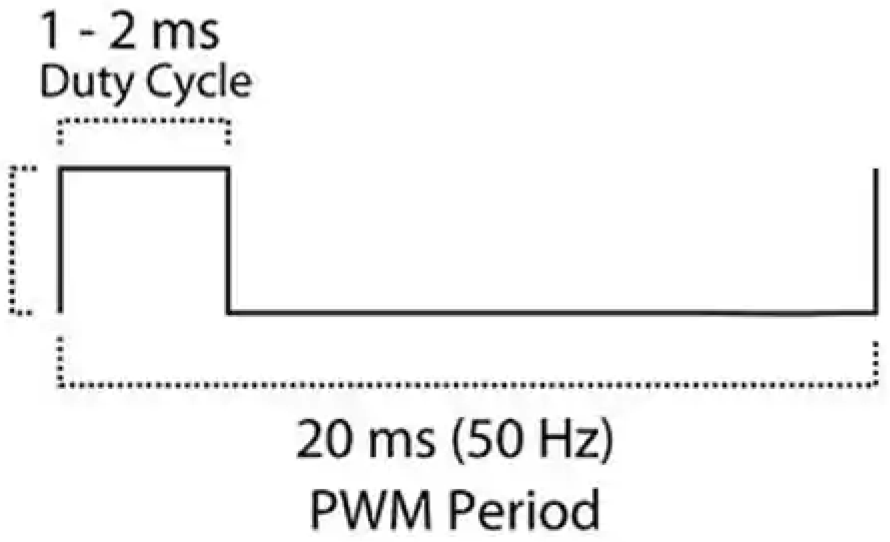


Fig. 4.4. Servo Timing Diagram

As the diagram shows, a 20ms/50Hz signal with a varying duty cycle between 1ms - 2ms needed to be produced from the microcontroller, which could have been done using the PIC32's Output Compare functionality.

The PIC32's Output Compare is another important piece of hardware that can be used in time-sensitive events to output a precise signal. Essentially, Output Compare works by outputting a signal set by the user, either PWM or constant high or low, when the Timer it is connected to reaches a certain value (see earlier for a description about Timers). The most important parts of the Output Compare are its mode, compare register, and compare register secondary. On initialization, the mode for Output Compare is chosen, which determines its behavior when the Timer reaches a certain value. This could be changing the signal from high to low or vice versa, or turning a duty cycle on or off. The compare register is the value the Output Compare will run until, also known as the duty cycle, and cannot be changed, after which it will perform the behavior determined by the mode, while the compare register secondary is the next value chosen by the user that will be loaded into the compare register in the next cycle. For example, for a PWM signal, the mode is chosen as being PWM, and for a 50% duty cycle, the compare register is given a value of 1562 when connected to a Timer with a PR of 3125. If the user wanted to change this duty cycle, they would change the value of the compare register secondary[69].

Figure 4.5 is a sample flowchart of how a standard run with the PIC32 and servos/ESC goes:

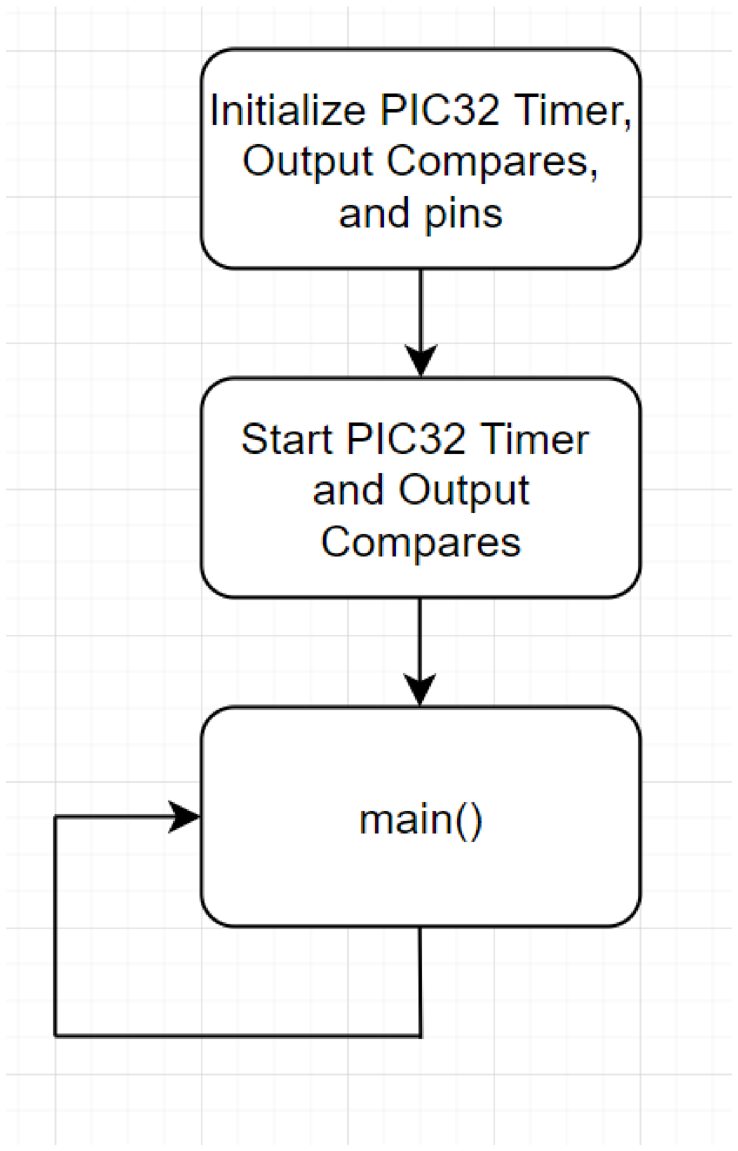

Initialize PIC32 Timer, Output Compares, and pins
Start PIC32 Timer and Output Compares
main()

Fig. 4.5. Sample Run with PIC32 and Servos/ESC

In the initialization phase, Timer 2 is initialized to be 20ms/50Hz by setting Timer 2's pre-scalar and PR to 256 and 3125 respectively. Two Output Compares, one for the servos and one for the ESC, are initialized to PWM mode with a starting duty cycle of 0%. The output signal pins to the servos and ESC are configured as outputs, and are given a starting value of 0. The Timer and Output

Compares are then started, and the program enters the main loop. In the main loop, if the user wants to change the duty cycle of the Output Compares, they change the value of the compare register secondaries. With Timer 2 having a max value of 3125, 3125 / 4 gives 25% duty cycle, 3125 / 2 gives 50% duty cycle, and 3125 gives 100% duty cycle.

To verify the behavior of the Output Compares for the servos and ESC/motors, the Output Compares were observed on an oscilloscope as seen in Figure 4.6.

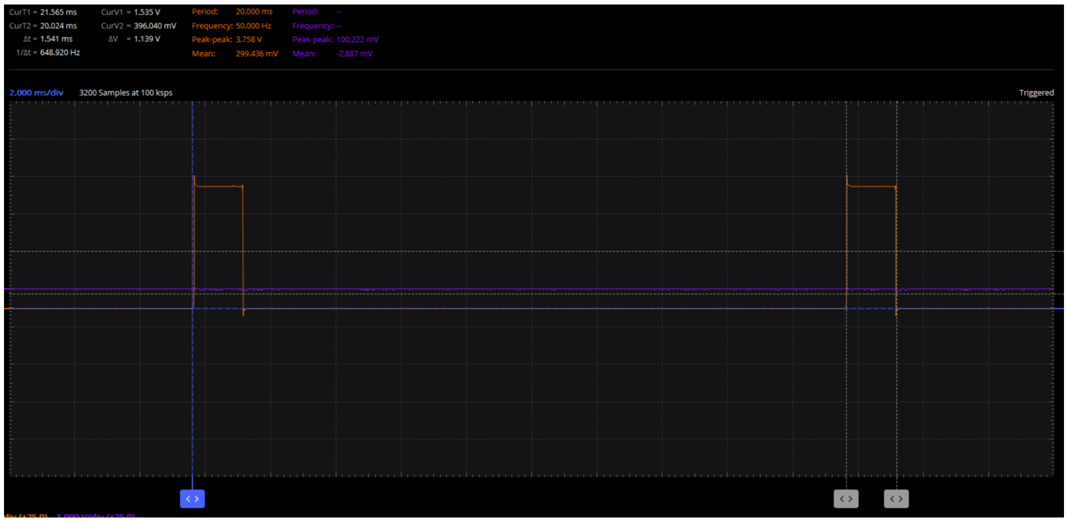


Fig. 4.6. 20ms/50Hz Frequency, 1.5ms Period Observed on an Oscilloscope

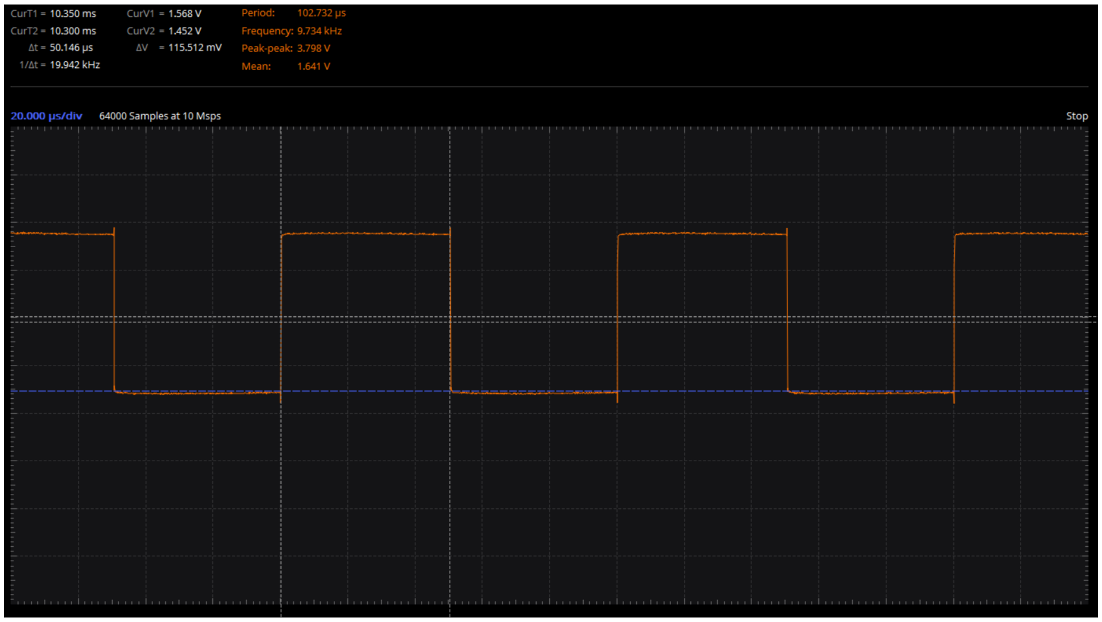


Fig 4.7. 10kHz, 50% Duty Cycle Signal Observed on an Oscilloscope

In this example, the frequency of the signal should have been 20ms/50Hz, with a duty cycle of 1.5ms, as seen in the top left corner of Figure 4.6. The signal was measured to be around 20.024ms, with a change in time of duty cycle of 1.541ms, so the Output Compares correctly outputted the right signal. This verified our implementation of Output Compare. In Figure 4.7 it can be seen that a 20kHz frequency signal with 50% duty cycle was successfully achieved when the Output Compare was configured to be half the value of the Timer it was connected to.

4.2.3 Remote Controller and Receiver, Servos, ESC, and Motors Integration

To use the remote controller receiver, servos, ESC, and motors in conjunction with each other such that the remote controller directly controls the servos and motors, the PIC32's Input Captures and Output Compares were utilized as described in the previous section. Figure 4.7 is a sample flowchart of how a standard run with the PIC32, remote controller receiver, servos, ESC, and motors goes:

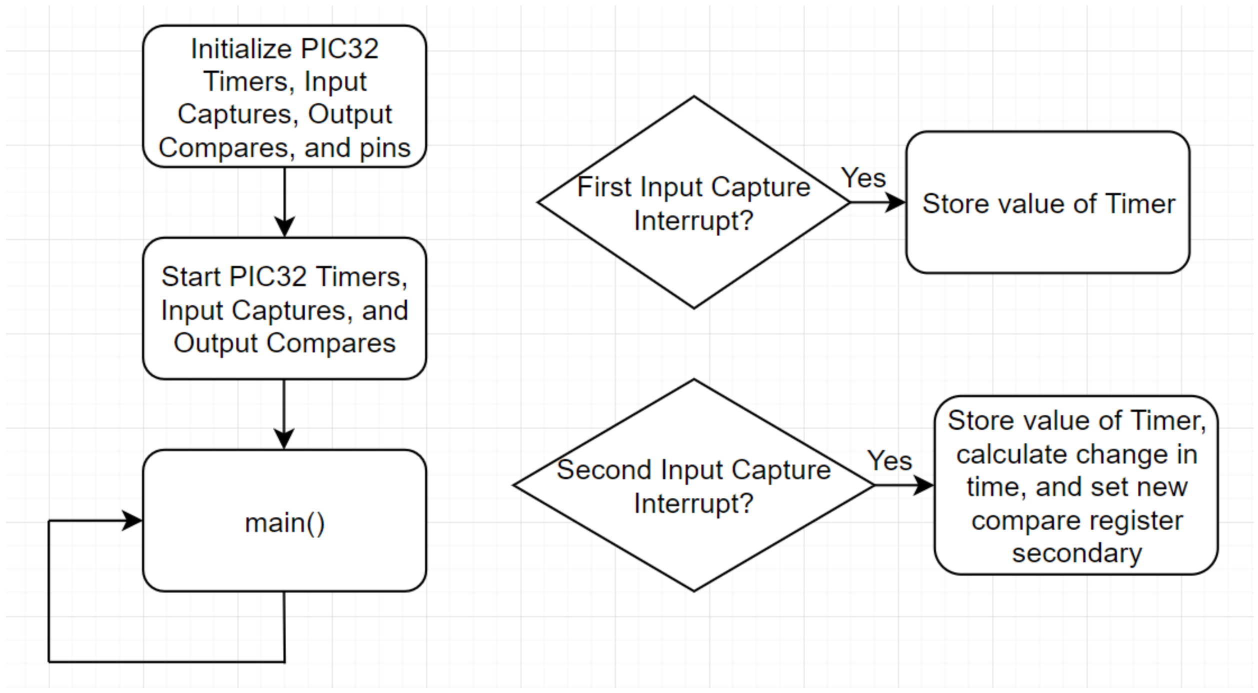

Initialize PIC32 Timers, Input Captures, Output Compares, and pins
Start PIC32 Timers, Input Captures, and Output Compares
main()
First Input Capture Interrupt?
Yes
Store value of Timer
Second Input Capture Interrupt?
Yes
Store value of Timer, calculate change in time, and set new compare register secondary

Fig. 4.7. Sample Run with PIC32 and Remote Controller, Receiver, Servos, and ESC

In the initialization phase, Timer 2 is initialized to be 20ms/50Hz by setting Timer 2's pre-scalar and PR to 256 and 3125 respectively. Four Input Captures are used for the four channels from the receiver with Timer 2 as their Timer source, and configured so that each triggers an interrupt on every rising and falling edge of their respective channel pin. Two Output Compares, one for the servos and one for the ESC, are initialized to PWM mode with a starting duty cycle of 0%. Each Input Capture pin is configured as an input, with a starting value of 0, while each Output Compare pin to the servos and ESC are configured as outputs, and are given a starting value of 0. The Timer, Input Captures, and Output Compares are then started, and the program enters the main loop. When a signal from the remote controller is sent to the receiver, the receiver outputs a signal with a rising edge and triggers the first Input Capture interrupt, and the user reads the value of the Timer the Input Capture captured. When the signal goes back low, the second Input Capture interrupt triggers, and the user once again reads the value of the Timer, and subtracts the first Timer value from the second to find the change in time of the signal. In this same interrupt, the change in time of the signal from the Input Capture is directly used as the compare register secondary value, essentially forwarding the signal from the Input Capture pin to the Output Compare pin. For example, if in the second Input Capture interrupt, the change in time was found to be 156, this is the value one of the compare register secondaries would be assigned in order to replicate the same signal as an output.

In the current implementation, the same one signal is given for all the servos, and another same signal is given to the ESC for all motors. This is why only two Output Compares are needed, one for the servos and one for the ESC. However, the goal was for a more complicated control system where the servos would be at different angles from each other, as well as the motors having different speeds from each other.

4.3 Propulsion System Verification

To verify the controls system with the remote controller receiver, servos, ESC, and motors, the physical setup viewed in figure 4.8 was implemented:

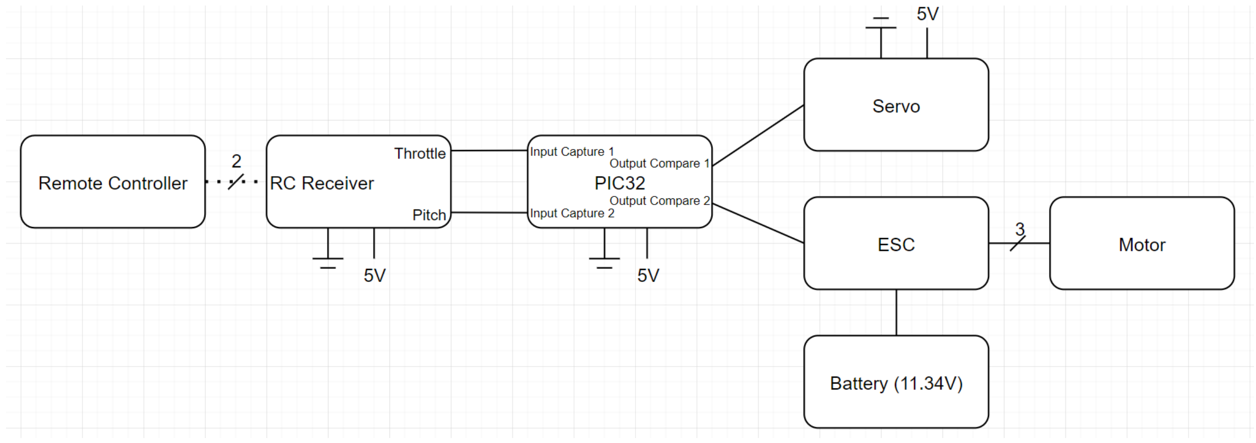


Fig. 4.8. Physical Setup with the PIC32, Remote Controller, Receiver, Servos, ESC, Motor, and Battery

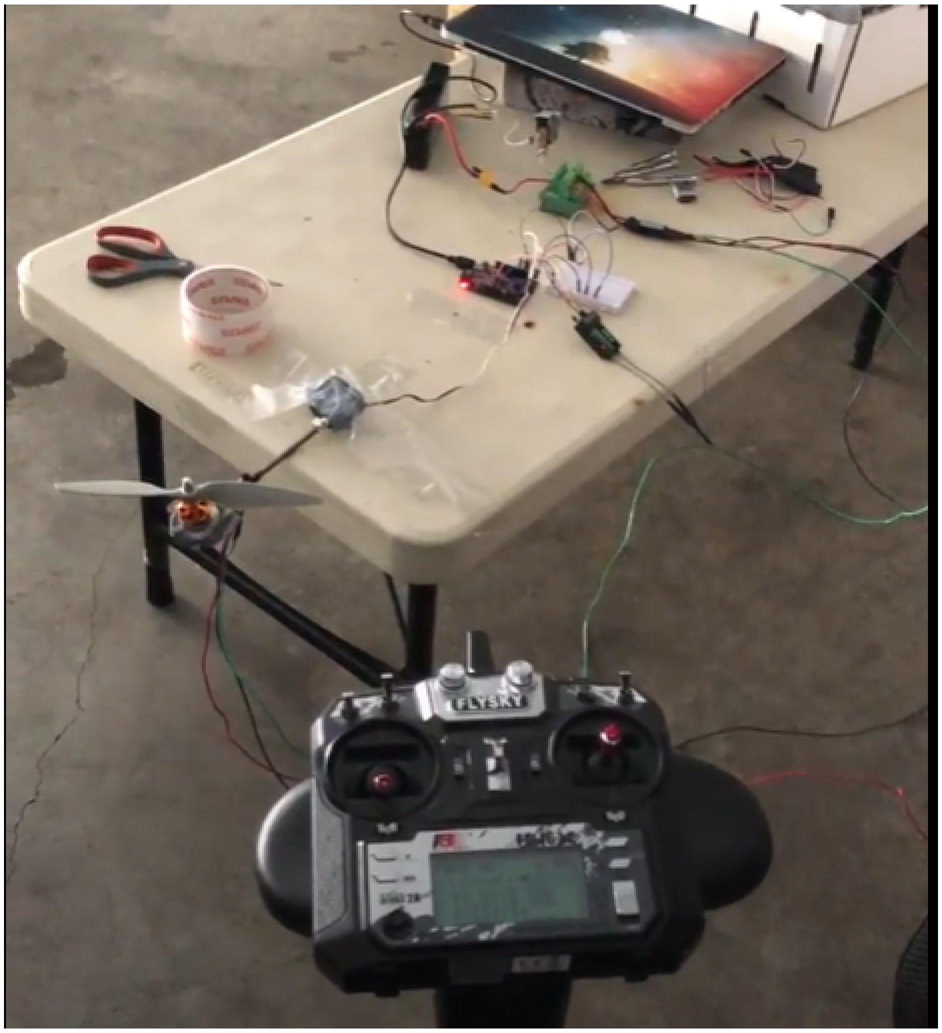


Fig. 4.9. Physical Verification of the PIC32, Remote Controller, Receiver, Servos, ESC, Motor, and Battery

With a shaft connected to the servo, and the motor with a propeller on the end of the shaft, the servo was able to rotate in either direction and the motor was able to spin at variable speeds in response to the remote controller. STR 3.1.2, Software for System Response of RC Control, has made progress in the form of controlling one servo and one motor, but is not complete until all servos and motors.

4.4 Conclusion

In this chapter we were successfully able to design, implement and test our propulsion system and its interface in order to attempt to meet STR 2.0.0, Drone Speed, STR 3.0.0, Remote Control, and STR 4.0.0, Autonomous. Although the final verification of the propulsion system for STR 2.0.0, Drone Speed, can be seen in Chapter 7. Here we were able to show progress on STR 3.1.2, Software for System Response of RC Control, though the implementation of Input Capture and Output Compare interrupts communication protocols between the remote controller, servos, and microcontroller, we were able to show that a single propulsion system can respond to user inputs.

Chapter 5: Sensor Array, State Estimation, and PCB Interface

The onboard sensor array of the drone includes an altimeter sensor for high altitude tracking, an IMU sensor for crash detection and positional state estimation, a GPS module for autonomous path following, and a pressure sensor for helium lift bag leakage detection. The sensor array communicates with the PIC32 microcontroller via I2C, SPI, UART, Input Capture, and Output Compare pins. The sensor array is built onto a single PCB to enable PCB interface between sensors and the microcontroller and processor where the processor can calculate the state. Voltage regulators are implemented within the PCB to distribute power from the drone’s onboard battery to the onboard ICs with appropriate voltages. Analog electronics, such as the 3 phase motors, transmitters, and receivers, are connected to the PCB via pinouts to reduce EMF interference with the onboard digital sensor array. The PCB is custom designed for our drone to integrate the sensors, voltage regulators, Raspberry Pi microprocessor ICs, and PIC32 microcontroller into one board, effectively reducing the drone mass and reducing electrical clutter.

5.1 Sensor Array

5.1.1 HCSR04 Ultrasonic Sensors

To satisfy STR 4.2.1, which was to use the ultrasonic sensors to detect constant height of 1m, the HCSR04 was chosen. The ultrasonic sensors would be used to detect distance from the drone. To configure the ultrasonic sensor, the following timing diagram from Elecfreaks was used as reference[70]:

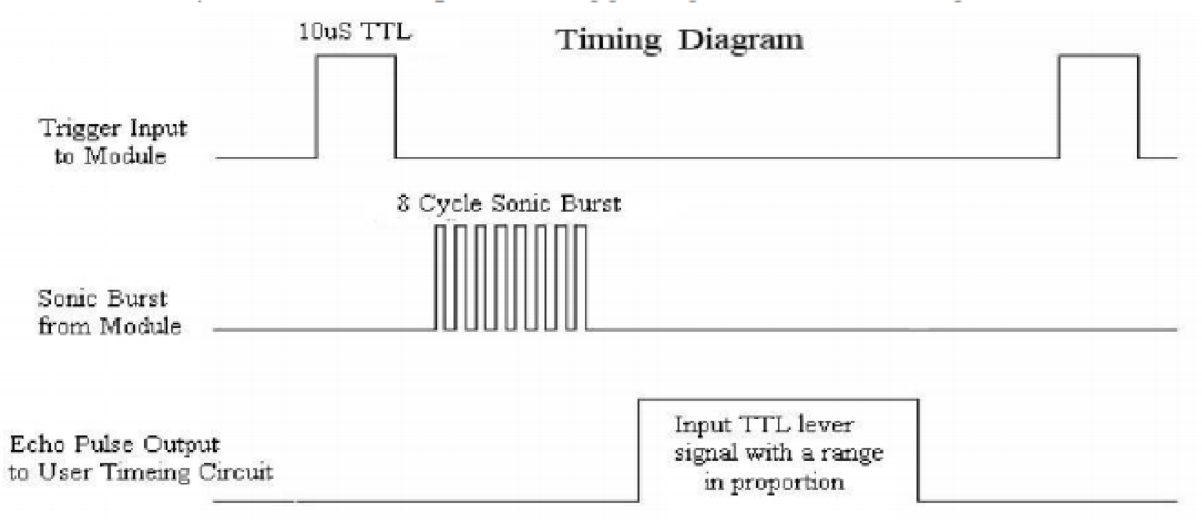

10uS TTL
Timing Diagram
Trigger Input to Module
8 Cycle Sonic Burst
Sonic Burst from Module
Echo Pulse Output to User Timeing Circuit
Input TTL lever signal with a range in proportion

Fig. 5.1. Ultrasonic Timing Diagram

To begin, the microcontroller would send a 10us pulse to the trigger input of the ultrasonic sensor, which is shown in the first signal of the timing diagram. Following this, the sensor would send out an 8 cycle sonic burst at 40kHz, shown in the second signal of the diagram, and immediately raise its echo pin, as shown in the third signal of the diagram, to indicate the sensor sent out a ping. Once that ping hit something and returned to the sensor, the sensor would lower its echo pin. By connecting the echo pin to the microcontroller and using the microcontroller to time the length at which the echo pin remained high, the amount of time the ping took to get back to the sensor could be determined. Knowing this and the speed at which sound travels, 343m/s, the distance of the object from the sensor could be calculated by multiplying the time, divided by 2 since the ping both travels to the object and back, by the speed. Between each trigger input from the microcontroller, 60ms cycles were recommended by the manufacturer in order to prevent a new input signal from going out to the sensor before the old echo signal came back.

In order to implement this into software, the PIC32's Timers and Input Captures were utilized (see Chapter 4 for description on Timers and Input Captures). With the ultrasonic sensor, the Input Capture pin would need to capture the value of a Timer when the echo pin went high, and then once again when it went low, and subtract the Timer values to find the total amount of time that had gone by.

The following is a sample flowchart of how a standard run with the PIC32 and ultrasonic sensor goes:

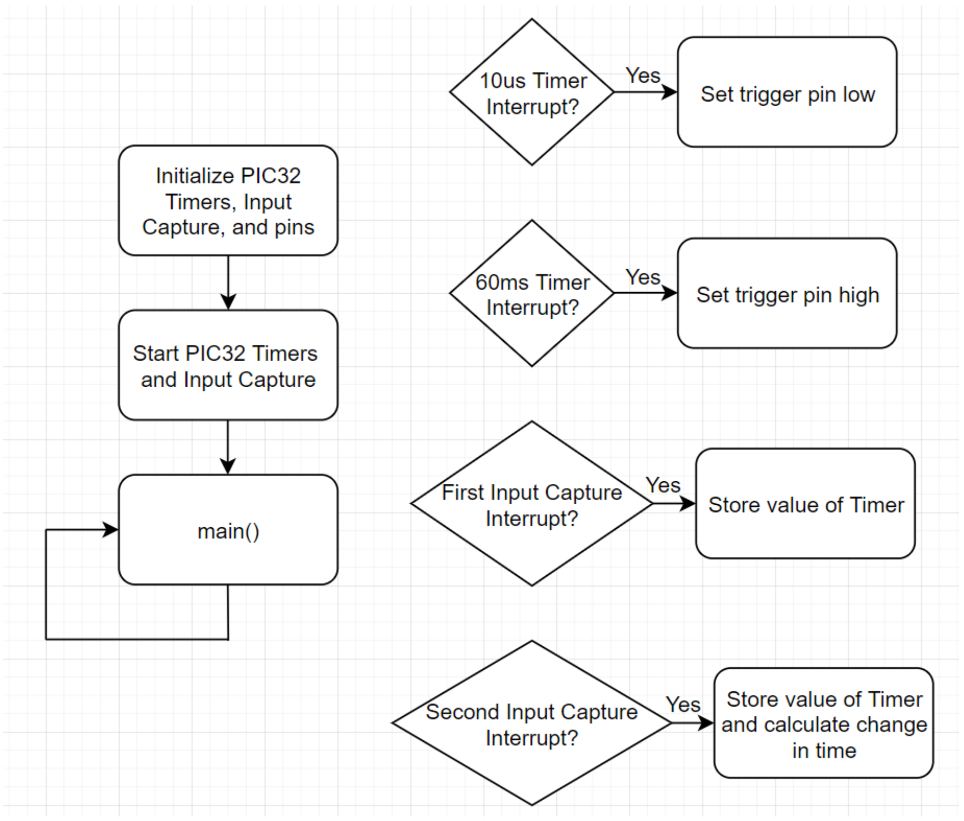

10us Timer Interrupt?
Yes
Set trigger pin low
Initialize PIC32 Timers, Input Capture, and pins
60ms Timer Interrupt?
Yes
Set trigger pin high
Start PIC32 Timers and Input Capture
First Input Capture Interrupt?
Yes
Store value of Timer
main()
Second Input Capture Interrupt?
Yes
Store value of Timer and calculate change in time

Fig. 5.2. Sample Run with the PIC32 and Ultrasonic Sensor

In the initialization phase, two Timers, Timer 2 and Timer 3 are initialized to be 60ms and 10us timers by setting Timer 2's pre-scalar and PR to 256 and 9375 respectively Timer 3's pre-scalar and PR to 16 and 25 respectively. Input Capture 1 is set up with Timer 2 as its Timer source, and so that it triggers an interrupt on every rising and falling edge of the echo pin. The pin from the PIC32 going to the ultrasonic trigger input is configured as an output pin with a starting value of 1, while the pin from the ultrasonic echo output to the PIC32 is configured as an input pin with a starting value of 0. The Timers and Input Capture are then started, and the program enters the main loop. Once the 10us Timer goes off and triggers an interrupt, the trigger pin is set to 0. If the 60ms Timer goes off and triggers an interrupt, the trigger pin is set back to 1. When the ultrasonic sensor sends out a ping and raises the echo pin, the first Input Capture interrupt goes off and triggers an interrupt, and the value of Timer 2 is stored. When the ping comes back to the ultrasonic sensor and the echo pin is lowered, the second Input Capture interrupt goes off and triggers an interrupt, and the value of Timer 2 is stored once again, and the first Timer 2 value is subtracted from the second Timer 2 value to find the change in time of the ping being sent and received. However, the values of the Timer are not equivalent to real-time, since they run off of the Peripheral Clock. To get this time in real-time, you need to multiply by the real-time you set the Timer to, which would be 60ms or 10us in this case, and divide by the PR of the Timer. This gives you the actual time of the ping, and by dividing by two to find the time taken for just the ping to be sent out from the ultrasonic sensor and reach some object, and then by multiplying by the speed of sound, the distance of the object from the sensor can be determined.

The HCSR04 ultrasonic sensor's trigger is soldered to the digital pins from the PIC32MX340F512H microcontroller. With a combined total of 4 ultrasonic trigger pins, each was soldered to their own digital output pins from the microcontroller. The microcontroller must output a logic high of at least 10us to command the ultrasonic sensor to start sampling distance.

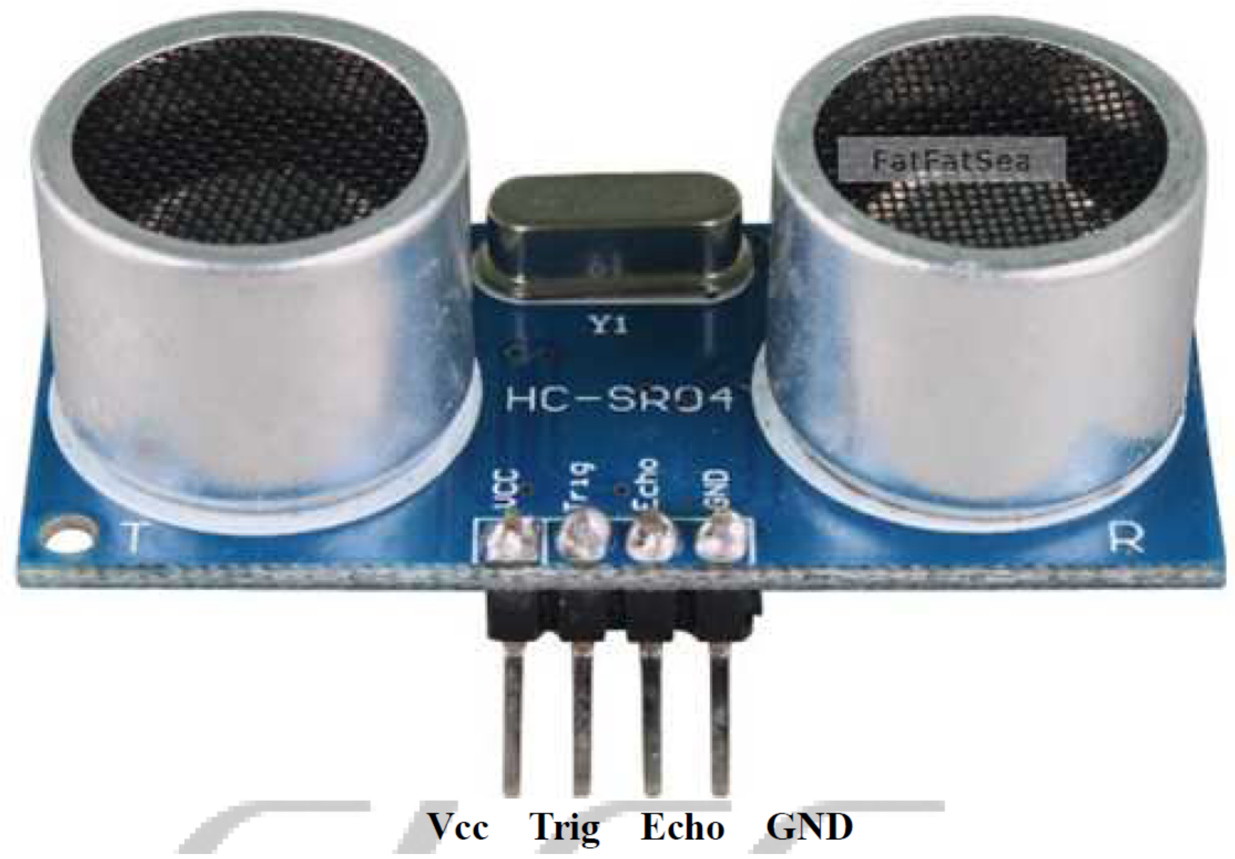


Fig. 5.3. HCSR04 Ultrasonic Sensor Breakout Board

Each echo pin is soldered to an Input Capture pin of the PIC32MX340F512H to have interrupt driven logic high time recording of each ultrasonic sensor. The logic high duration determines the distance between the object to the sensor.

The ultrasonic sensor itself outputs digital signals via the trigger and echo pins and only needs 5V to Vcc to power and ground pin. The algorithm to calculate the time taken for the ultrasonic burst is done on a separate breakout board on the ultrasonic sensor itself with ICs on the back of the breakout board [21].

To connect the pinouts from the ultrasonic sensor to the PCB, 4 wires are used to connect the Vcc, Trig, Echo, and GND pin to the pinout holes shown below in figure 5.4.

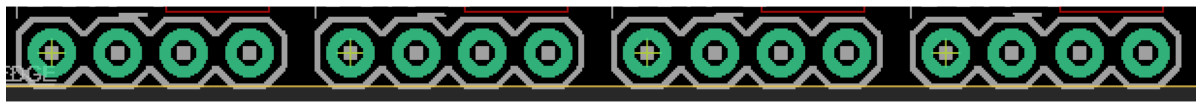


Figure 5.4. PCB pinout for 4 HCSR04 Ultrasonic Sensors

4 ultrasonic sensor modules are connected to the PCB board, with each module connected to a set of 4 through hole pinouts. A zoomed in diagram of each set of ultrasonic pinout is shown in figure 3 below.

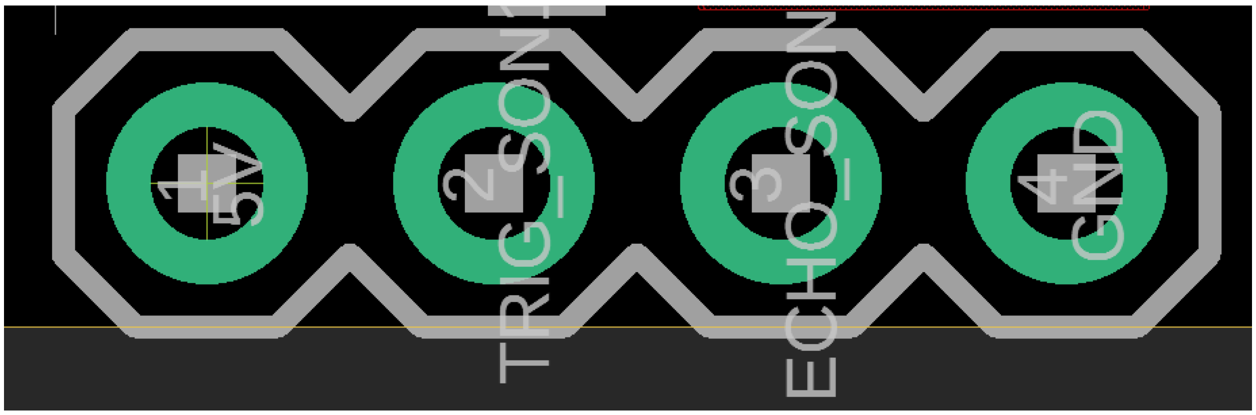


Fig. 5.5. 4 Through Hole Pinouts on PCB for HCSR04 Ultrasonic Sensor

Each set of ultrasonic sensor pinouts on the PCB is arranged the same way as in figure 5.3. With the 5V pinout on the left most side soldered to the wire connecting to HCSR04, for power to Vcc pin on the HCSR04 pinout. The TRIG_SON(n) pin on figure 5.3 is soldered to the wire connecting the TRIG_SON(n) pin of figure 5. 3, followed by the ECHO_SON(n) soldered to the wire connecting the Echo pin of HCSR04. Lastly the GND pin is soldered to the wire connecting to the GND pin of HCSR04.

5.1.2 MPL3115A2 Altimeter (Barometric and temperature sensor)

To satisfy STR 4.2.2, which was to use the altimeter to monitor altitudes above 4m, the MPL3115A2 altimeter was chosen. The altimeter would be used to measure elevation of the drone. The altimeter uses I2C, so an I2C procedure had to be written on the PIC32, which has I2C hardware on it.

I2C is one of the other essential pieces of hardware on the PIC32 which allows for serial transmission as both a master and a slave. The PIC32 I2C module has seven special function registers ranging in purpose of controlling the module, checking the state of it, setting the baud rate of the SDA line, and transmitting and receiving data. To follow the standard I2C procedure, the following transmission diagram from the PIC32 I2C datasheet was used to write the procedure[71]:

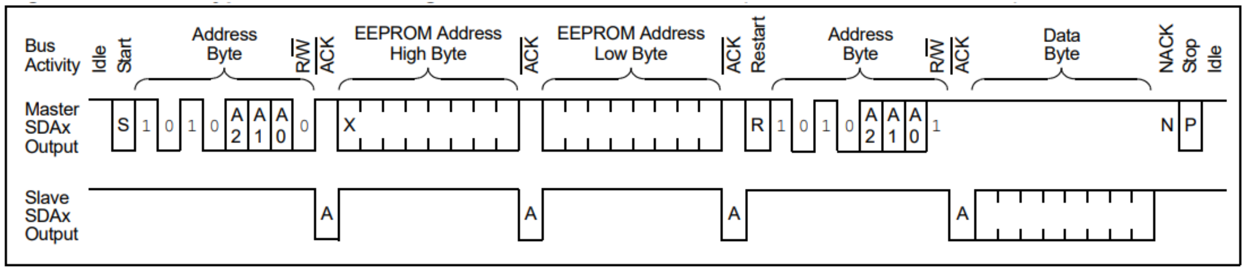


Fig. 5.6. I2C Procedure

In this example the master, the PIC32, is reading from the slave. The PIC32 first sends a start bit along the I2C SDA line to indicate start of transmission. It then sends the address of the slave as a byte to identify it, and waits until the slave acknowledges and responds, which will happen when the slave pulls the SDA line low. Next, the master sends out the address of the specific register it wishes to read or write from the slave, and once again waits for acknowledgement from the slave. Then, the PIC32 sends out a restart signal to indicate that it is about to read or write from the slave. It then sends the address of the slave once again, with the last bit (least-significant bit) signifying whether the PIC32 is reading or writing, 1 for read and 0 for write, which in this case is 1 for reading. It then waits for another acknowledgment from the sensor, and reads the data byte being sent from the slave. If the PIC32 were writing to the sensor, the last bit of the slave address would be 0, and it would write a byte of data along the SDA line. Finally, the PIC32 sends acknowledgement back to the slave to indicate success or failure of reading/writing, and sends a stop signal to indicate it is done.

To implement this in software, the most important components of the I2C module that needed to be configured were the control, status, baud rate, transmit data, and receive data registers. The control register sets the operational control of the module, most importantly being the enable bit to turn on the module. The status register is self-regulated, and can be checked upon to see the acknowledgement status and transmit and receive status. The baud rate register sets the baud rate of the SCL line, which is calculated based on the Peripheral Clock and is found by using this equation from the datasheet:

$$I2CxBRG = \left[\left(\frac{1}{(2 \cdot F_{SCK})} - T_{PGD}\right) \cdot PBCLK\right] - 2$$

Equation 5. 1. I2C Baud Rate Generator

For a TPGD of 104ns, which is the delay given by the datasheet, a PBCLK of 40MHz, the Peripheral Clock of the PIC32, and the desired FSCK of 100kHz for the SCL line, the BRG, or baud rate generator value to be written to the baud rate register, should be 0x02C. Finally, the transmit data and receive data registers are used to send and receive a byte of data.

The altimeter also had a dedicated interrupt pin that would let the PIC32 know when new data was ready to read from, and could be utilized with the PIC32's Change Notification functionality. The PIC32's Change Notification allows the PIC32 to trigger an interrupt when the state of a pin changes. The main registers are the control register, which configures the change notification module to be on or off, as well as the enable register, which contains which pins should be tracked.

To read from the altimeter, the following flowchart from the altimeter datasheet was referenced[72]:

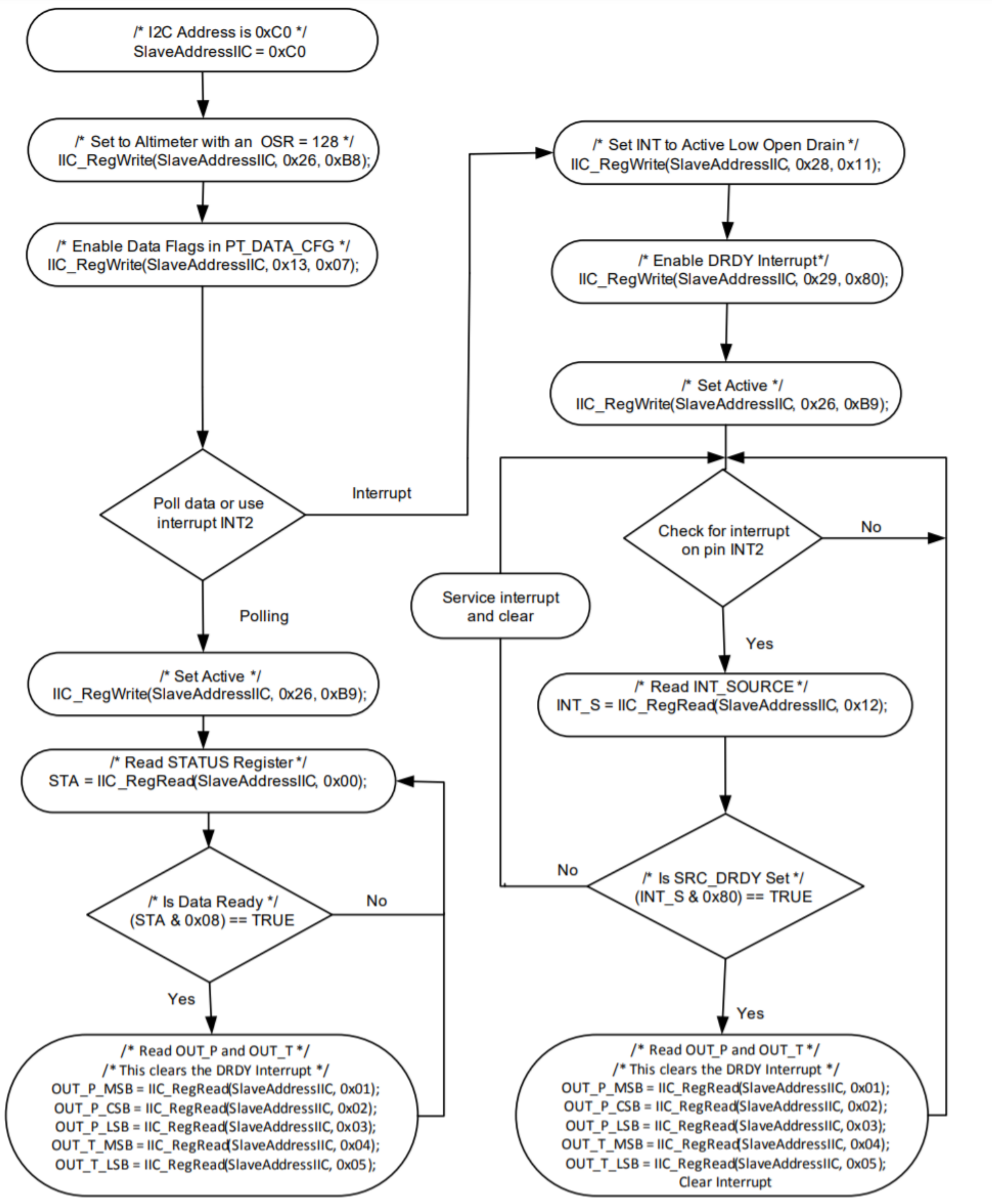

/* I2C Address is 0xC0 */
SlaveAddressIIC = 0xC0
/* Set to Altimeter with an OSR = 128 */
IIC_RegWrite(SlaveAddressIIC, 0x26, 0xB8);
/* Enable Data Flags in PT_DATA_CFG */
IIC_RegWrite(SlaveAddressIIC, 0x13, 0x07);
Poll data or use interrupt INT2
Interrupt
Polling
/* Set Active */
IIC_RegWrite(SlaveAddressIIC, 0x26, 0xB9);
/* Read STATUS Register */
STA = IIC_RegRead(SlaveAddressIIC, 0x00);
/* Is Data Ready */
(STA & 0x08) == TRUE
No
Yes
/* Read OUT_P and OUT_T */
/* This clears the DRDY Interrupt */
OUT_P_MSB = IIC_RegRead(SlaveAddressIIC, 0x01);
OUT_P_CSB = IIC_RegRead(SlaveAddressIIC, 0x02);
OUT_P_LSB = IIC_RegRead(SlaveAddressIIC, 0x03);
OUT_T_MSB = IIC_RegRead(SlaveAddressIIC, 0x04);
OUT_T_LSB = IIC_RegRead(SlaveAddressIIC, 0x05);
/* Set INT to Active Low Open Drain */
IIC_RegWrite(SlaveAddressIIC, 0x28, 0x11);
/* Enable DRDY Interrupt*/
IIC_RegWrite(SlaveAddressIIC, 0x29, 0x80);
/* Set Active */
IIC_RegWrite(SlaveAddressIIC, 0x26, 0xB9);
Check for interrupt on pin INT2
No
Yes
Service interrupt and clear
/* Read INT_SOURCE */
INT_S = IIC_RegRead(SlaveAddressIIC, 0x12);
/* Is SRC_DRDY Set */
(INT_S & 0x80) == TRUE
No
Yes
/* Read OUT_P and OUT_T */
/* This clears the DRDY Interrupt */
OUT_P_MSB = IIC_RegRead(SlaveAddressIIC, 0x01);
OUT_P_CSB = IIC_RegRead(SlaveAddressIIC, 0x02);
OUT_P_LSB = IIC_RegRead(SlaveAddressIIC, 0x03);
OUT_T_MSB = IIC_RegRead(SlaveAddressIIC, 0x04);
OUT_T_LSB = IIC_RegRead(SlaveAddressIIC, 0x05);
Clear Interrupt

Fig. 5.7. Altimeter Flowchart

To start, the altimeter is configured to 128x sample rate and to log new pressure data, and convert it to altitude by writing to its Control Register 1 and PT Data Configuration Register. Then, for the interrupt procedure, Control Register 3 and Control Register 4 are configured to trigger an interrupt on new data, followed by writing to Control Register 1 to start the altimeter. When the altimeter reads new data and triggers an interrupt, the Pressure Data Out registers are read from, with Pressure Data Out being divided into three registers. To convert the Pressure Data Out values from bits to a signed fractional number, the most significant byte must be left shifted by 24 bits into a 32-bit variable, logically OR'd with the central significant byte left shifted by 16 bits, and logically OR'd with the least significant byte left shifted by 8 bits. This value multiplied by 65536 gives the altitude in meters.

The following is a sample flowchart of how a standard run with the PIC32 and altimeter goes:

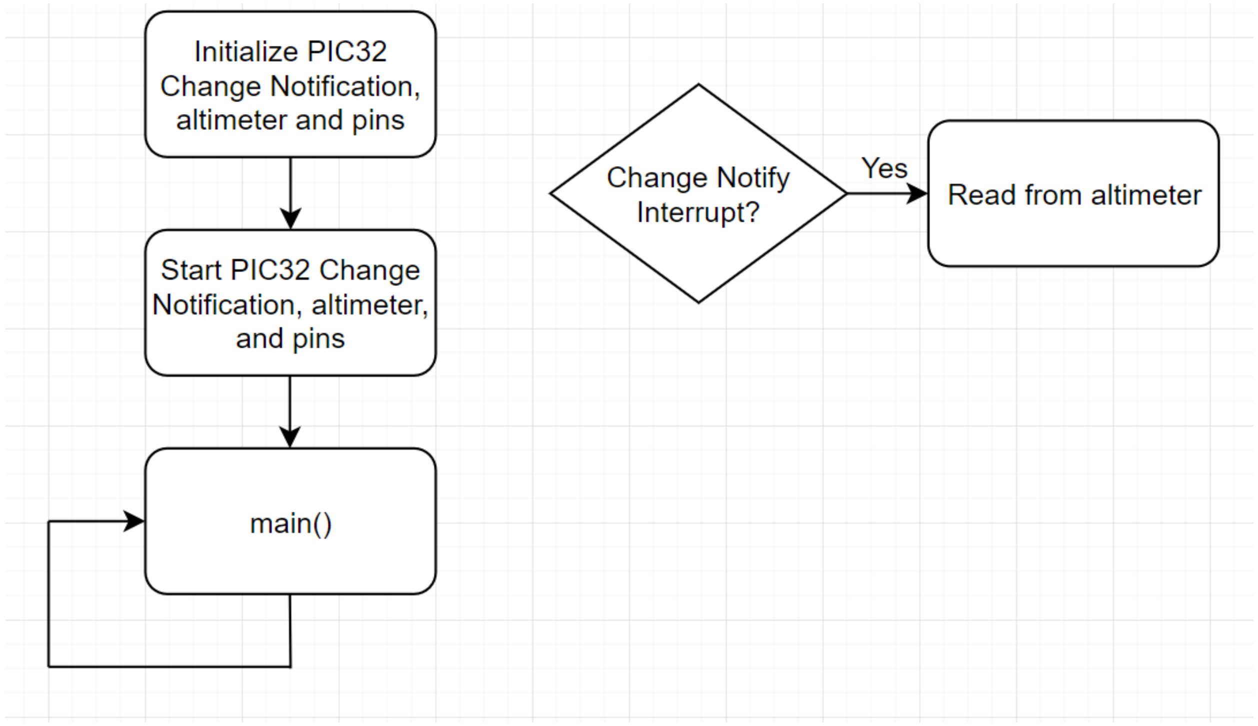

Initialize PIC32 Change Notification, altimeter and pins
Start PIC32 Change Notification, altimeter, and pins
main()
Change Notify Interrupt?
Yes
Read from altimeter

Fig. 5.8. Sample Run with the PIC32 and Altimeter

In the initialization phase, the Change Notification module is configured with the desired pins to keep track of, and the pin itself is configured as an input, and is given a starting value of 0. The altimeter is also initialized to max sample rate and to log pressure data and trigger an interrupt when new data is ready. The Change Notification module is then turned on, the altimeter is enabled, and the program enters the main loop. When the altimeter has new data ready, it will trigger its interrupt pin, and enable the Change Notification interrupt for the pin connected to the PIC32. In this interrupt, the proper registers from the altimeter are read from (as described in the flowchart earlier), and the data from the sensor is processed.

The altimeter uses I2C pinout (SDA and SCL) to communicate with the PIC32 microcontroller to provide pressure data to enable high altitude tracking where the ultrasonic sensors cannot track. As shown in figure 5.19 below.

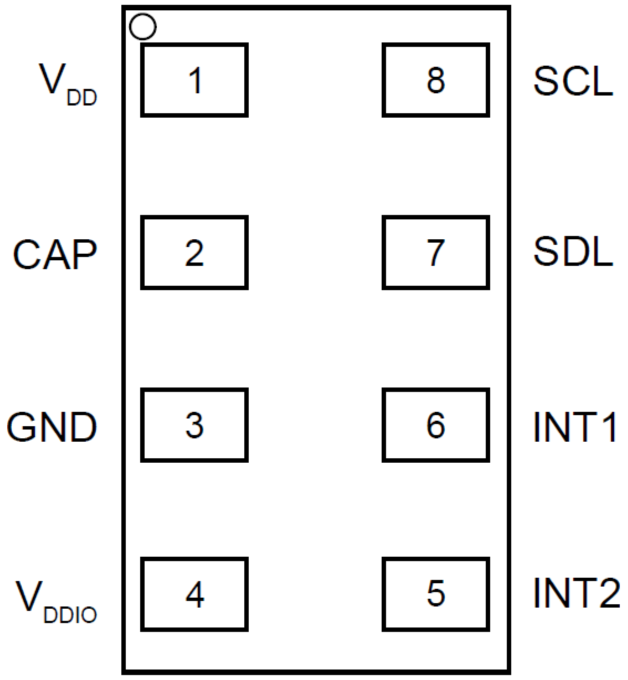


Fig. 5.9. MPL3115A2 Altimeter LGA Pinout

VDD/pin 1 and VDDIO/pin 4 in figure 5.9 are connected to the 3.3V plane of the PCB. VDD/pin 1 is the operating supply voltage and VDDIO/pin 4 is the supply voltage for SDL, SCL, INT1, and INT2 pins. The CAP/pin 2 is connected to an external 0.1uF capacitor as recommended by the datasheet[22]. The SCL/pin 8 or clock line of I2C is connected to the PIC32 microcontroller's SCL1/pin37, the wiring is done through the PCB trace which will be explained in section 3 of this chapter. The SDL/pin7 of the altimeter is connected to the SDA1/pin of the PIC32. The INT1/pin6 is connected to RB0/pin16 of PIC32, its purpose is to send a logic high for interrupt driven data sampling of the atmospheric pressure. INT2/pin5 is connected to RB1/pin15 of the PIC32, INT2 is programmable via I2C bus to MPL3115A2 to trigger interrupt when a certain pressure has been detected/reached. It is recommended by the manufacturer's datasheet to have a 100nF and 10uF capacitor placed in parallel to bypass the internal regulator as shown in figure 5.10 below.

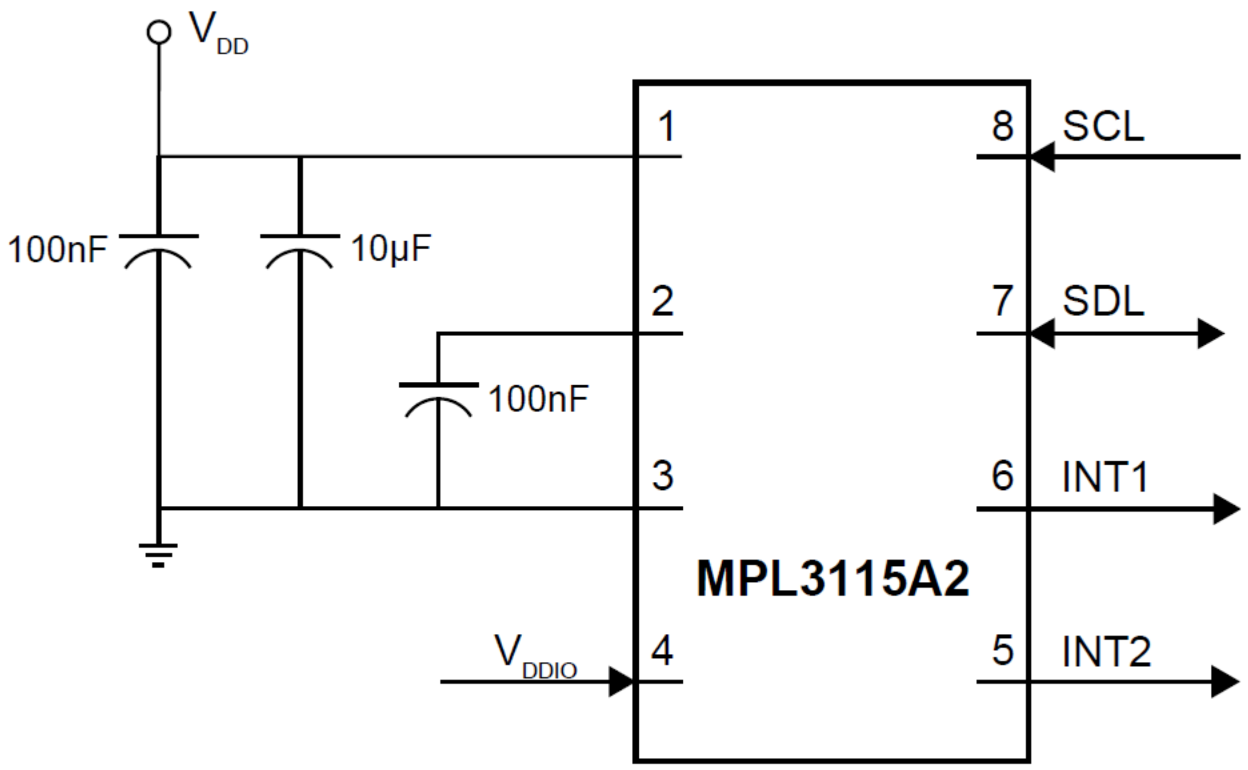


Fig. 5.10. MPL3115A2 Altimeter Typical Application Diagram

5.1.3 ICM-20948 9DoF IMU

To satisfy STR 4.3.1, which was to use the IMU sensor to detect the orientation of the drone, the ICM-20948 9DoF IMU was chosen. To read from the IMU sensor, a similar procedure to the altimeter's was followed. The IMU sensor uses I2C as a form of communication, and had a dedicated interrupt pin that would signal when new data was ready, which meant the PIC32's Change Notification module would once again be used. The main registers used in the IMU sensor were the LP_CONFIG, INT_PIN_CFG, INT_ENABLE, ACCEL_OUT, GYRO_OUT, and H registers. The LP_CONFIG register determined the mode of operation, while the INT_PIN_CFG and INT_ENABLE registers configured and set the interrupt trigger. The ACCEL_OUT, GYRO_OUT, and H registers were each three pairs of registers that contained a 16-bit value over a single pair representing the acceleration, rotational speed, or magnetic field in one of x, y, or z directions[73].

The following is a sample flowchart of how a standard run with the PIC32 and IMU goes:

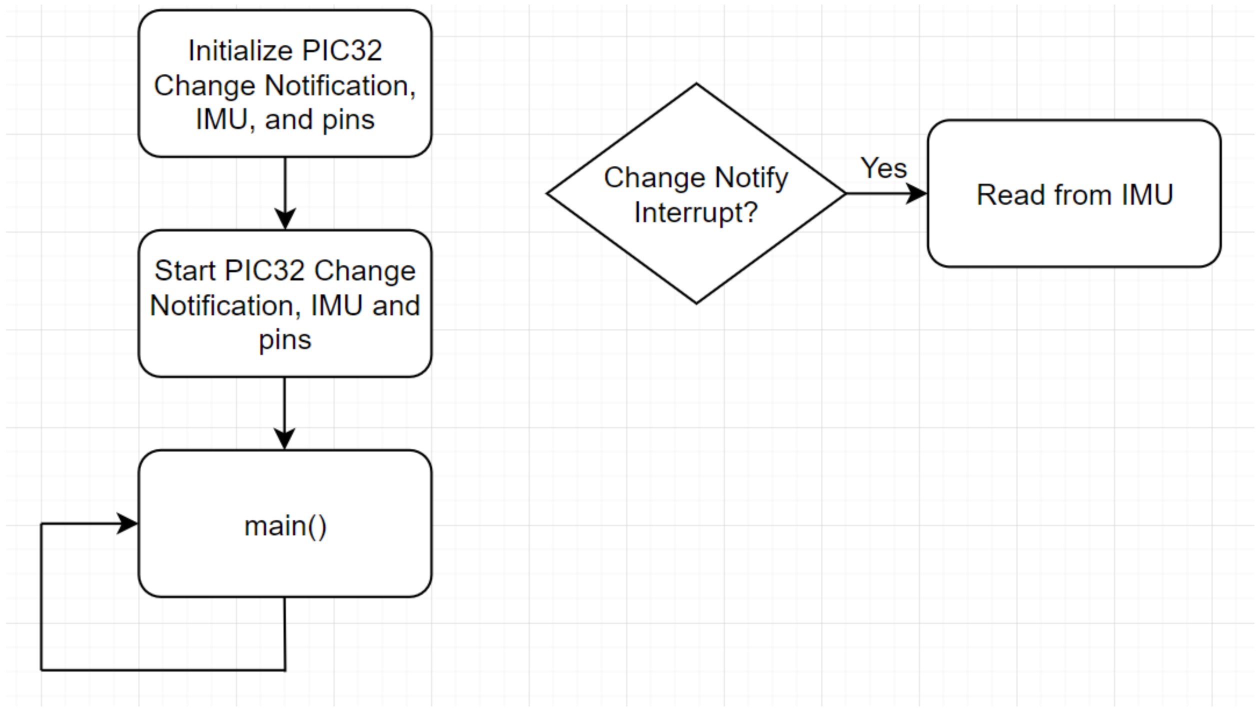

Initialize PIC32 Change Notification, IMU, and pins
Start PIC32 Change Notification, IMU and pins
main()
Change Notify Interrupt?
Yes
Read from IMU

Fig. 5.11. Sample Run with the PIC32 and IMU

In the initialization phase, the Change Notification module is configured with the desired pins to keep track of, and the pin itself is configured as an input, and is given a starting value of 0. The IMU is also initialized to max sample rate and to log acceleration, rotational speed, and magnetic field and trigger an interrupt when new data is ready. The Change Notification module is then turned on, and the program enters the main loop. When the IMU has new data ready, it will trigger its interrupt pin, and enable the Change Notification interrupt for the pin connected to the PIC32. In this interrupt, the proper registers from the IMU are read from (as described in the earlier), and the data from the sensor is processed.

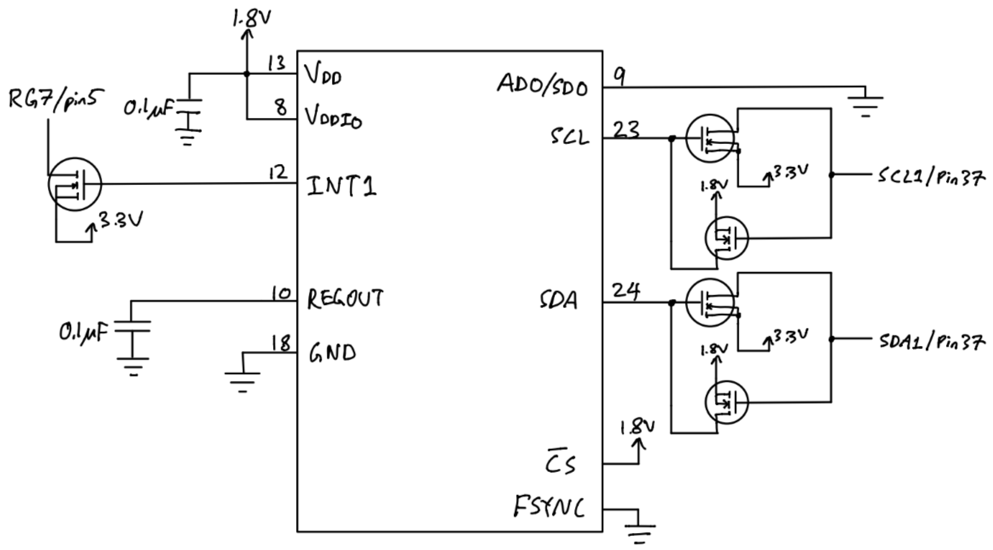


Fig. 5.12. ICM-20948 IMU Pinout to PIC32 Schematic

The ICM-20948 IMU's VDDIO only allows a maximum voltage of 1.95V, this also means the logic high of I2C bus is limited to 1.8V. Two logic level converters are implemented in the PCB to convert 1.8V output from SCL and SDA to 3.3V of the IMU, as shown in figure 5.12 on pin 23 and 24, into 3.3V signal into SCL1 and SDA1 pins of the PIC32 microcontroller. The PIC32 microcontroller outputs 3.3V I2C signals to the IMU, so another two logic level converters are implemented to convert 3.3V signal to 1.8V into the IMU.

An interrupt driven output pin called INT1/pin12 of the IMU is connected to a logic level converter to output 3.3V interrupt to RG6/pin 5 of the microcontroller, which enables interrupt driven accelerometer data.[23]

5.1.4 MT3339 GPS Module

The GPS module satisfies STR 4.1.1 for location tracking via UART to PIC32 microcontroller, accurate within 5 meters and a sampling rate of at least 3Hz.

The GPS module communicates through the use of a UART, and could be interfaced with using the PIC32's UART. In addition, the GPS module has a fix pin, which outputs a high signal when the GPS module locks onto a satellite, and could be used with the PIC32's Change Notification functionality to detect when to start taking measurements from the GPS module[74].

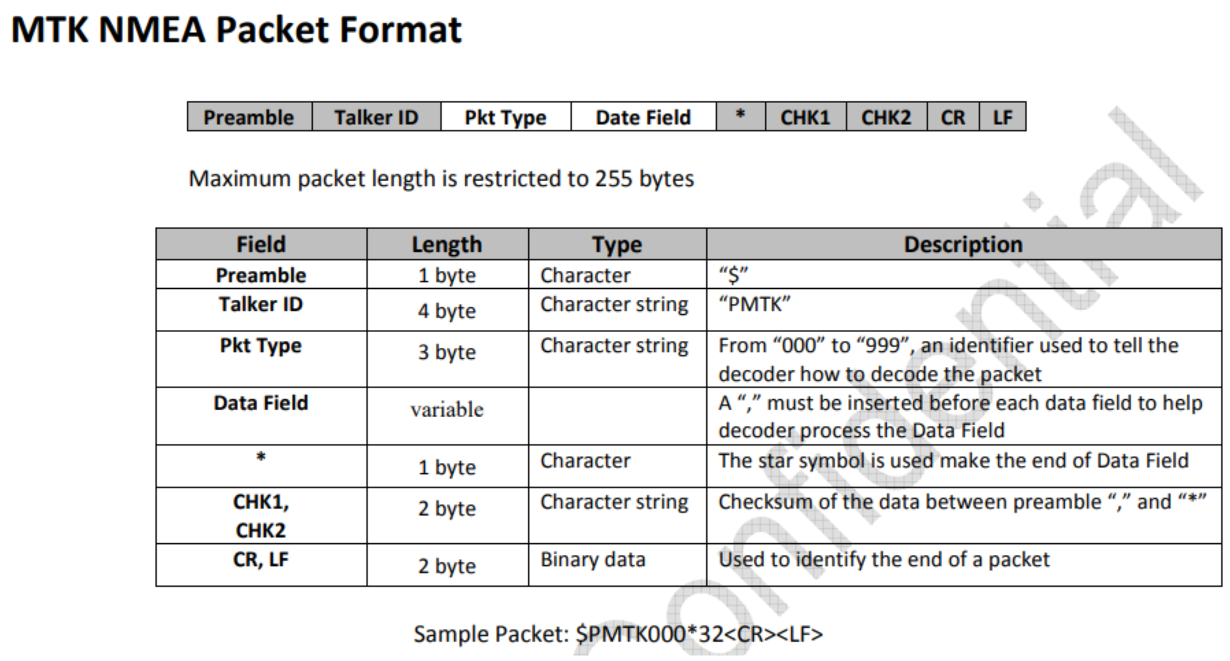
The PIC32's UART is another essential piece of hardware on the PIC32 that allows for serial communication. The main registers within the UART module are the mode, status and control, transmit and receive, and baud rate registers. The mode register sets the mode of the UART module, including turning it off or on and the number of data, parity, and stop bits. The status and control register sets the interrupts, as well as controls and keeps track of the state of the transmission and any possible errors. The transmit and receive registers are for data to be transmitted and received, while the baud rate register sets the baud rate of the UART and is found by using the following equation from the PIC32 UART datasheet[75]:

$$UxBRG = \frac{F_{PB}}{4 \cdot \ Baud\ Rate} - 1$$

Equation 5.2. UART Baud Rate Generator

For example, for a desired baud rate of 115200, with FPB being the Peripheral Clock at 40MHz, the BRG value to be loaded into the baud rate register should be 21.

The GPS module uses a special protocol called MTK NMEA Packet Protocol, and is explained in the datasheet for the GPS module:



MTK NMEA Packet Format

Preamble	Talker ID	Pkt Type	Date Field	*	CHK1	CHK2	CR	LF

Maximum packet length is restricted to 255 bytes

Field	Length	Type	Description
Preamble	1 byte	Character	"$"
Talker ID	4 byte	Character string	"PMTK"
Pkt Type	3 byte	Character string	From "000" to "999", an identifier used to tell the decoder how to decode the packet
Data Field	variable		A "," must be inserted before each data field to help decoder process the Data Field
*****	1 byte	Character	The star symbol is used make the end of Data Field
CHK1, CHK2	2 byte	Character string	Checksum of the data between preamble "," and "*"
CR, LF	2 byte	Binary data	Used to identify the end of a packet

Sample Packet: $PMTK000*32<CR><LF>

Fig. 5.13. GPS Module NMEA Packet Protocol

Essentially, as the datasheet shows, the packet length consists of a 255-byte length packet, where the packet type and data field are used to access certain parts of the GPS module and command the GPS. The main packet types to consider are 001 PMTK_ACK, 010 PMTK_SYS_MSG, 251 PMTK_SET_NMEA_BAUDRATE, and 622 PMTK_Q_LOCUS_DATA. 001 PMTK_ACK in the GPS module indicates that the module acknowledged a command, while 010 PMTK_SYS_MSG outputs a system message whether an operation was successful or not. 251

PMTK_SET_NMEA_BAUDRATE sets the baud rate of the GPS module, and 622 PMTK_Q_LOCUS_DATA dumps all stored data.

The following is a sample flowchart of how a standard run with the PIC32 and GPS goes:

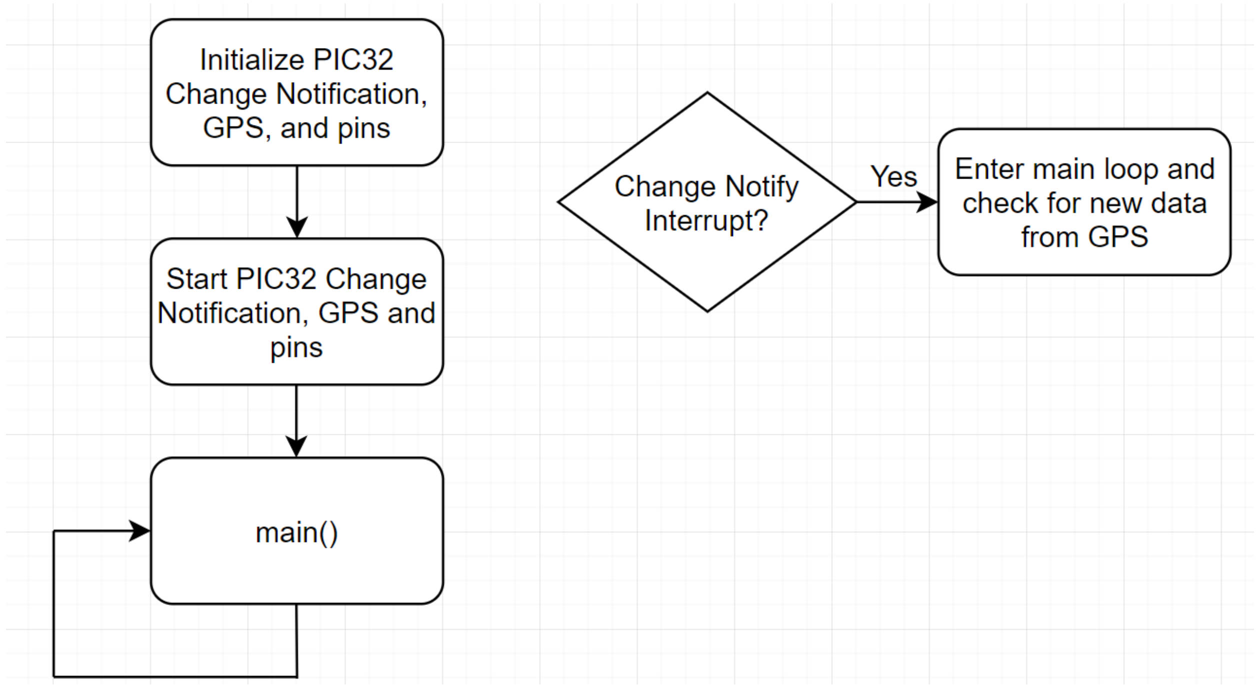

Initialize PIC32 Change Notification, GPS, and pins
Start PIC32 Change Notification, GPS and pins
main()
Change Notify Interrupt?
Yes
Enter main loop and check for new data from GPS

Fig. 5.14. Sample Run with the PIC32 and GPS

In the initialization phase, the Change Notification module is configured with the desired pins to keep track of, and the pin itself is configured as an input, and is given a starting value of 0. The GPS is also initialized to its max sample rate and a baud rate of 115200. The Change Notification module and GPS are then turned on, and the program enters the main loop. The main program waits until the GPS has a fix by waiting until a flag is set by the Change Notification interrupt, and then in the main loop, if the GPS has new data, it will be read from by using the packet protocol described earlier.

The corresponding pinouts for the MTK3339 GPS Module is shown in figure 5.15 below.

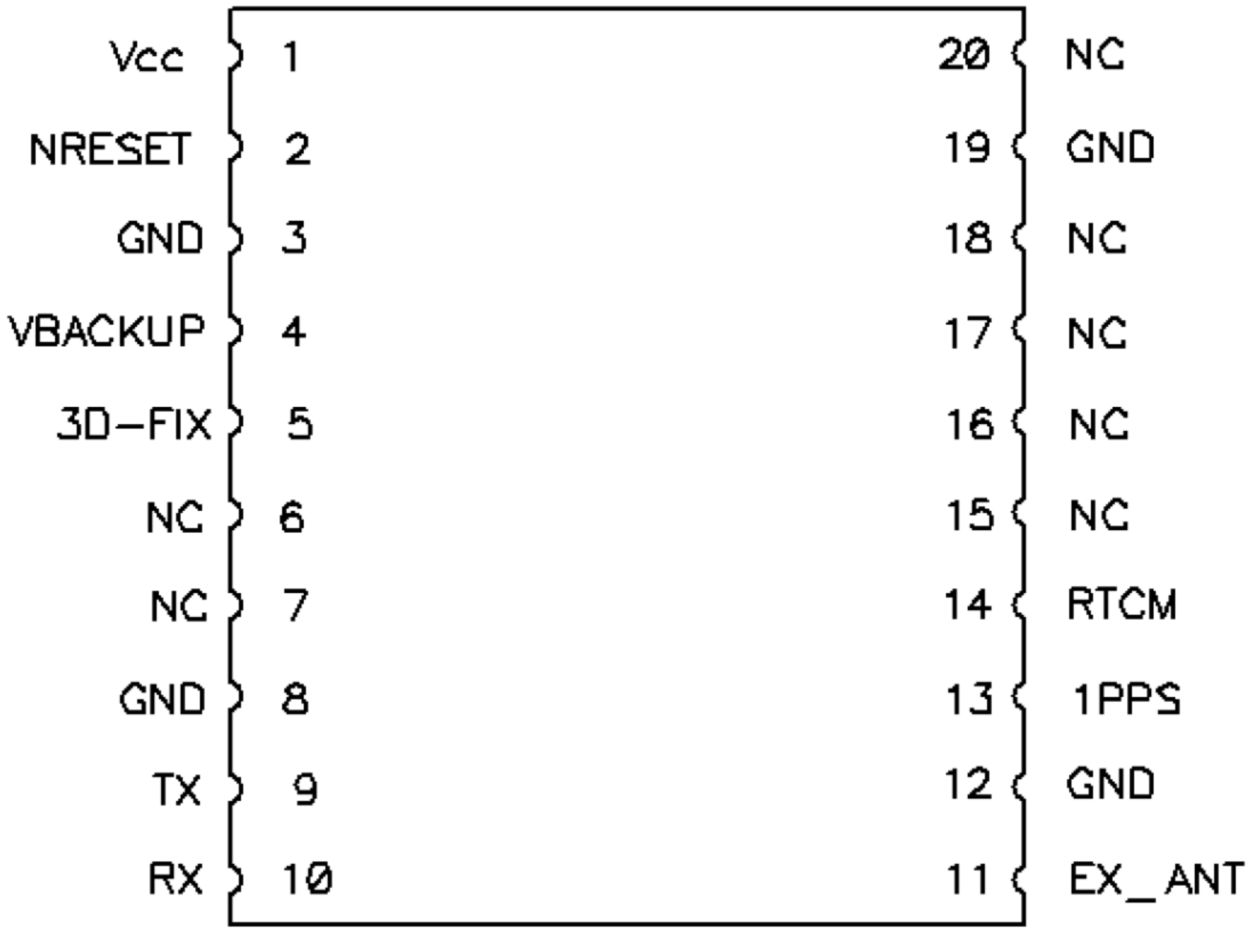


Fig. 5.15. Top Down Pinout Diagram of MTK3339 Module

The MT3339 GPS Module takes 3.3V from the 3.3V plane from the PCB into VCC/pin1. The NRESET/pin is not connected since the module is not needed to be reset during flight, not the NRESET pin is active low so do not ground the pin when in use. The VBACKUP pin is connected the positive terminal of a CR1220 coin battery holder on the PCB board. The 3V power to VBACKUP keeps the internal real time clock of the GPS module running when there is no power to VCC/pin1. The 3D-FIX/pin5 is connected to RF6/pin 35 of the PIC32 microcontroller. A 1Hz pulse indicates the GPS module is seeking a lock on a GPS and a logic low indicates at least 1 satellite lock is acquired, which is needed on startup of the drone to show if the GPS is capable of autonomous flight. Pins 6,7,15,16,17,18 are not connected to any traces on the PCB but are soldered in place. The ground pins 8, 12, and 19 are connected to the ground plane of the PCB using a via hole. TX/pin 9 is connected to U1RX/pin 34 of the PIC32 microcontroller and RX/pin 10 is connected to U1TX/pin 33 of the microcontroller for sending coordinate packets and other debug packets via UART protocol. [24]

5.1.5 MPRLS0001PG00001C Balloon Pressure Sensor

The MPRL balloon pressure sensor satisfies STR 4.3.3. By having a tube attached from the nozzle of the balloon to the gauge reference hole, the sensor shall detect a decrease in pressure and send an error message via I2C to the PIC32 microcontroller.

The MPR balloon pressure sensor is powered by 3.3V from the PCB 3.3V power plane to VDD/pin 12 and also has a 0.1uF bypass capacitor soldered on the PCB for voltage noise suppression[25].

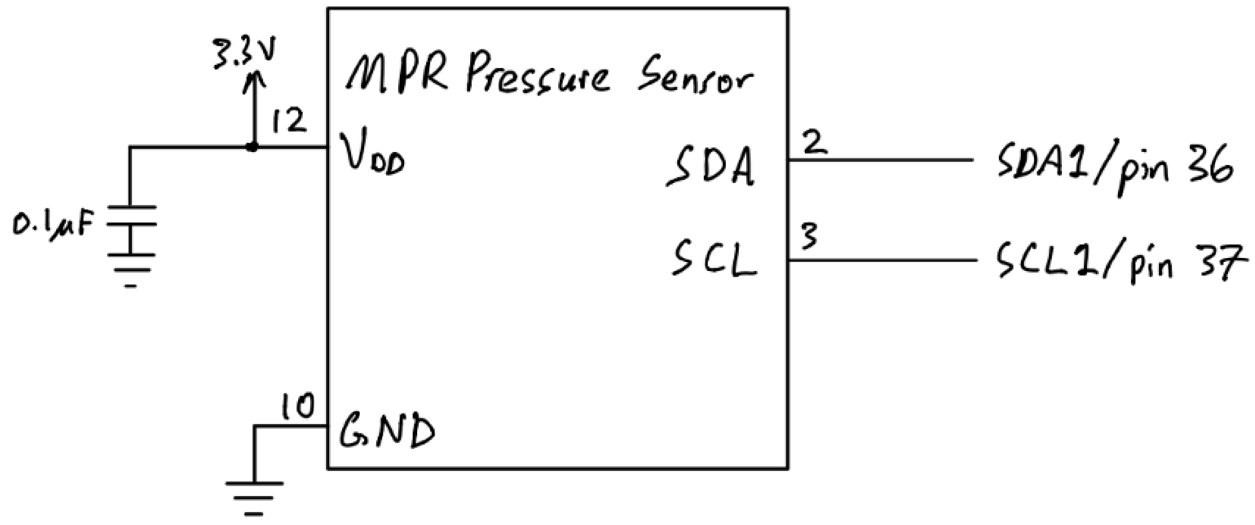


Fig. 5.16. MPR Pressure Sensor Pinout Diagram

GND/pin 10 is connected to the ground plane of the PCB through a via pin. SDA/pin 2 of the pressure sensor is connected to SDA1/pin 36 of PIC32 microcontroller for data line of I2C and SCL/pin 3 is connected to SCL1/pin 37 of the microcontroller for the clock of I2C. Pressure data is polled via pin 2 and pin 3 of the pressure sensor instead of interrupt driven since there are no interrupt pins available on the sensor IC itself.

5.1.6 Microcontroller Selection

Once the proper sensors and remote controller receiver, servos, and motors were chosen, the microcontroller could be chosen. These are the hardware requirements that were needed for the microcontroller:

- Raspberry Pi - 1 unique SPI
- Ultrasonic sensor - 1 unique Input Capture module
- IMU sensor - 1 shared I2C module and 1 unique Change Notification module
- Altimeter - 1 shared I2C module and 1 unique Change Notification module
- GPS - 1 unique UART module and 1 unique Change Notification module

- Remote controller receiver - 4 unique Input Capture modules
- Servos - 2 Output Compares
- ESC/Motors - 2 Output Compares

In total, the microcontroller needed to have 1 SPI, 1 I2C, 3 Change Notification, 1 UART, 5 Input Capture, and 4 Output Compare modules. For this reason, the PIC32MX340F512H was chosen as the microcontroller for having the following hardware specifications:

- 2 SPI modules
- 2 I2C modules
- 2 UART modules
- 5 Input Capture modules
- 5 Output Compare modules
- 17 Change Notification modules

In addition, the PIC32MX340F512H was compatible with the MPLAB X IDE, which allowed for easy access to low level hardware.

5.1.7 Microprocessor Selection

For the microprocessor, there were only a few requirements that were needed of it:

- PIC32MX340F512H - 1 unique SPI module
- Data Telemetry Transmitter - 1 unique UART module

In total the hardware requirements the microprocessor needed were 1 SPI and 1 UART module, so a Raspberry Pi Compute Module 3+ was chosen, as it had the following hardware:

- 2 SPI modules
- 2 UART modules

5.1.8 Sensor and Microcontroller Verification

In order to verify the ultrasonic, altimeter, IMU, and GPS sensors performed to their requirements, all sensors were logged at the same time over a period of time, and had their precision determined by using the actual known values the sensors were measuring against, and comparing it to the measured values. The setup with all the sensors looked like the following:

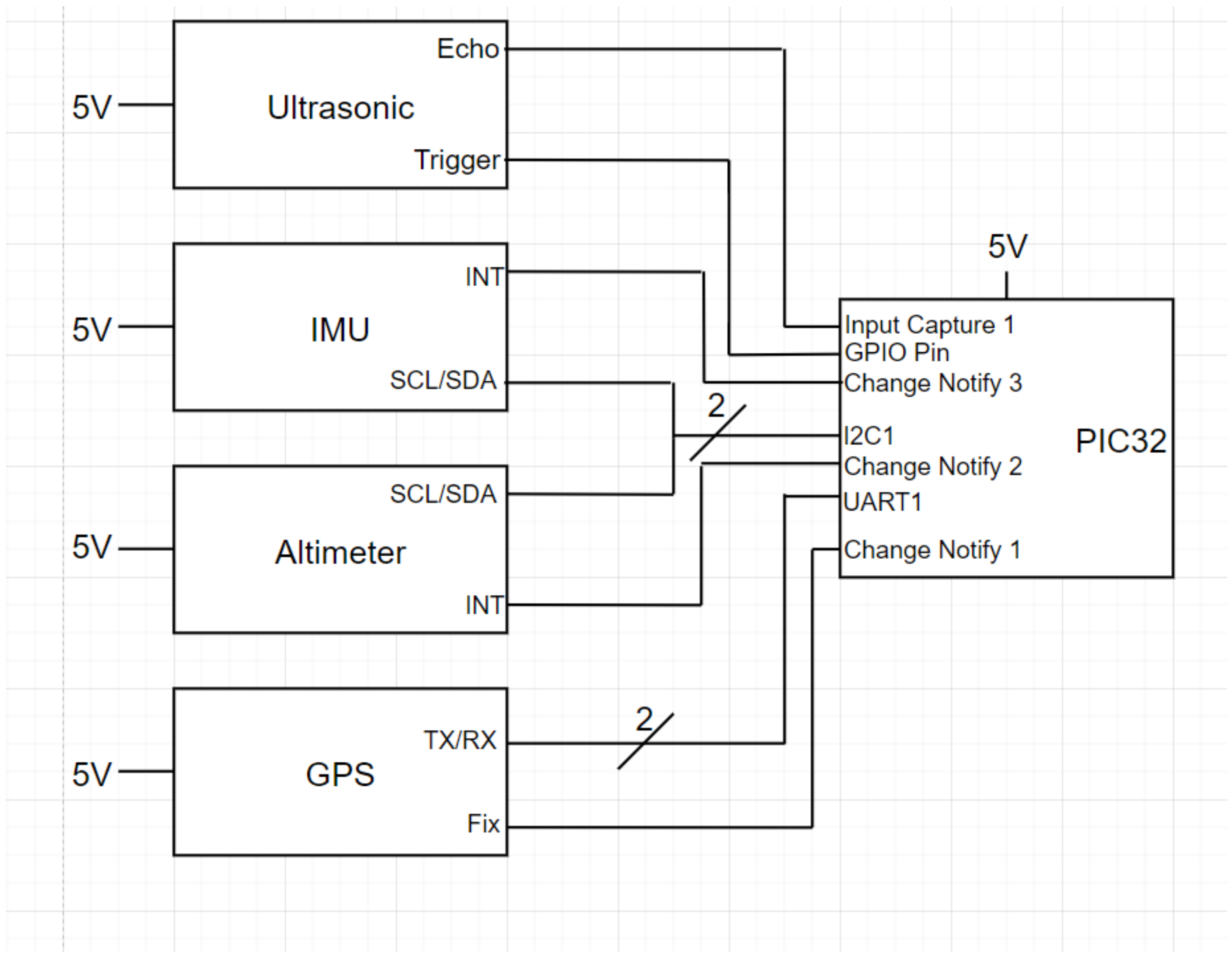


Fig. 5.17. Wiring Diagram of PIC32 with Ultrasonic, IMU, Altimeter, and GPS

Once the sensors were set up, measurements were taken from each of the sensors and compared with their expected values:

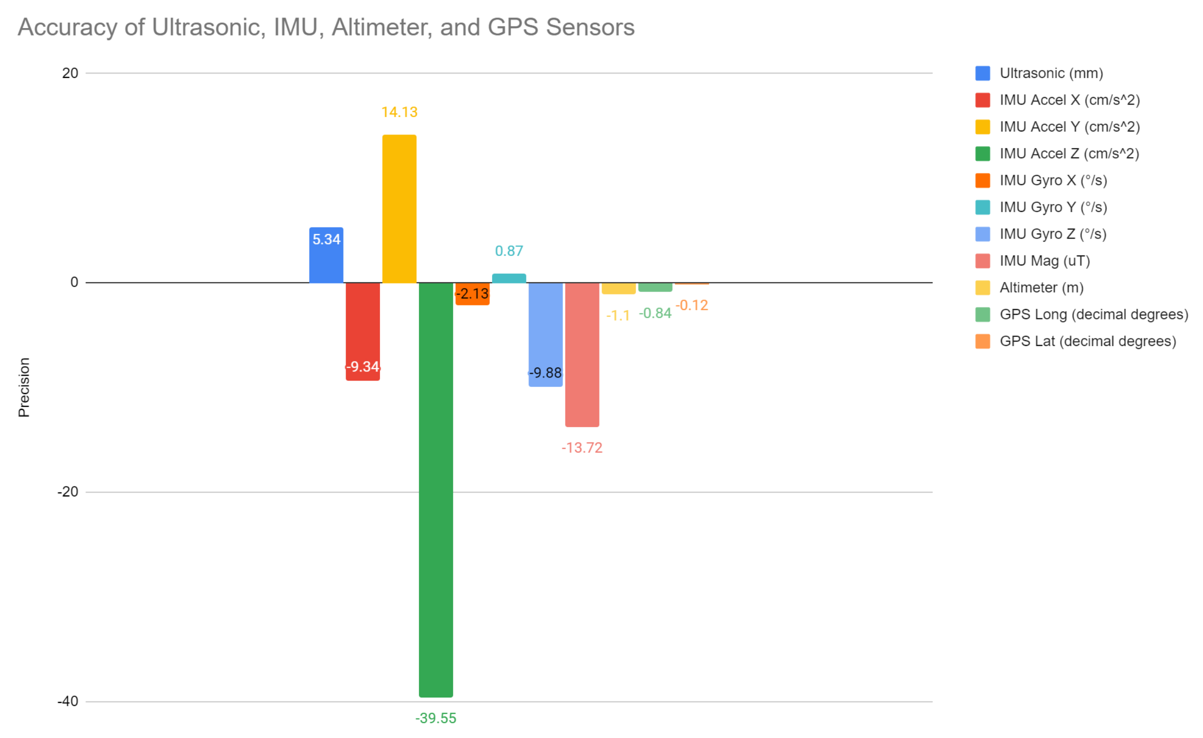


Fig. 5.18. Graph of Accuracy of Ultrasonic, IMU, Altimeter, and GPS

As the graph shows, the ultrasonic was within 5mm accuracy, the IMU acceleration along X, Y, Z axes was around 9, 39, and 2 cm/s^2 precision respectively, the IMU rotational speed along X, Y, Z axes was around 2, 1, and 10 degrees/s precision respectively, the IMU magnetic field magnitude was around 14 uT accuracy, the altimeter was around 1m accuracy, and the GPS longitude and latitude values were around 1 and 0 decimal degrees respectively. With the requirements for the ultrasonic to be within 15cm accuracy, IMU to detect orientation of the drone, the altimeter to be within 1m accuracy, and the GPS to be within 5m accuracy, every sensor but the GPS met system requirements, as it was almost 111 km off. This is believed to be caused by error in coding, and further deliberation with the GPS's datasheet must be done. In regards to the PIC32 microcontroller, it had the required hardware needed to run all sensors, and is also verified to work correctly with the remote controller receiver, servos, and motors in Section 4.2. To satisfy STR 3.1.2, where the system shall be able to handle user control inputs and error handling.

5.2 State Estimation

The state of the drone can never be fully known or directly measured, so functions need to be implemented to estimate all states that are necessary to the drone control system. The states needed for the Closed-Loop RC system will be height, pitch and roll, as well as their derivatives, as described in section 6.7.2

5.2.1 Complementary Filter for Pitch and Roll

Pitch and roll estimation is vital to ensuring the drone maintains stability during flight. The IMU has both accelerometers and gyroscopes and can be used to estimate the drone's pitch and roll. The gyroscopes are very accurate in the short term, but error and biases add up over time so just integrating the gyroscope values will be an inaccurate estimation of angles. The accelerometers are very accurate in the long term since acceleration over time is 9.8m/s/s downwards, but in the short term, the drone accelerates constantly and is inaccurate. Using a complementary filter, a high gain can be applied to gyroscopic data to take advantage of its high frequency accuracy, and the accelerometers can be given a low gain to take advantage of the low frequency accuracy of the sensor and correct for low frequency gyroscopic error. The complementary filter design in simulink is shown in Fig 5.19.

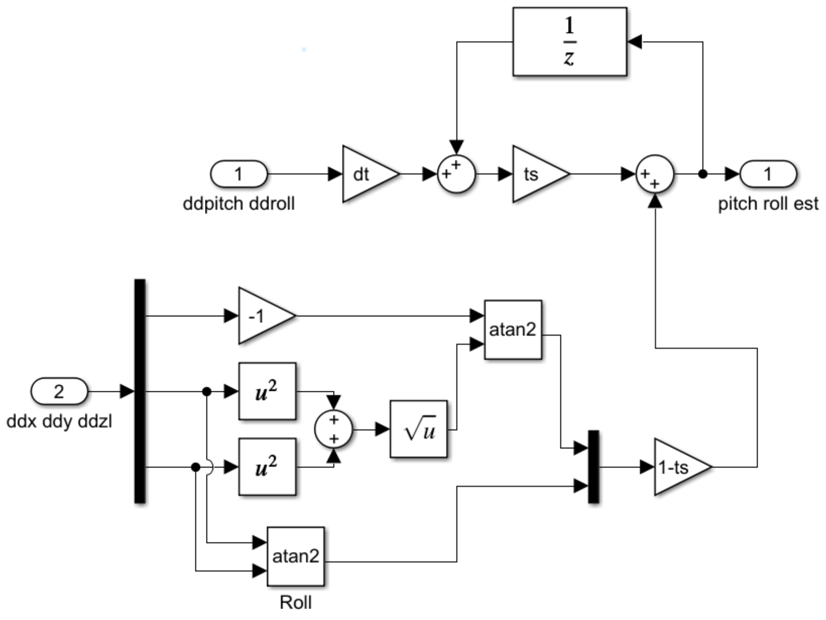


Fig. 5.19. Complementary Filter

The system takes in the accelerometer data in the bottom path and gyrospic data in the top path to be interpreted. dt is the time step of the system, and ts in the ratio of the gryropic data, and ts would be adjusted to get the most accurate pitch and roll state estimations. The system was meant to be tested and tuned in VREP, but since simulation was not complete, the design was not tested.

5.2.2 Terrain Tracking

Terrain tracking is one of the most important features of the drone since the function would allow the drone to be close to the ground without crashing and improve sensor accuracy.

The data of the ultrasonic sensor can be used to calculate the height fairly easily by just translating distance into height, but this does not cover the full area underneath the drone as seen in Fig. 5.20.

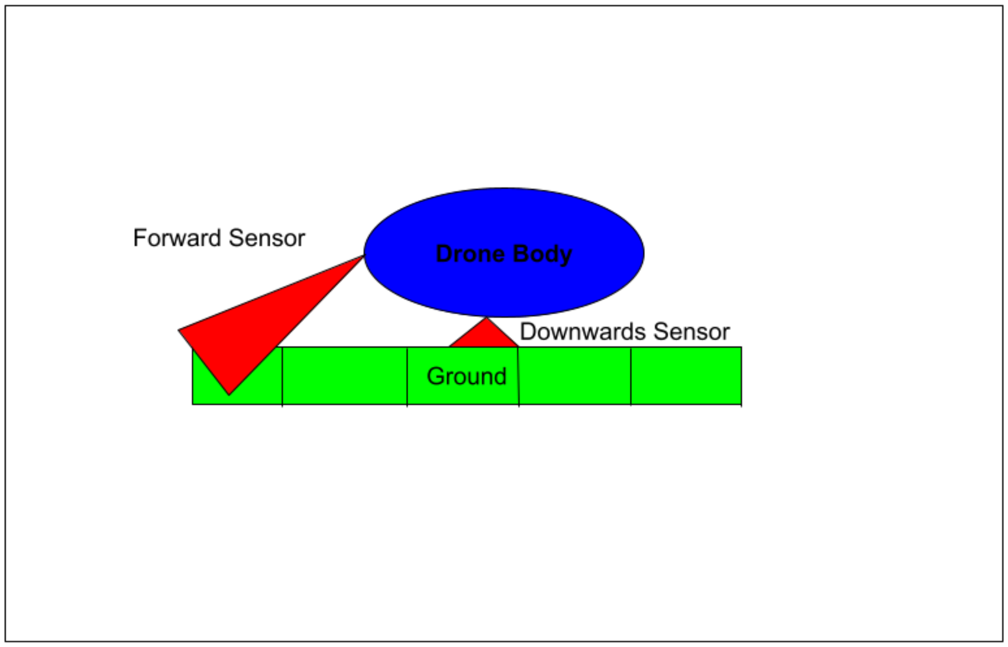


Fig. 5.20. Terrain Tracking Setup

To get a more accurate picture, two approaches were planned but not fully designed. The first, is to break the segment underneath the drone several blocks of a set distance, and measure the nearest ground value and record it as height. The drone accelerometer would be needed to estimate when the drone moves past this block and to the next block, where only the highest value will be recorded for that segment. Then, when the last block is passed behind the drone, the values can all be shifted over one space in memory, so the drone can keep track of the height underneath the drone, even if the ultrasonic is not covering the whole area. The values of blocks that are not being populated with current ultrasonic data would also need accelerometer data to adjust the measured height measurements, but the shift is equal for all blocks. This method enables the drone to keep track of important parts of data, but it isn't required to store every single recorded value, since that would take a lot of memory and slow down the system. To help reject disturbance, a boxcar average could be used to help reject noise or random errors while data is being recorded in every block. The size of the boxcar average would be determined by memory and sample rate, where as the distance chunks storing height values can be chosen as determine to be best during simulation, but even an arbitrary start of 1 m lengths will most likely be ok, and the lengths can be decreased of memory and computational ability is available. Once again, the system was not implemented, but is something that will be worth pursuing in future work on the project.

5.2.3 Other State Estimation

The estimation for the position of the drone and yaw angle using gps, accelerometer, and gyroscopic data was not completed since the autonomous system was not developed.

5.3 PCB Interface

5.3.1 Trace Width Calculation

The board house used to manufacture the PCB is OSH Park's 4 layer PCB manufacturing process. They recommend using 4pcb.com trace width calculator to determine rough minimum trace width of the PCB. The formula 4pcb uses is shown below in equation 5.1.

$$Width = \frac{\left(\frac{I[A]}{k * T[°C]^b}\right)^{\frac{1}{c}}}{Thickness[oz] * 1.378[\frac{mils}{oz}]} \qquad (5.3)$$

Where k = 0.024, b = 0.44, and c = 0.725 for internal layers of the IPC-2221 standard of PCB. For the external layer, k = 0.048, b = 0.44, c = 0.725[26]. These constants are results from curve fitting to the IPC-2221 curves in figure 5.21[27]. I is the current through the trace, T is the max temperature in celsius of the trace and thickness is the depth in oz of the copper trace. The resulting value of the width is in mils (1/1000 of an inch).

The curve in figure 5.21 shows the graph of current vs. trace width of the top and bottom layer of a IPC-2221 board with temperature curves ranging from 10°C to 100°C. Whereas, the curve in figure 5.22 shows the graph of current vs. trace width of the middle two 3.3V and GND layer ranging from 10°C to 45°C. The k, b, c constants from 4pcb are fitted to curves such as figure 5.20 and 5.21 to obtain their values. The curve came from UltraCAD's own experimentation and testing and is not done by the IPC organization, which is why trace widths are always recommended to be greater than calculated trace width.

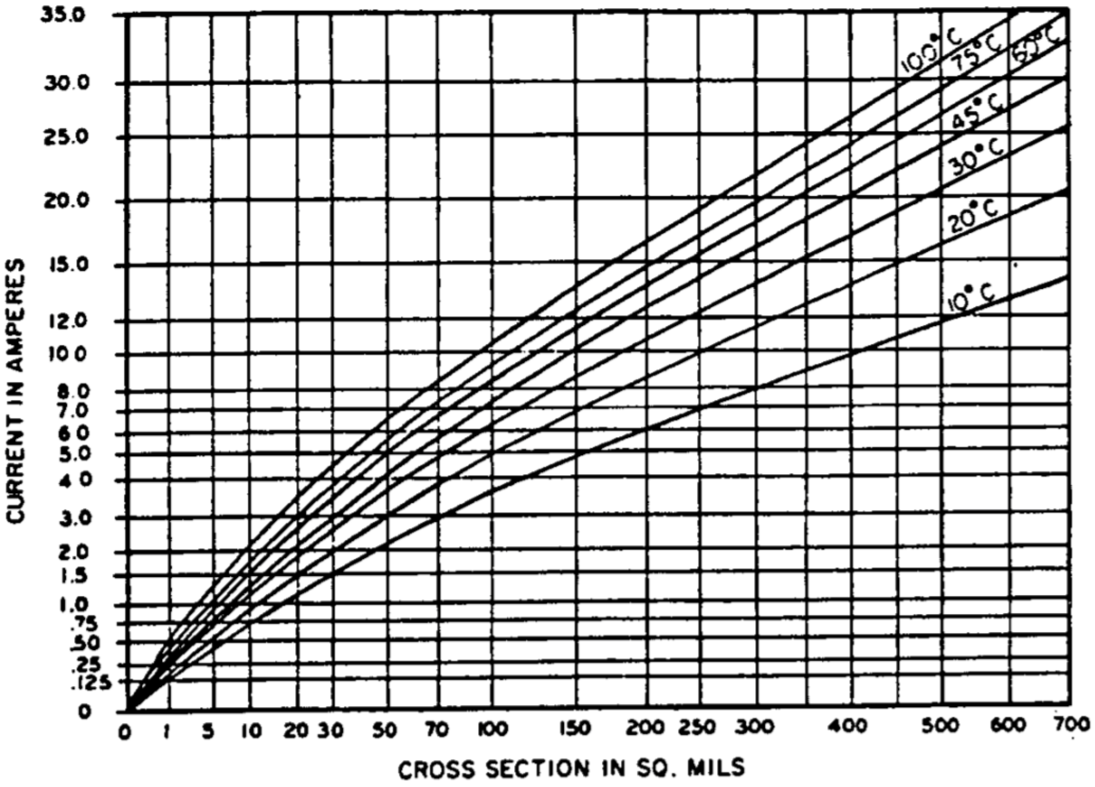


Fig. 5.21. External Conduction

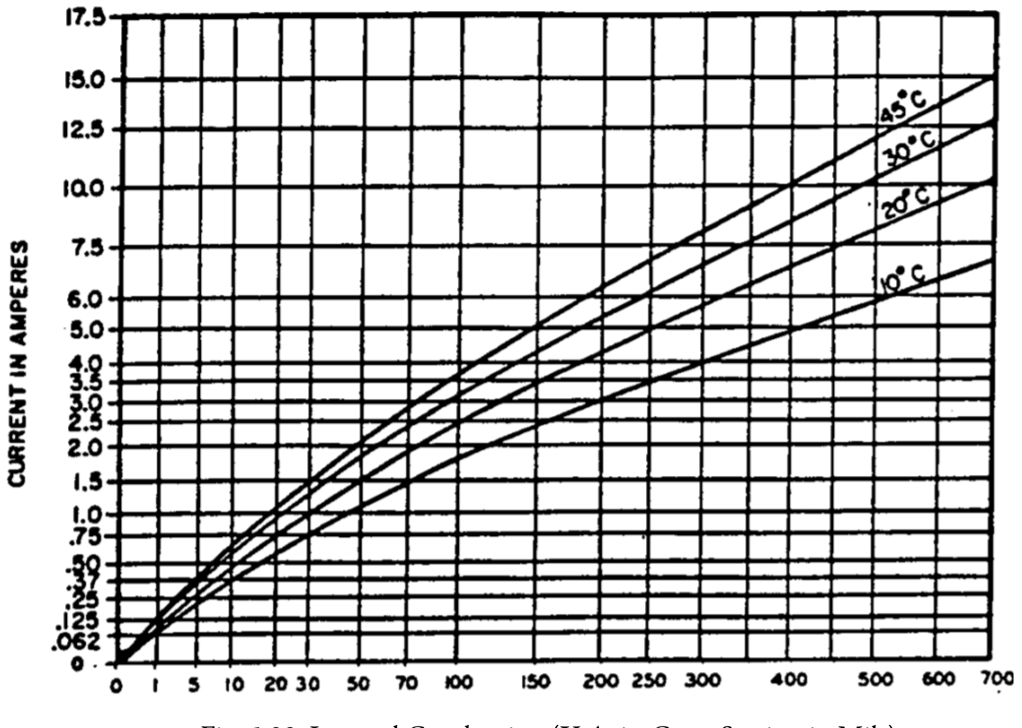


Fig. 5.22. Internal Conduction (X-Axis: Cross Section in Mils)

The 4 layer board from OSH Park has 1oz copper traces on both top and bottom and 0.5oz copper traces in the middle two layers of the PCB. The traces used by the drone are wider in width than the width the formula calculates.

Table 5.23
Calculated Trace Width Given Voltage, Current and Thickness

Voltage[V]	Current[A]	Thickness[oz/ft^2]	Width[mils]	Actual Width[mil]
12	14.949	1	224	250
5	0.59	1	2.59	16
3.3	0.534	0.5	11.7	16
1.8	0.268	1	0.872	16

3.3V has 0.5 copper thickness due to board manufacturing specifications for internal layers. 3.3V is chosen to be the power plane of the PCB because most of our sensor array uses 3.3V such as the MPL3115A2 altimeter, MT3339 GPS module, and MPRL balloon pressure sensor. 12V, 5V, and 1.8V rails are run on both top and bottom layers, which have 1 oz thickness. 300mil traces on both top and bottom are used to connect between the 12V battery to ESC pinouts as shown in figure 5.24.

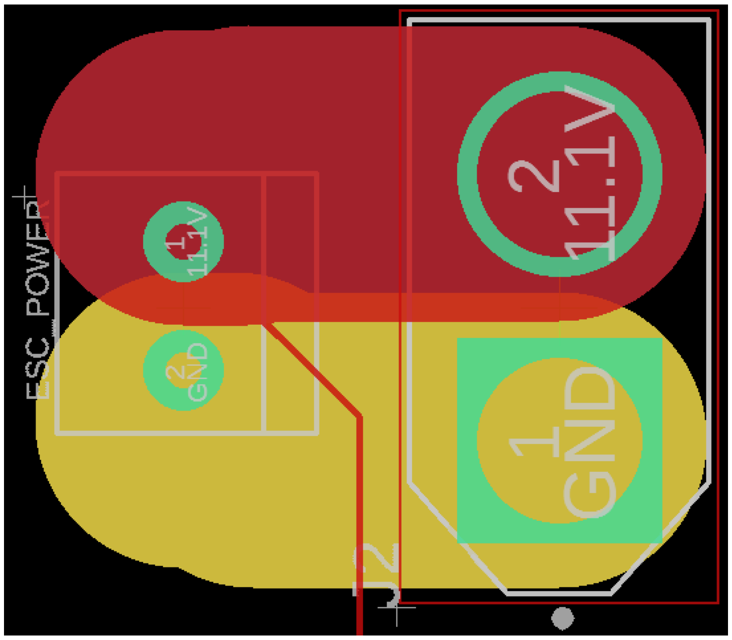


Fig. 5.24. Trace Between 12V Battery Pinout (right) to ESC Pinout (left)

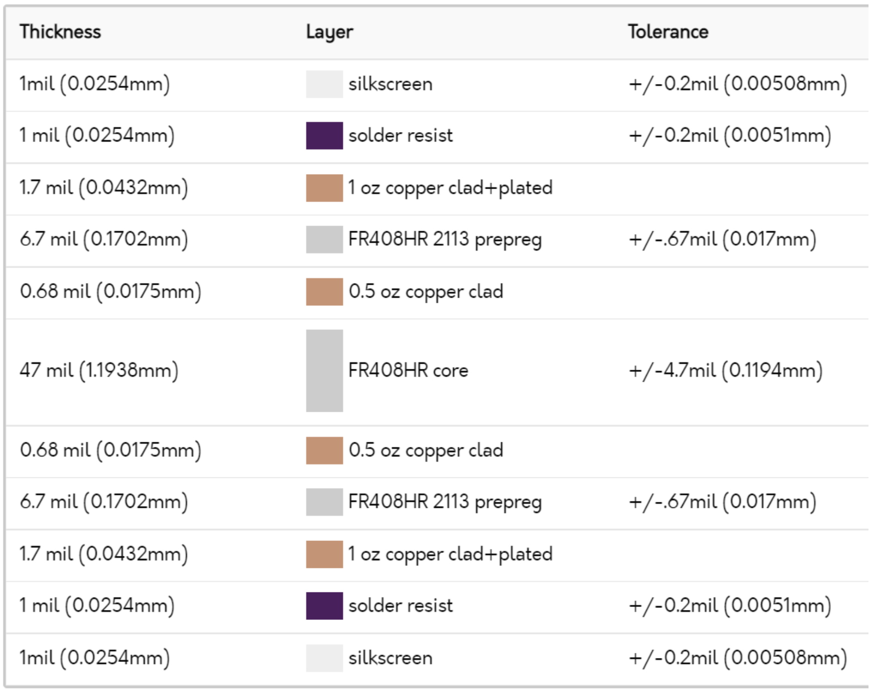
This is to ensure the required current by ESC is supplied through the traces without heating or burning up the traces which could prove detrimental to the PCB board and would require a new board to be installed. OSH Park requires a minimum trace width of 5 mils, and using the formula as a guideline, 16 mils are used for 5V, 3.3V, and 1.8V voltage regulators to ensure no traces are too thin to be burnt up. Also, due to having only one 1.8V voltage regulator and having the Raspberry Pi microprocessor and IMU needing 1.8V for power at opposite ends of the PCB, longer traces are needed to reach those components, so a wider trace width prevents risk of having high resistance in the power lines. Too large of a trace width will reduce top plane surface area for IC pads and overall separation of power and signal traces.

5.3.2 Board Manufacturer Design Rule Check

OSH Park has a downloadable DRC file that users can download for EagleCAD board design. This automatically sets the minimum required distance between via holes to pads, minimum amount of copper added around plated through-holes for soldering or via holes, and copper and core thickness relative to each other.

5.3.3 PCB 4-Layer Stackup

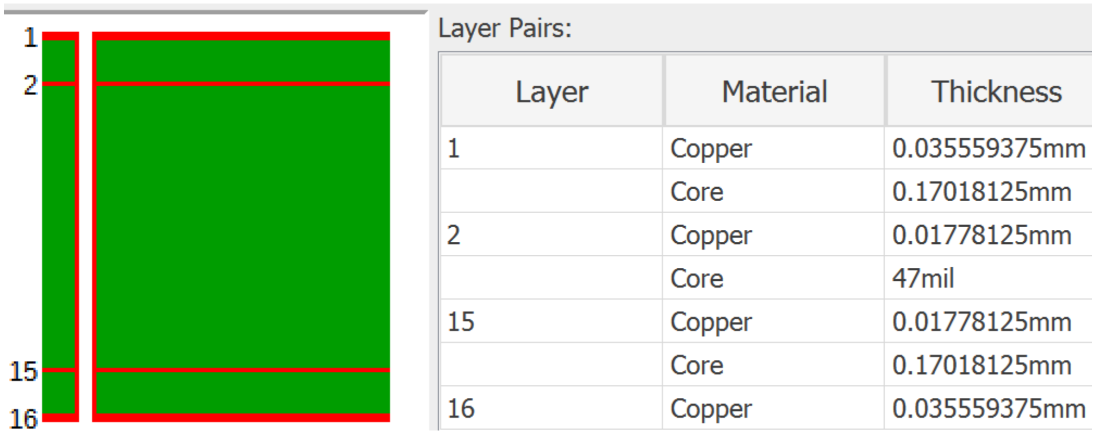
Figure 5.25 shows a helpful guide of layer-by-layer of thicknesses for silkscreen for text on the board, solder resist for top and bottom to repel solder on unwanted surface of the boards, 1oz and 0.5oz copper, and prereg/core of the board that separates the four copper planes from each other.



Thickness	Layer	Tolerance
1mil (0.0254mm)	⬜ silkscreen	+/-0.2mil (0.00508mm)
1 mil (0.0254mm)	🟪 solder resist	+/-0.2mil (0.0051mm)
1.7 mil (0.0432mm)	🟫 1 oz copper clad+plated	
6.7 mil (0.1702mm)	⬜ FR408HR 2113 prepreg	+/-.67mil (0.017mm)
0.68 mil (0.0175mm)	🟫 0.5 oz copper clad	
47 mil (1.1938mm)	⬜ FR408HR core	+/-4.7mil (0.1194mm)
0.68 mil (0.0175mm)	🟫 0.5 oz copper clad	
6.7 mil (0.1702mm)	⬜ FR408HR 2113 prepreg	+/-.67mil (0.017mm)
1.7 mil (0.0432mm)	🟫 1 oz copper clad+plated	
1 mil (0.0254mm)	🟪 solder resist	+/-0.2mil (0.0051mm)
1mil (0.0254mm)	⬜ silkscreen	+/-0.2mil (0.00508mm)

Fig. 5.25. 4-Layer Stackup of OSH Park Board

Once the 4 layer DRC file has been downloaded and loaded onto EagleCAD under DRC>File, the layer pairs under DRC>Layers should show tolerated thicknesses of the copper and core material as shown in figure 5.26 below.



Layer Pairs:

Layer	Material	Thickness
1	Copper	0.035559375mm
	Core	0.17018125mm
2	Copper	0.01778125mm
	Core	47mil
15	Copper	0.01778125mm
	Core	0.17018125mm
16	Copper	0.035559375mm

Fig. 5.26. EagleCAD 4-Stack Layer Thicknesses DRC Setting

5.3.4 Oscillator Placement

The primary 8MHz oscillator labeled X1 as shown in figure 5.27, and secondary 32.687kHz oscillator labeled Y1 in figure 5.28 below are placed no more than 1 inches away from the PIC32 microcontroller's pads on the left.

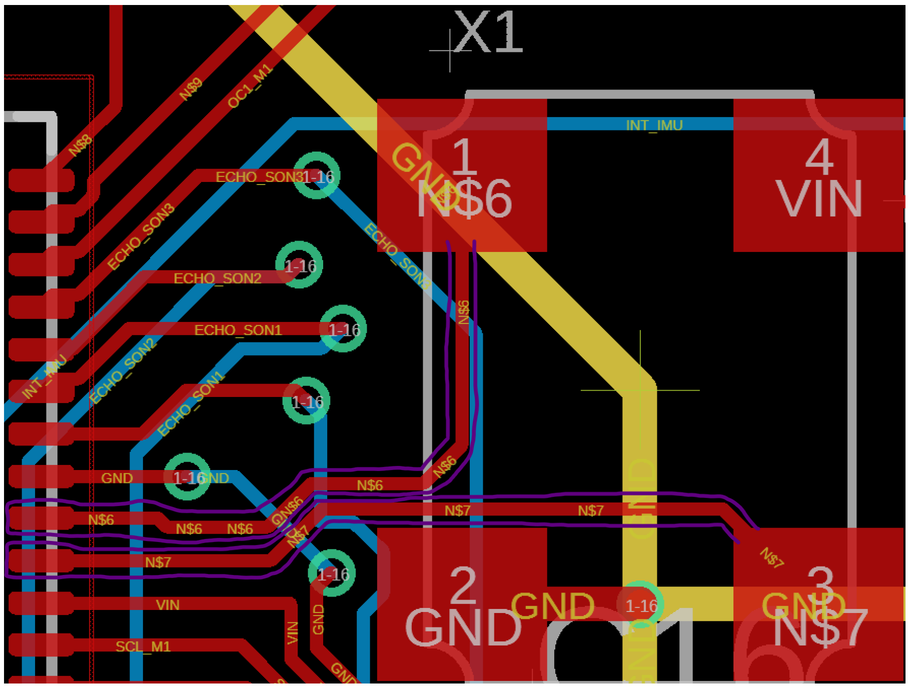


Fig. 5.27. 8MHz Oscillator Traces on the Right Leading to PIC32 Microcontroller Pads on the Left (circled in purple)

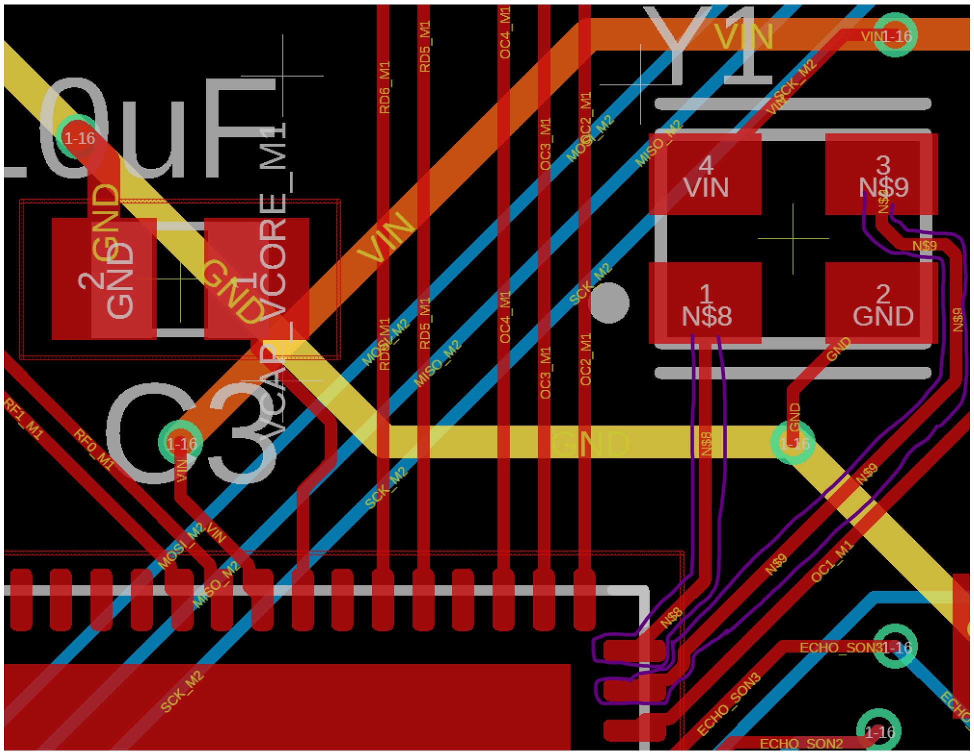


Fig. 5.28. 32.768kHz Oscillator Traces on the Top Right Leading to PIC32 Microcontroller Pads on the Bottom Left (Circled in Purple)

The 8MHz and 32.768kHz crystal oscillators are considered to produce the most noise out of any other components, so it should be placed as close to the microcontroller as possible to reduce trace length as much as possible. In Henry Ott's Electromagnetic Compatibility Engineering textbook section 16.1.4 [28], it is important to keep clock traces as short as possible as they emit the most EMI even though they make up less than 10% footprint of the board. The 8MHz was placed as close to the PIC32 microcontroller as first consideration to have minimal trace length before sensor IC footprints were placed to reduce EMF as much as possible and also to satisfy STR 6.0.0 (magnetometer interference shall be less than 10nT).

5.3.5 4-layer Board Stackup

A 4 in by 4 in PCB board was designed as the only real constraints of the board were gondola size which needed to house the PCB board underneath the balloon and also benefits to cutting down weight by having a small PCB board with an overall surface area of $16in^2$(not including the bottom signal layer of the PCB). A 4-layer stackup was chosen over the 2-layer stackup to allow for densely packed IC's that included microcontrollers, microprocessors, IMU, GPS module with surface mount U.FL coax antenna, two pressure sensors, and logic converters. Pinouts for ultrasonic sensors, high current battery, switching regulators, and transmitters and receivers also fit within the 4 in by 4 in footprint.

Under section 16.4.2.2 Four-Layer Boards of Henry Ott's ECE textbook[29], 4 layer boards improve electromagnetic compatibility and signal integrity of the sensor array and clocks of the oscillators by having a 0.0067 inch spacing between the top signal and ground layers. 4 layer stackup also reduces ground plane impedance because the ground plane is closer to the signal layer, whereas a 2 layer stackup does not have a ground plane at all, benefiting STR 6.0.0 to working towards a magnetic interference less than 10nT. By having an improved 4 layer board layer spacing as shown in figure 5.29 below compared to figure 5.30.

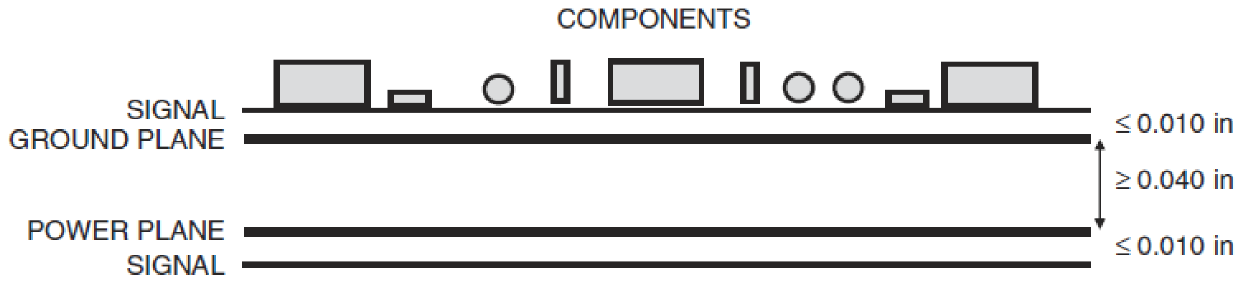


Fig. 5.29. Improved 4 Layer Board Configuration

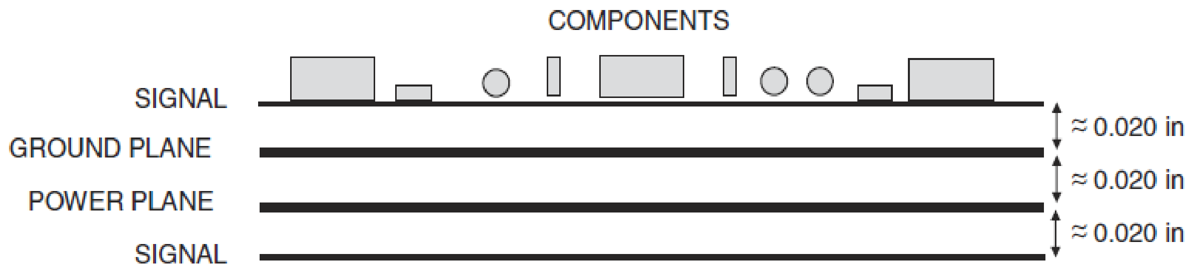


Fig. 5.30. Equally Spaced 4 Layer Board Configuration

OSH Park produces boards that satisfy figure 5.29 specifications as shown in figure 5.25, with the two inner planes separated by 0.047 in, satisfying the ≥ 0.04 in in figure 5.29, and by having the ground plane and signal plane closer results in a ground plane inductance of 0.085nH/in (shown in figure 5.30) compared to 0.13nH/in for a equally spaced 4 layer board in figure 5.29, also reducing ground noise by 35%.

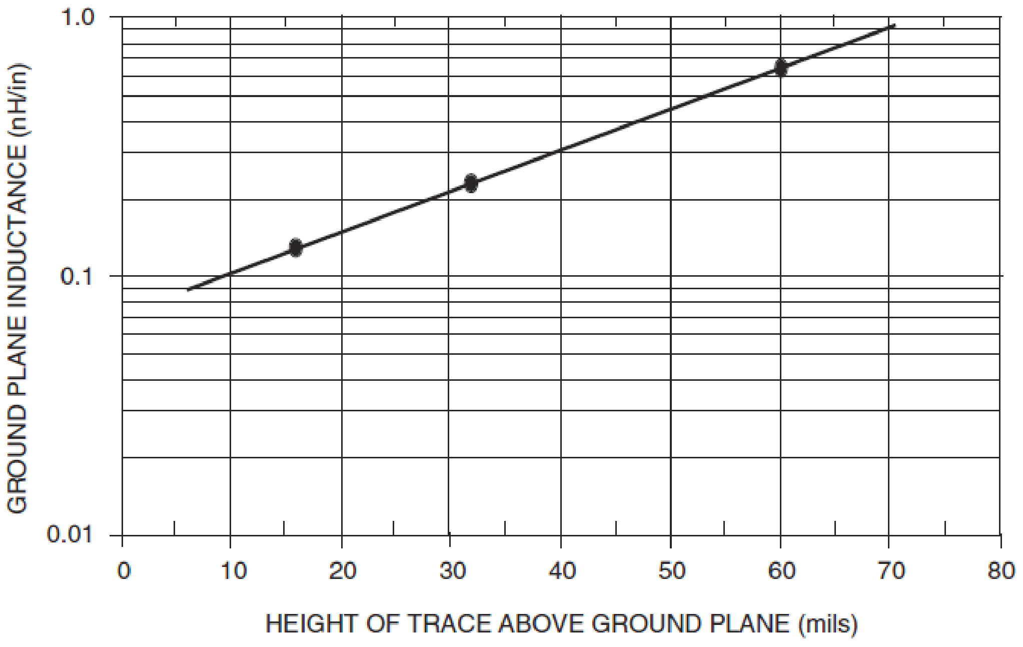


Fig. 5.31. Ground Plane Inductance vs. Trace Height

Having a 4 layer board allows for ground and power traces to be moved into the inner two planes, with the ground plane closest to the top traces and power plane closest to the bottom traces. The removal of power and ground traces on the top and bottom layers allows for more room on the signal trace layer to have more IC's to be soldered and allows for easier traces to be made by a novice engineer. However, the manufacturing time for a 4 layer PCB is considerably longer compared to 2 layer PCB because it requires two processes: the middle two layers to be etched as well as the outer two layers, whereas the 2 layer PCB only requires the top and bottom layers to be etched. As of June 2021, OSH Park board house ships 2 layer PCB within 4-5 days, whereas 4 layer PCB can only ship as fast as 9-14 days. This is why it is critical to follow strict PCB design guidelines before sending it to the boardhouse or else one can risk having a defectively designed PCB shipped and wasting money.

5.3.6 Power and Signal Location

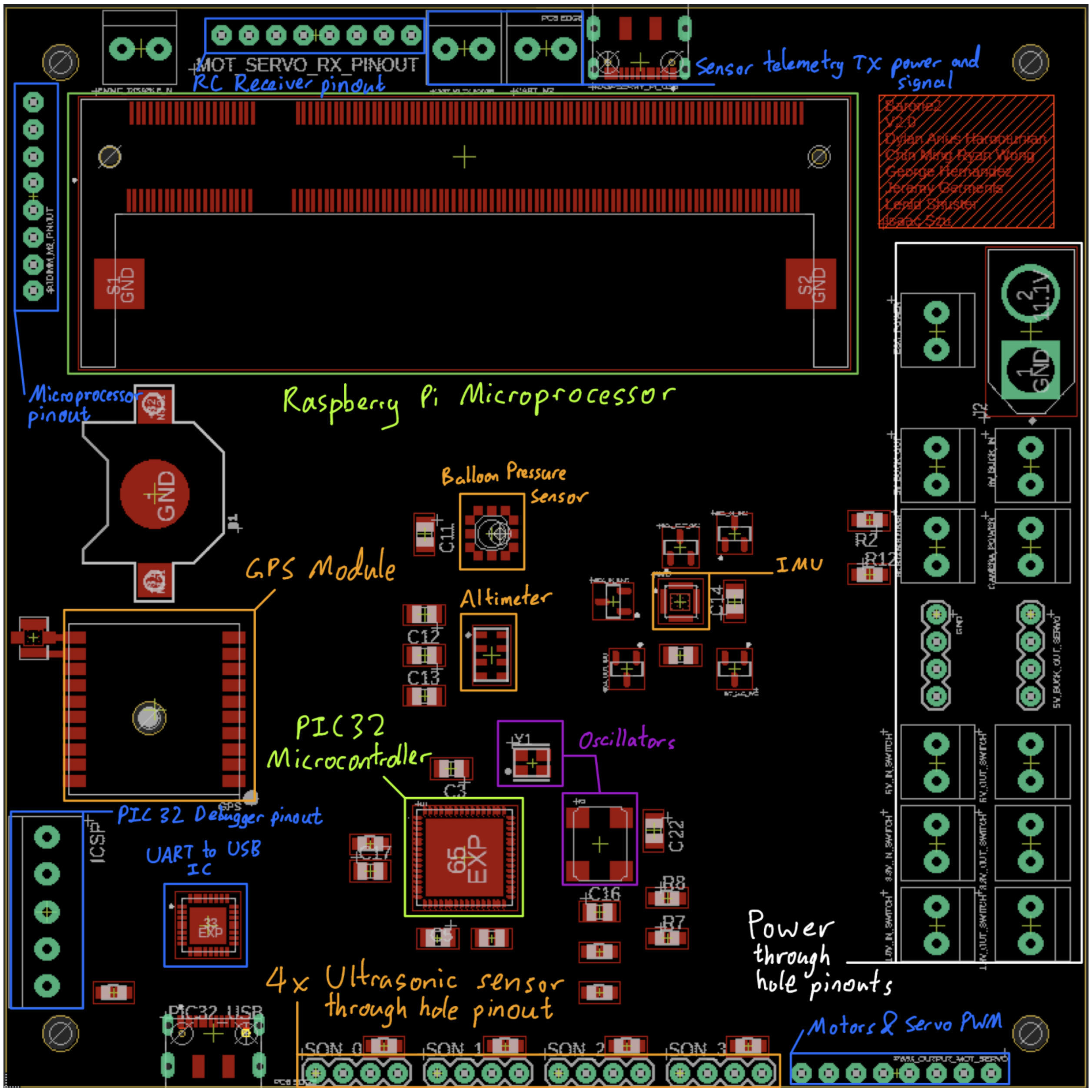


Fig. 5.32. PCB V2.0 Annotated

To reduce EMI of the sensors from power, pinouts for power are located on the right side of the PCB board as shown in figure 5.32. Sensors and microcontroller are placed in the middle to allow microcontroller's pinout pads to be able to reach the ultrasonic sensor pinouts, IMU, altimeter, balloon pressure sensor, GPS module and Raspberry Pi microprocessor.

All components except the GPS U.FL antenna located left of the GPS module is the only analog signal in the entire board which are digital. That is why the U.FL surface mount antenna is placed furthest from the power rails to reduce as much signal noise as possible. The signal traces for I2C, SPI, trigger and echo pins for sensor array are on the top and bottom layer, while the 3.3V power plane that powers the balloon pressure sensor, oscillators, altimeter, microcontroller and microprocessor is on the inner lower plane. The GND plane is the inner upper plane right below the top layer trace. Having a ground plane right below a low-frequency analog signal allows for a signal return path for the U.FL antenna [30], and high frequency clocks such as the 8MHz and 32.768kHz oscillators in figure 5.32.

Pinouts for all breakout boards/components are located on the side of the PCB to allow for ultrasonic sensor ribbon cables to be wired out through the side of the gondola. The ICSP pinout is located on the left bottom side to be closer to the PIC32 microcontroller as it works to flash the microcontroller, furthermore the signal pads connected to ICSP pinout is located on the left top and bottom side of the PIC32MX340F512H microcontroller, shortening trace length to the pinouts. The UART to USB converter IC is located right above the USB-C hardware port where USBD+ and USBD- signal traces are connected to for shorter trace length. The UART bus from the UART to USB IC then leads to the microcontroller's second UART pinout. The microprocessor pinouts located top left of figure 5.32 is conveniently located there for shorter trace length from the pinouts to the Raspberry Pi microprocessor in the top center, the same reasoning applies for the RC motor and servo radio RX as its PWM signals are inputted to the Raspberry Pi microprocessor as well.

5.3.7 Power Verification

Upon receiving the PCB V2.0 board, the first verification process is to solder the two 5V switching regulators shown in figure 5.33 below.

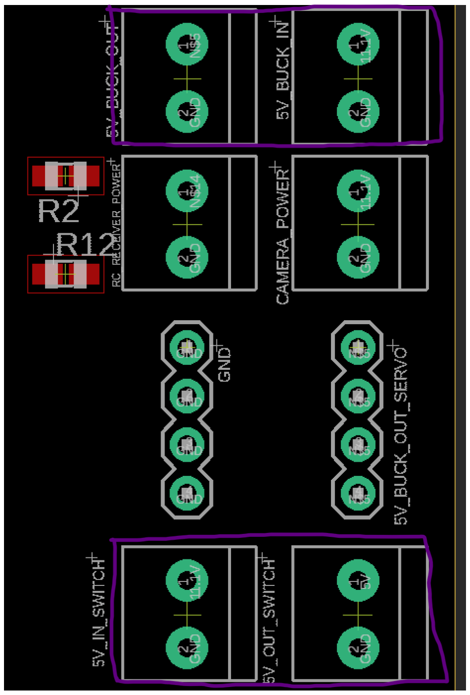


Fig. 5.33. 5V Switching Regulator Pinouts (Boxed in Purple)

Using breadboard wires to solder the power input from the 11.1V rail and ground pin, shown in figure 5.34.

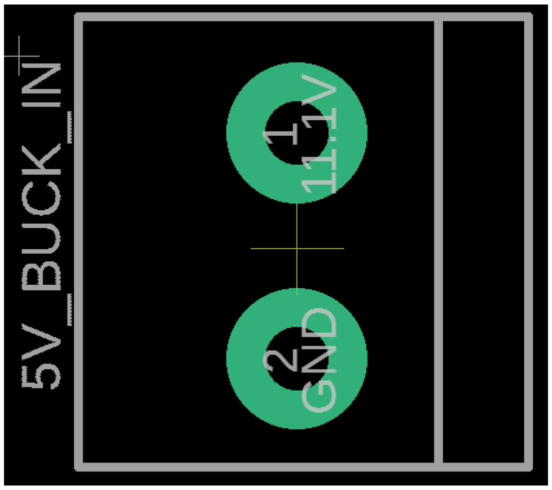


Fig. 5.34. Zoomed in Version of Figure 5.33 Input Voltage from 11.1V

Then solder the through hole pins for the 5V power out from the switching regulator and ground pin as shown in figure 5.35.

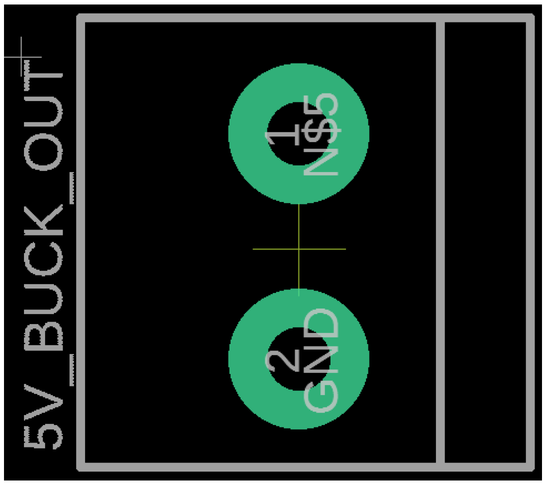


Fig. 5.35. Zoomed in Version of Figure 22 Output Voltage for 5V

To provide power to the 5V switching regulator, solder the XT60 header for the 11.1V battery to the J2 header shown in figure 5.36.

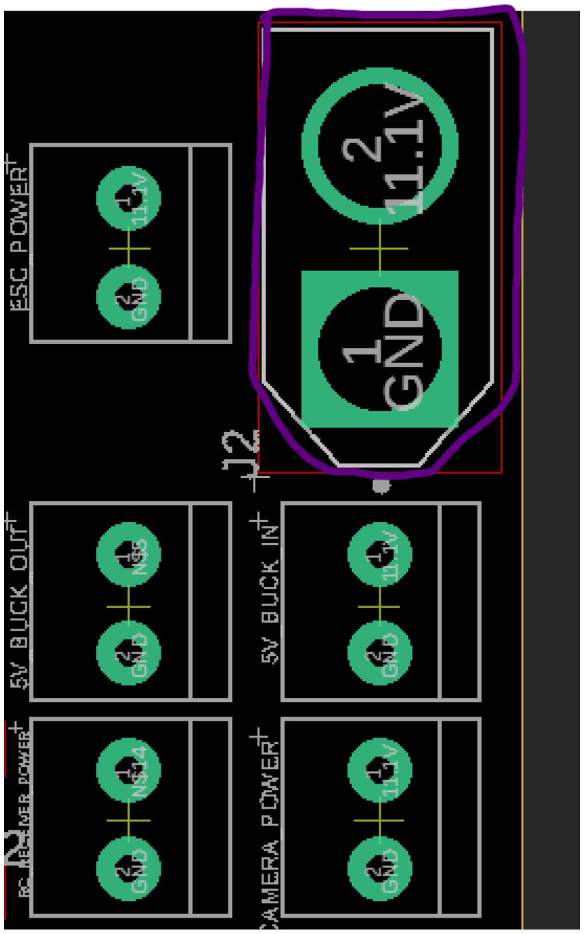


Fig. 5.36. 11.1V XT60 Battery Input Header

To test out the power delivered by the 5V switching regulator, use a voltage meter and probe the following through holes in figure 5.37 with the negative probe on one of the 4 ground pinouts labeled GND left of the servo power.

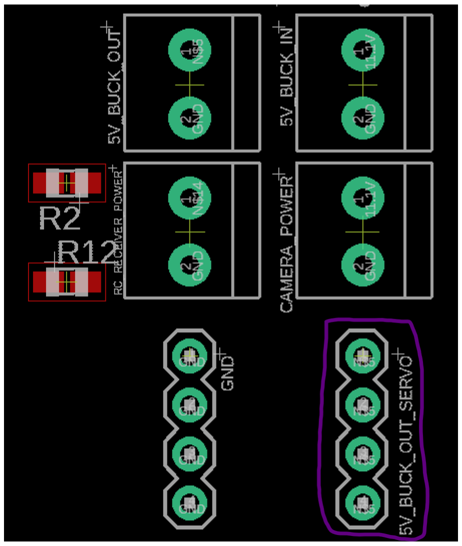


Fig. 5.37. Servo Power Pinout Test Point (Circled in Purple)

The second 5V switching regulator located beneath the servo power pinouts, as shown in figure 5.38, provides 5V power to the ultrasonic sensors and the Raspberry Pi microcontroller.

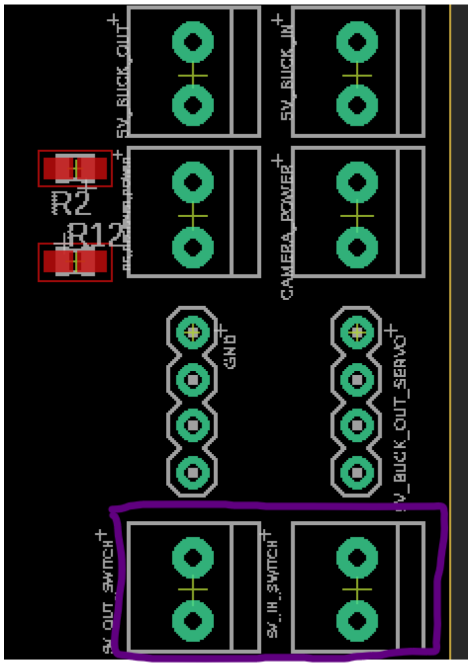


Fig. 5.38. 5V Switching Regulator Pinouts for Raspberry Pi Microprocessor and Ultrasonic Sensor

To verify 5V power coming out of the second switching regulator, use a multimeter to probe the Raspberry Pi CM3+ compute module's pins 197-200 shown in figure 5.39 below.

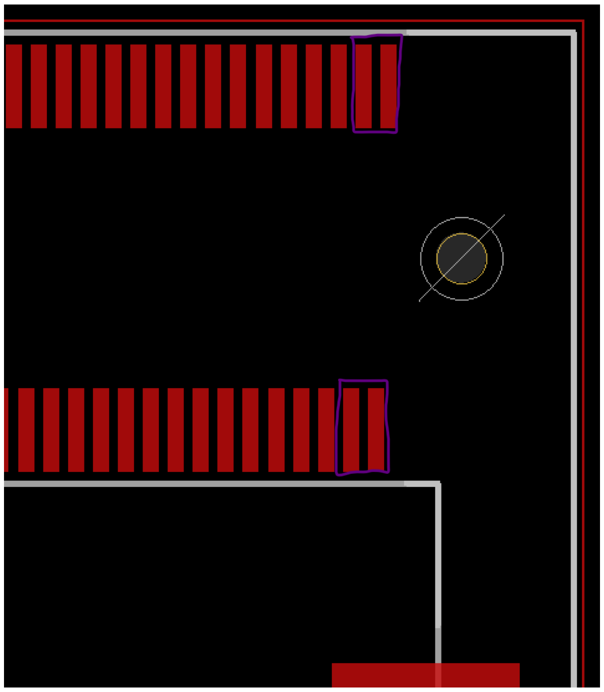
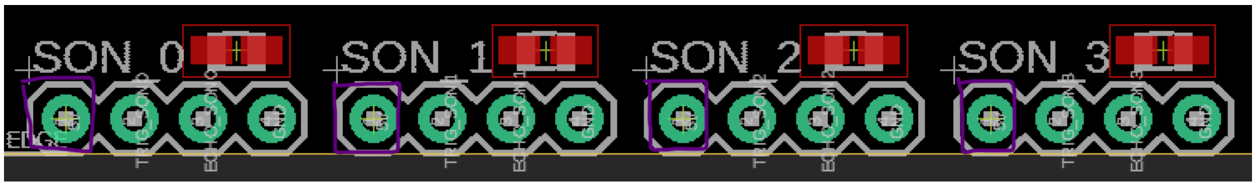


Fig. 5.39. Raspberry Pi Microcontroller 5V Pads (Boxed in Purple)

Verify 5V going into each of the four ultrasonic sensor power pins located on the bottom of the board as shown in figure 5.40 below.



Fig, 5.40. Ultrasonic Sensor 5V Test Points (Boxed in Purple)

Next, solder the 3.3V switching regulators to these locations shown in figure 5.41 below.

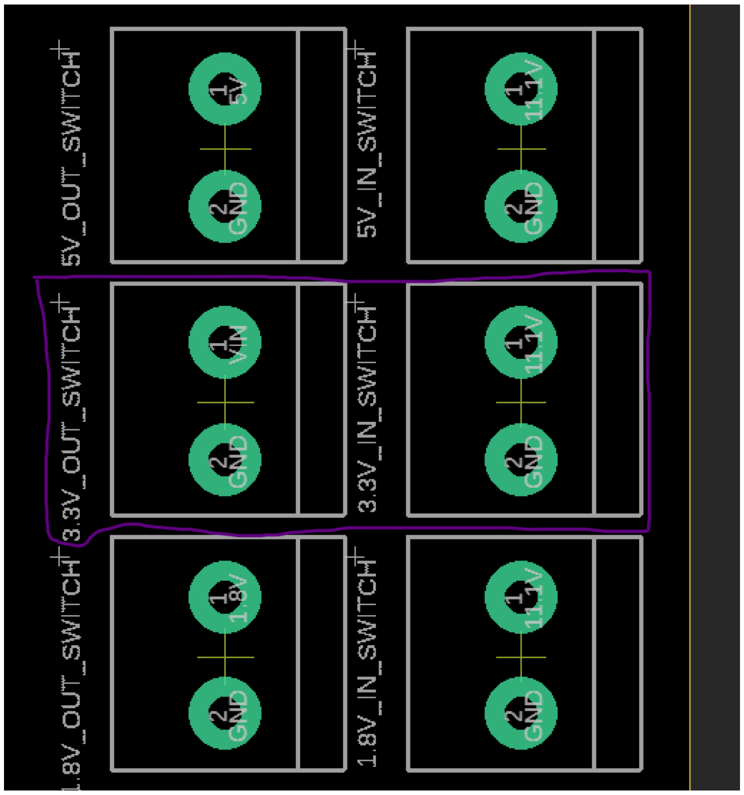


Fig. 5.41. 3.3V Switching Regulator Through Hole Solder Points

Ensure 11.1V battery is plugged into the PCB board and now the power plane of the board should have 3.3V supplying to the altimeter (MPL3115A2 barometric and temperature sensor). Probe the following boxed pads in figure 5.42 below and make sure 3.3V is present on the voltage meter.

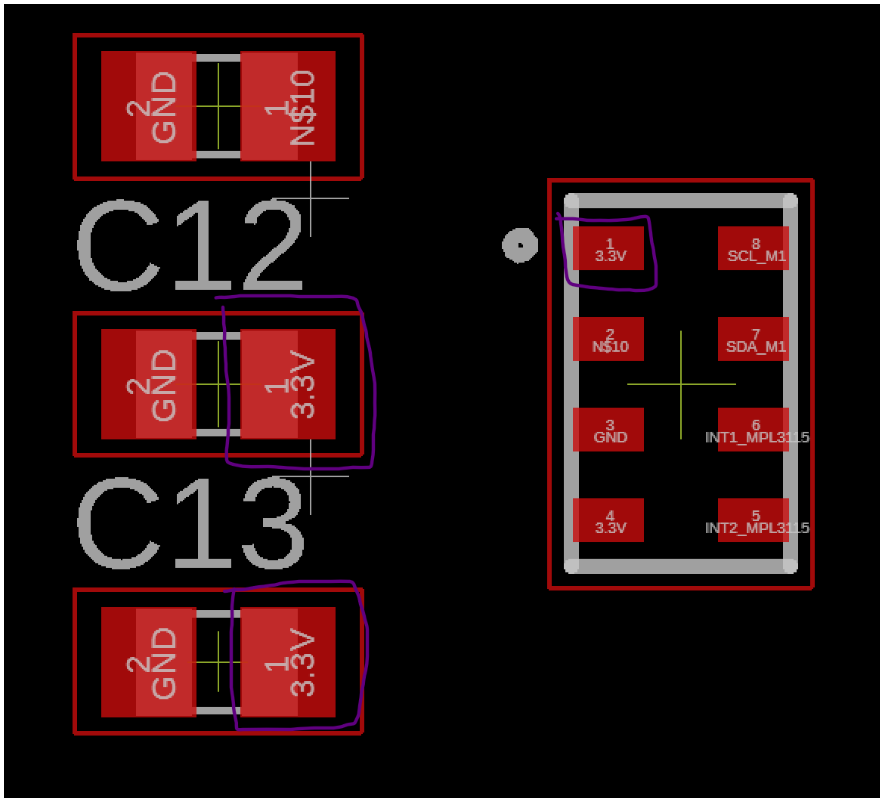


Fig. 5.42. Altimeter Sensor 3.3V Probe Points (Boxed in Purple)

Verify the MPR barometric sensor for the balloon right above the altimeter shown in figure 5.43 below.

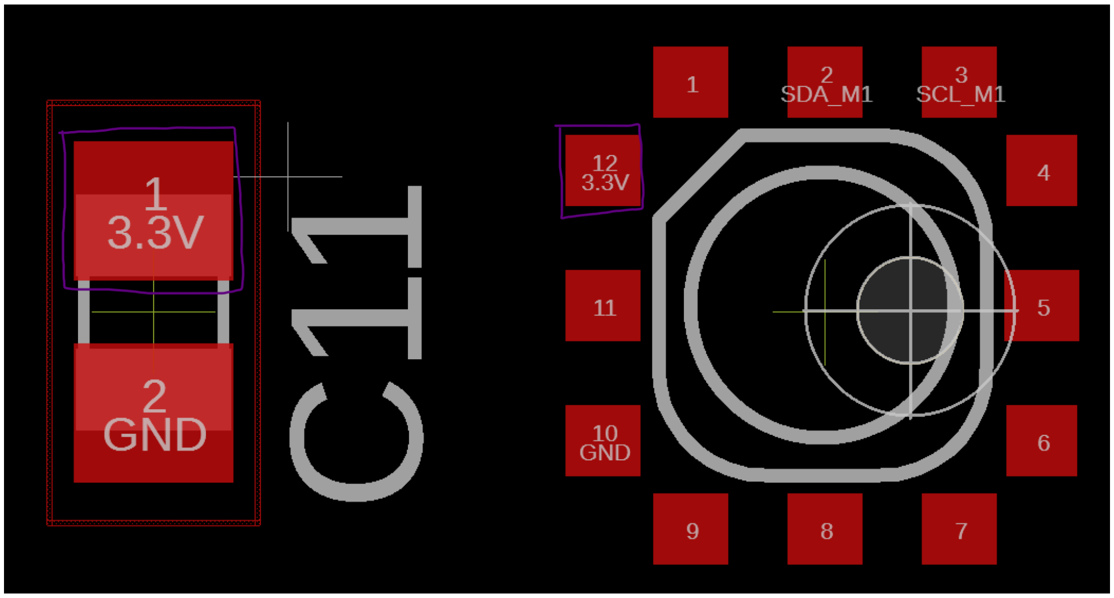


Fig. 5.43. Balloon Pressure Sensor 3.3V Probe Points (Boxed in Purple)

Next, probe the 3.3V for the GPS module Vin located center left of the board as shown in figure 5.44 below.

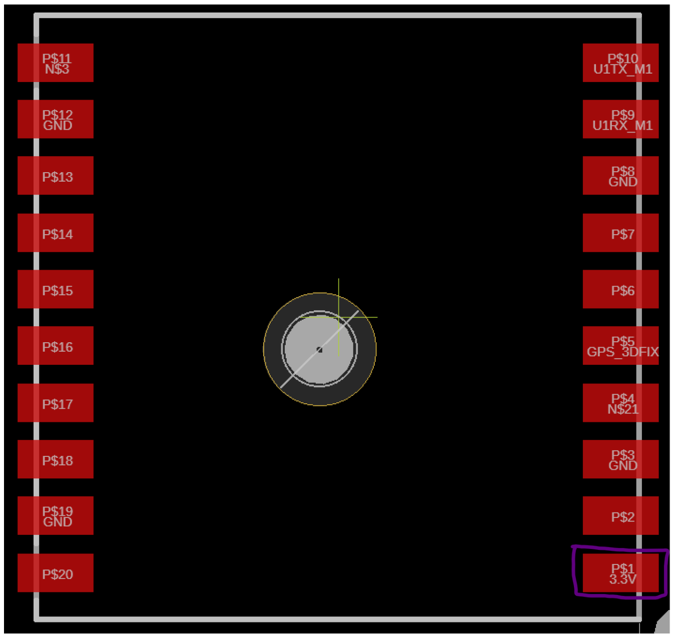


Fig. 5.44. GPS Module 3.3V Probe Points (Boxed in Purple)

Most of the sensor array power input should be verified by now, except for the ultrasonic sensor and IMU. For microcontroller and oscillator power verification, probe the following in figure 5.45, that is the PIC32MX340F512 pins 10, 26, 38, and 57.

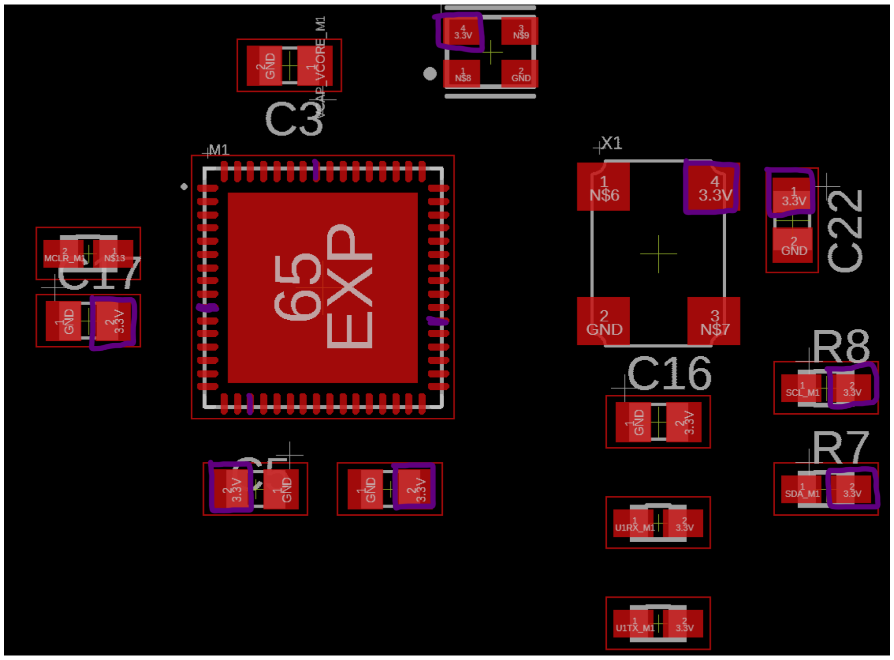


Fig. 5.45. Micrcontroller, Oscillators, and Bypass Capacitors 3.3V Probe Points (Boxed and Colored in Purple)

Next, probe the Raspberry Pi microprocessor pinouts for 3.3V on pins 39, 40, 41, 42, 189, 190, 191, 192, 193, and 194 as shown in figure 5.46 below.

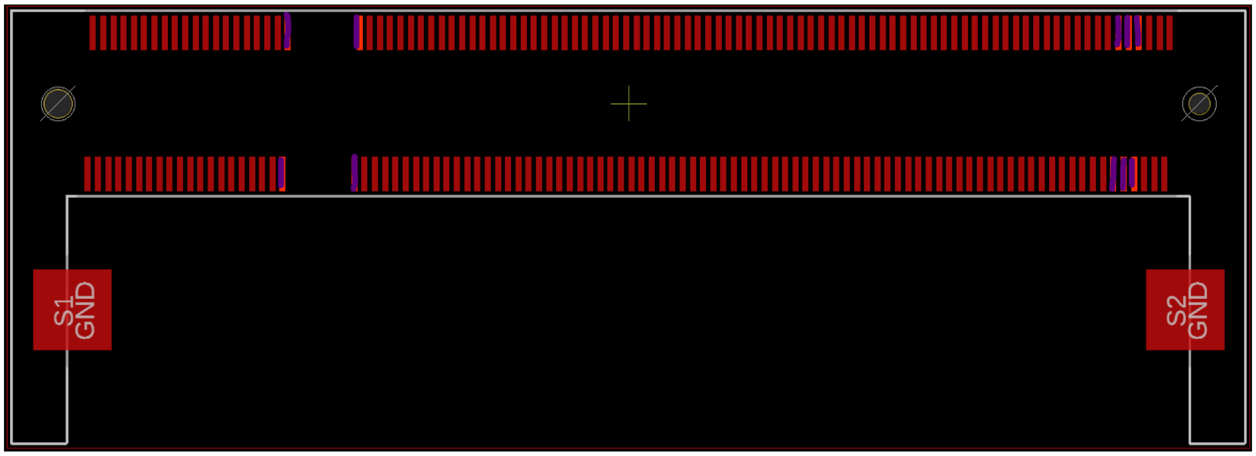


Fig. 5.46. Raspberry Pi Microprocessor 3.3V Probe Points (Colored in Purple)

For 1.8V power delivery to IMU and Raspberry Pi microprocessor, solder the 11.1V through hole to the 1.8V switching regulator breakout board and the 1.8V voltage input to the PCB through hole shown in figure 5.47 below.

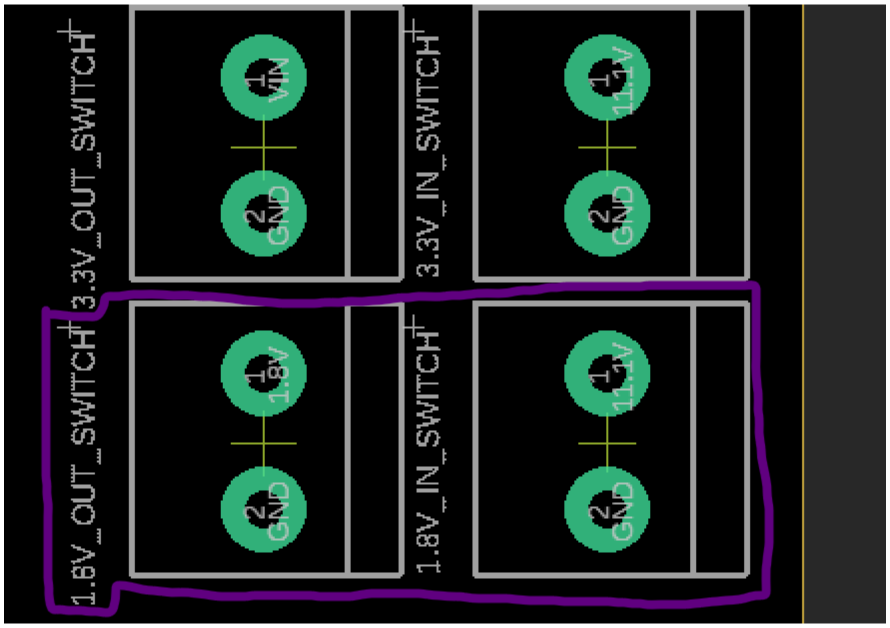


Fig. 5.47. 1.8V Switching Regulator Pinout

To verify that 1.8V is being supplied by the switching regulator to the IMU probe the IMU's pins 8, 13, and 22 with the voltage meter and check for 1.8 volts, as well as probing the C14 bypass capacitor and the logic converters.

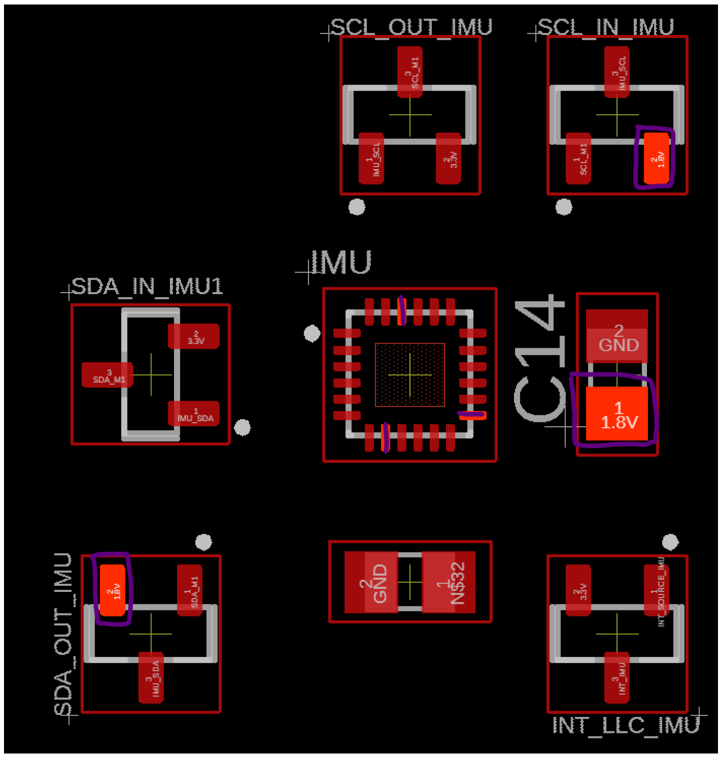


Fig. 5.48. IMU 1.8V Probe Points on Logic Converters, and Bypass Capacitor (Boxed and Colored in Purple)

The raspberry pi microprocessor also uses 1.8V, again probe the microprocessor's pins 183-186 using the voltage meter and check for 1.8V in figure 5.49 below.

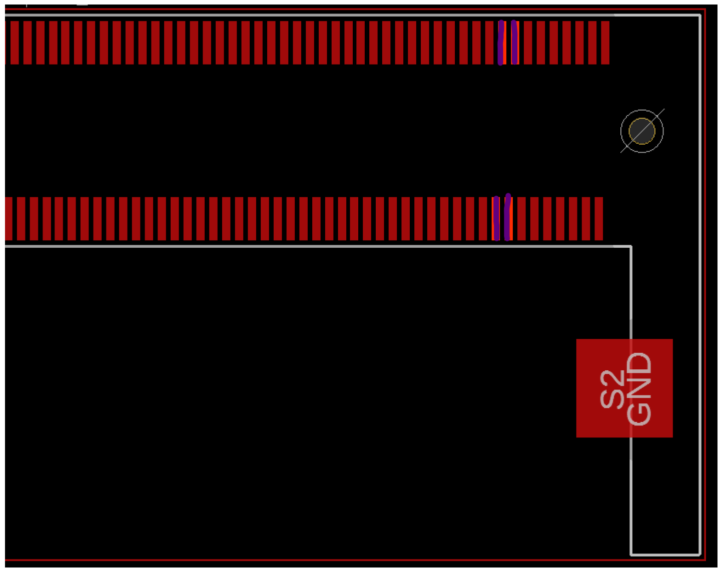


Fig. 5.49. Left Hand Side of Raspberry Pi Microprocessor 1.8V Probe Points (Colored in Purple)

5.3.8 I2C Bus Verification

Using the continuity function on the multimeter, check the I2C clock signal between the PIC32 microcontroller to the balloon pressure sensor, altimeter, and IMU as shown in figure 5.50 below. The multimeter, when set to continuity mode, should output a static beep sound when two probes are touching any two pads mentioned above. If there is no sound it means the trace is not connected and either the board house did not manufacture the PCB correctly or it is a design error where the engineer made a critical error in forgetting to place a trace path between the two pads.

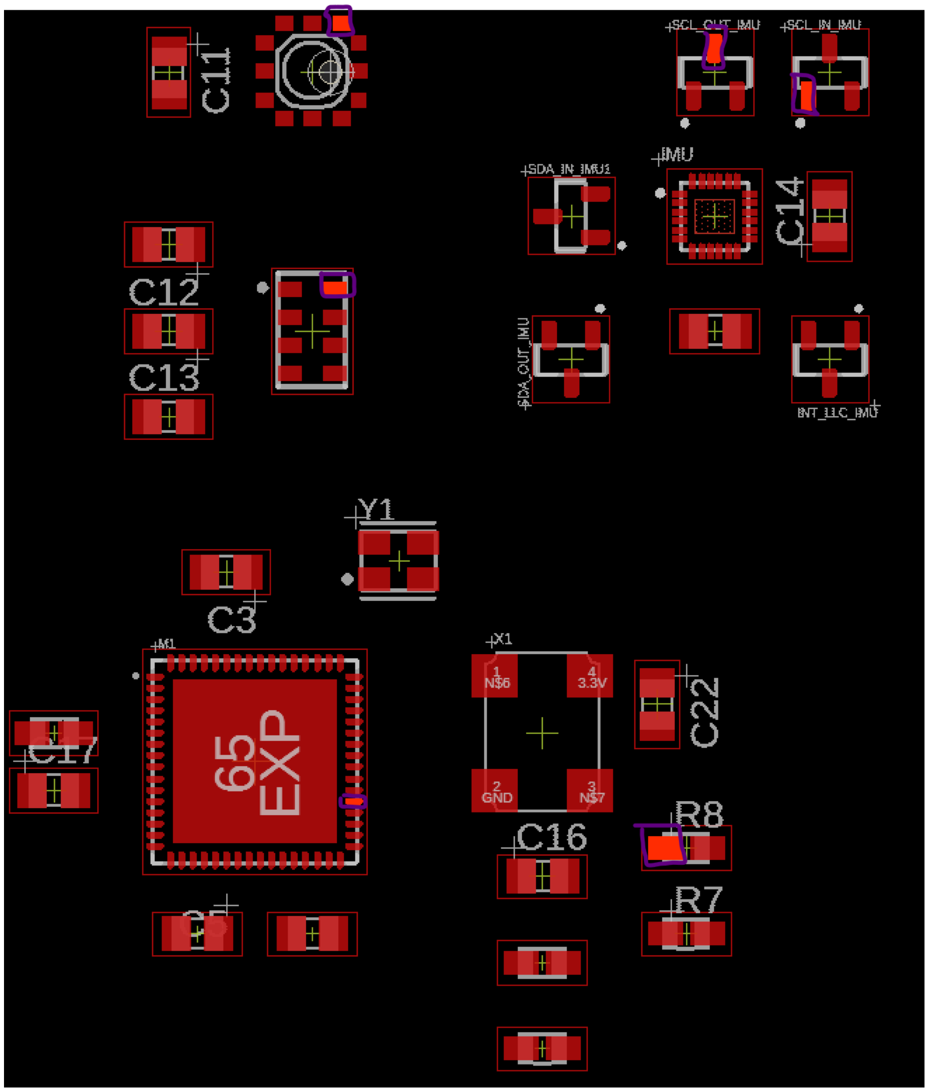


Fig. 5.50. I2C Clock Signal Test Points for Continuity Test (Boxed in Purple)

Check the continuity of the data signal or SDA between the microcontroller to the same sensor array as stated above in I2C Bus Verification subheading, as shown in figure 5.51 below.

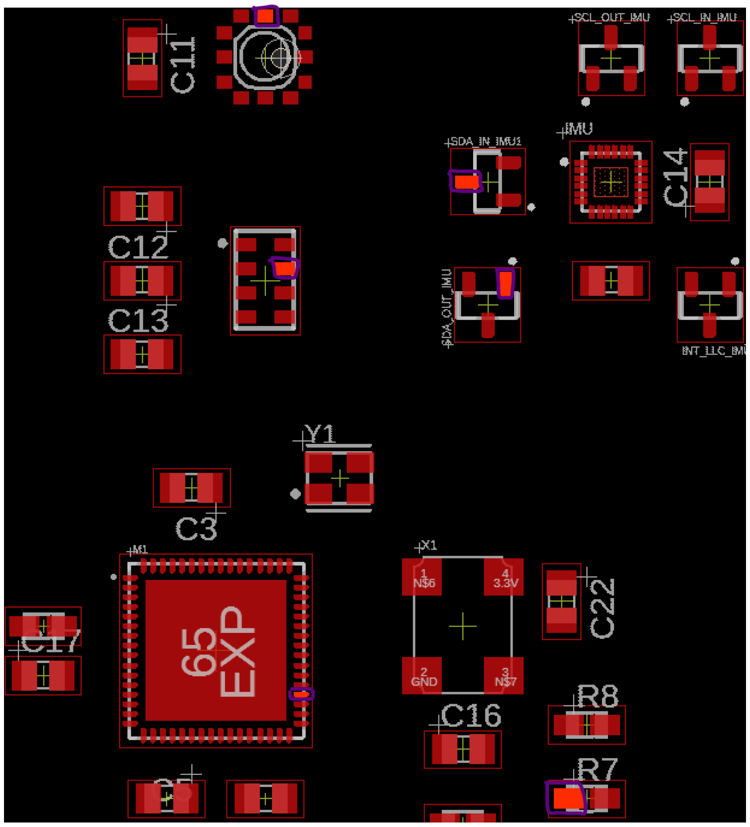


Fig. 5.51. I2C Data Signal Test Points for Continuity Test (Boxed in Purple)

5.3.9 UART Bus Verification

The UART signal bus from PIC32 microcontroller has two separate buses, first U1TX running to the GPS module and the second one U2TX running to the FTR232RQ UART to USB IC. To verify the traces are connected between them, set the multimeter to continuity mode and place the probes on the following pads shown in figure 5.52.

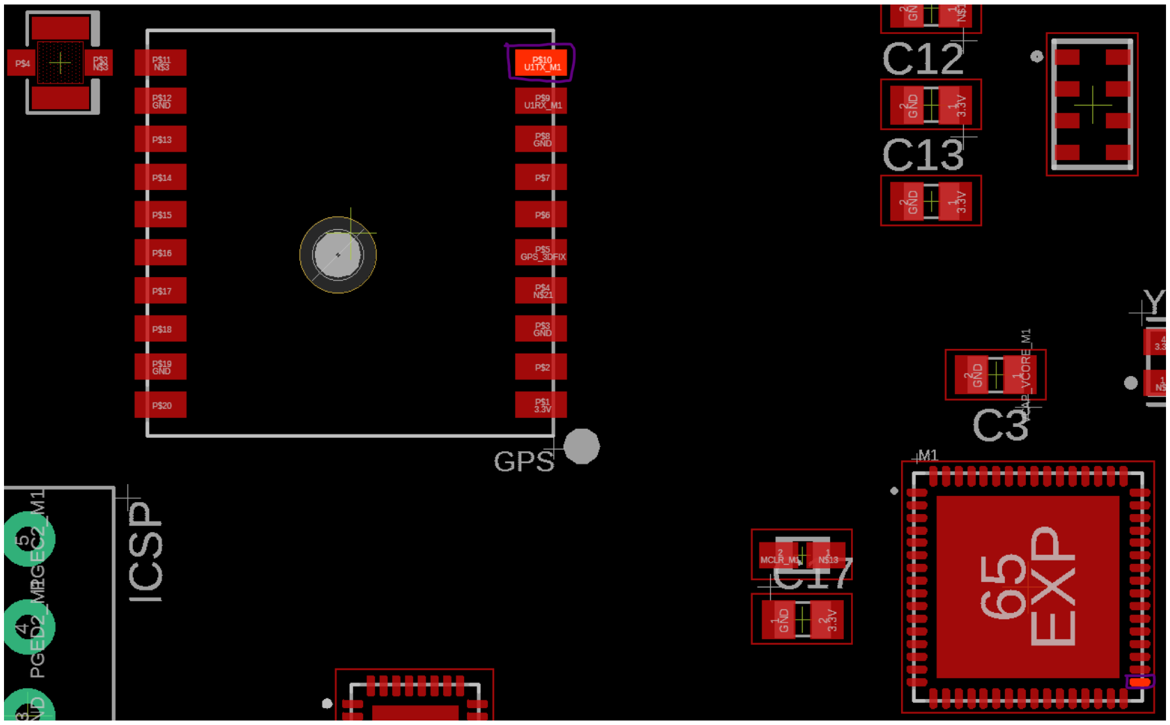


Fig. 5.52. UART Microcontroller Transmitter Continuity Test Point to GPS Module (Boxed in Purple)

Check the continuity of the RX of microcontrollers UART 1 with the GPS module as shown in figure 5.53 below.

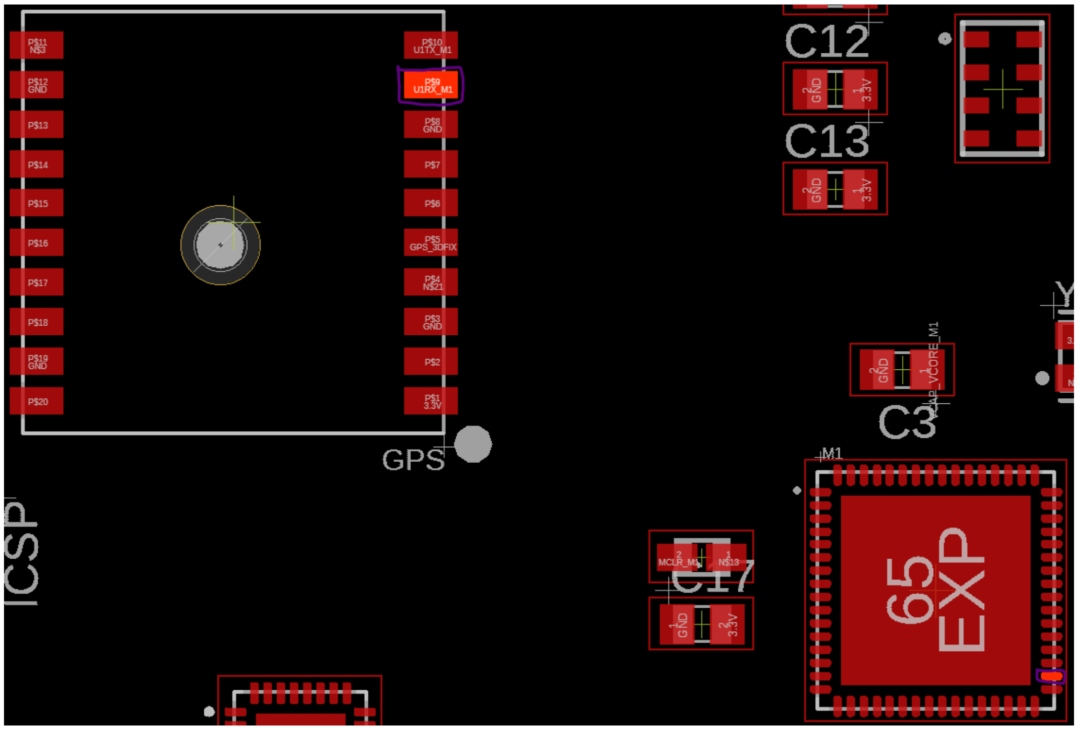


Fig. 5.53. UART Microcontroller Receiver Continuity Test Point to GPS Module (Boxed in Purple)

Now to verify the continuity of the second UART bus between the microcontroller to the FT232RQ UART to USB IC to make sure printf statements will work from the microcontroller to a personal computer serial terminal. Probe the transmitter pin of the microcontroller and the receiver pin of the UART to USB IC as shown in figure 5.54.

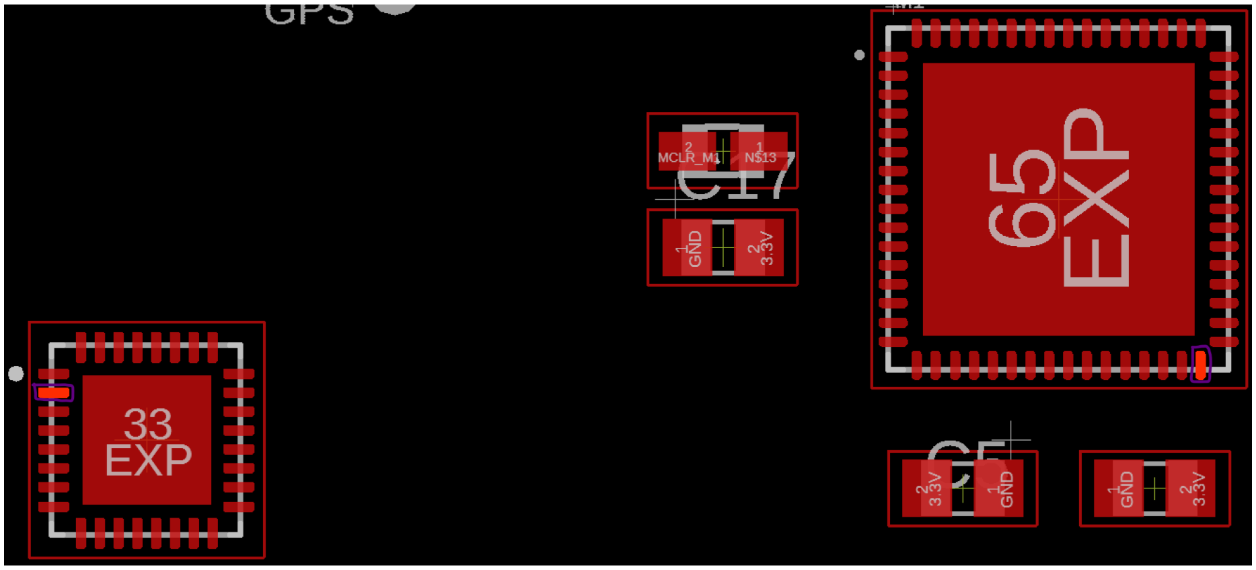


Fig. 5.54. UART Microcontroller Transmitter Continuity Test Point to FT232RQ IC (Boxed in Purple)

Next, verify the transmitter pad of the microcontroller and the receiver pad of the FT232RQ IC as shown in figure 5.55.

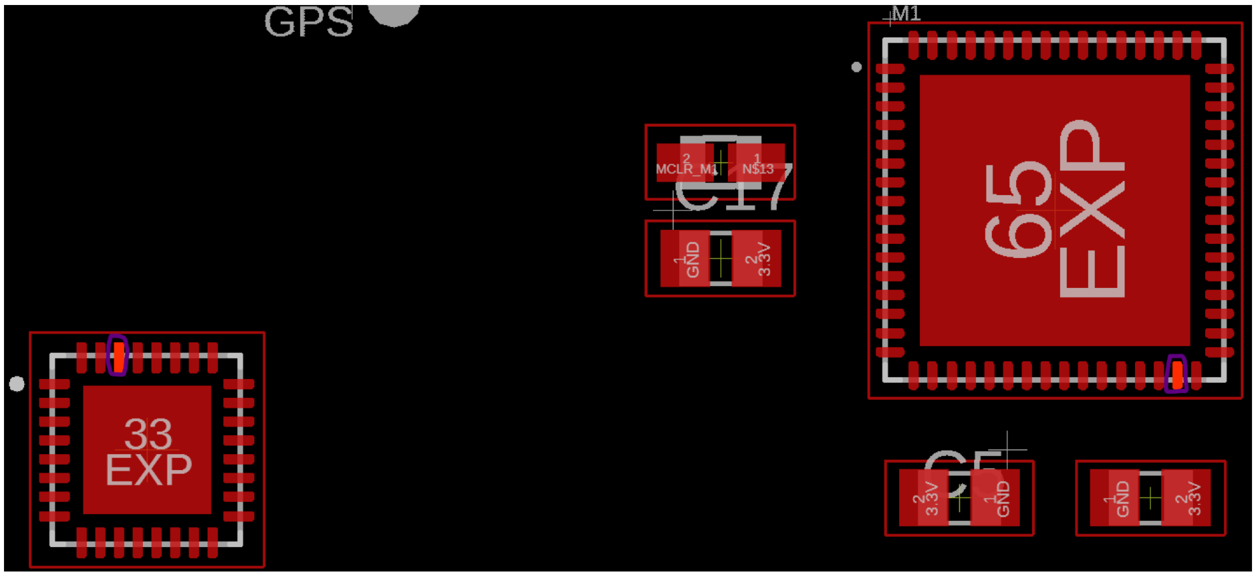


Fig. 5.55. UART Microcontroller Receiver Continuity Test Point to FT232RQ IC (Boxed in Purple)

5.3.10 SPI Bus Verification

The SPI traces connect the PIC32 microcontroller to the Raspberry Pi microprocessor and are connected by 4 signal traces, the MOSI, MISO, SCK, and CS. Verify the continuity of the MOSI trace by probing pin 33 of the Raspberry Pi microprocessor and pin 11 of the PIC32 microcontroller. Verify MISO trace by probing pin 29 of the Raspberry Pi microprocessor and pin 12 of the PIC32 microcontroller. Verify SCK trace by probing pin 35 of the Raspberry Pi microprocessor and pin 13 of the PIC32 microcontroller, and finally verify CS of PIC32 microcontroller is grounded since it is the only daughter device to the Raspberry Pi microprocessor parent.

5.3.11 Ultrasonic Sensor Signal Bus Verification

Using the continuity setting of the multimeter, probe the trigger pin of the first ultrasonic sensor named TRIG_SON0 shown in figure 5.56 below with pin 1 of PIC32 microcontroller, and probe pin 42 of the microcontroller with ECHO_SON0 pin of the ultrasonic sensor.

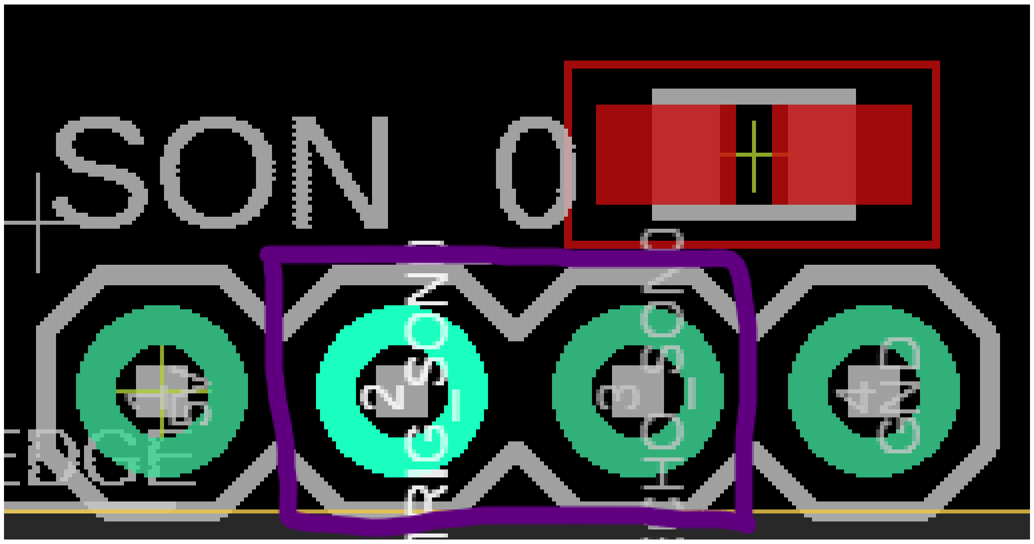


Fig. 5.56. 1st Ultrasonic Sensor Trigger and Echo Probe Point (Boxed in Purple)

Repeat the above steps for SON_1, SON_2, SON_3.

- SON_1
    - TRIG_SON1 probe with pin 2 of microcontroller
    - ECHO_SON1 probe with pin 43 of microcontroller
- SON_2
    - TRIG_SON2 probe with pin 3 of microcontroller
    - ECHO_SON2 probe with pin 44 of microcontroller

- SON_3
  - TRIG_SON3 probe with pin 4 of microcontroller
  - ECHO_SON3 probe with pin 45 of microcontroller

5.3.12 Actuator Verification

Verification of continuity between the servo and motor to PIC32 microcontroller. The test points for the motors and servos are shown in figure 5.57 below.

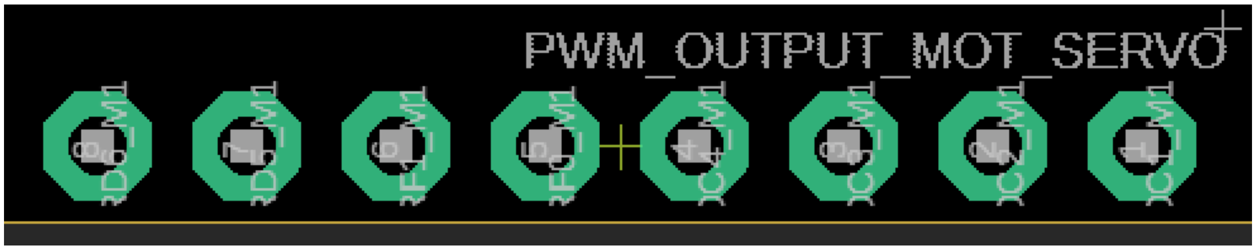


Fig. 5.57. Motor and Servo PWM Output Pinouts

Using the continuity setting of the multimeter, probe the following:

- Pin 8 of figure 5.57 with pin 54 of PIC32 microcontroller
- Pin 7 of figure 5.57 with pin 53 of PIC32 microcontroller
- Pin 6 of figure 5.57 with pin 59 of PIC32 microcontroller
- Pin 5 of figure 5.57 with pin 58 of PIC32 microcontroller
- Pin 4 of figure 5.57 with pin 51 of PIC32 microcontroller
- Pin 3 of figure 5.57 with pin 50 of PIC32 microcontroller
- Pin 2 of figure 5.57 with pin 49 of PIC32 microcontroller
- Pin 1 of figure 5.57 with pin 46 of PIC32 microcontroller

After the traces for each of the PWM traces have been verified, use a 8-channel logic analyzer and connect each of the probes to the pinouts on figure 5.57. Plug in the 11.1V battery and the logic analyzer should be able to read the periodic high time of the PWM pins.

5.4 Conclusion

After choosing the ultrasonic, IMU, altimeter, GPS, and air pressure sensors to meet system level requirements, the PIC32 and Raspberry Pi were able to be chosen to account for those sensors. Then, the sensors were tested with the PIC32, and the ultrasonic, IMU, and altimeter were verified to be within their requirements. The GPS, while functioning, was not as precise as it should have been, and the air pressure sensor was not implemented and tested.

PCB design for V2.0 includes design considerations for a dense IC board from Henry Ott's textbook "Electromagnetic Compatibility Engineering". 4 layer design allows return paths for high frequency components such as clocks, I2C signal bus, UART, and SPI by having the ground plane underneath the signal trace. Locations for power and signal components were considered in V2.0 PCB to reduce EMI between the microcontrollers, microprocessors and the sensor array from the voltage regulators.

Incremental verification methods are introduced but not executed. Verification is done with power rails first to ensure the rest of the system can be verified since sensors and microcontrollers must have power delivered to them for verification methods to be executed.

Chapter 6: System Behavioral Design

Chapter 6 applies the physical system developed over the previous chapters to develop drone behavior from Startup Procedures to Remote and Autonomous Control to Landing Procedures. To begin the chapter, the system state machine is introduced covering the high level layout of the system, and the detailed operations within those high level function blocks are developed. Closed-Loop control and a few auxiliary functions were tested and verified in MATLAB, but were not verified in VREP. Also, autonomous control was not completed.

6.1 Discussion of Drone Operating Environments

The drone must be able to perform in the operating environments defined by the client, Primarily STR 2.0.0, Minimal Drone Speed, the Drone shall be able to fly at least 5mph in winds up to 15mph. Assuming standard temperature pressure for design, will need to identify limitations to this assumption by analyzing the change of buoyancy, but this should only matter if loss of buoyancy is beyond motor capabilities. Disturbances can be caused by outside forces such as drag that can destabilize critical system states, and these must remain stable for the drone to be controllable.

6.2 Day in the Life of The Barone

In order to satisfy STRs 3.0.0, Remote Control and 4.0.0, Autonomous Control, a system state machine had to be developed that would be capable of performing multiple tests before and during flight, switching between different modes of drone control, and providing user feedback. To do this, a system flowchart was created that would show the drones' actions from a high-level perspective:

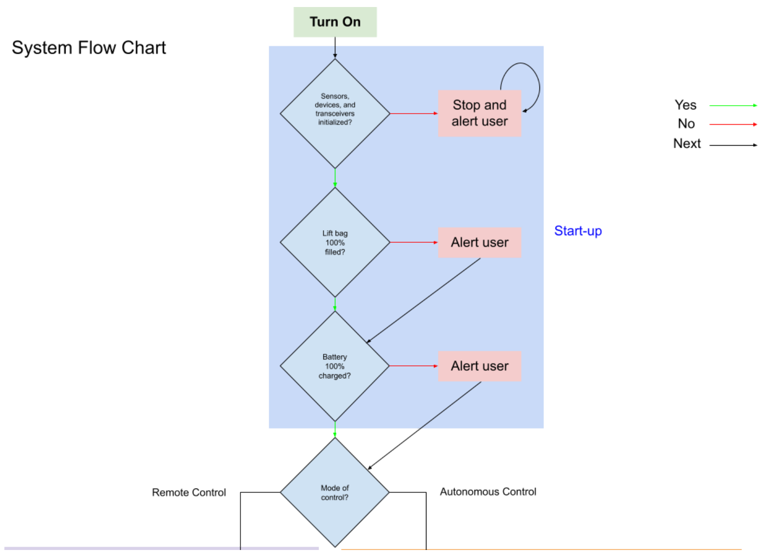


Fig. 6.1. System Flow Chart Part 1-preFlight Functions

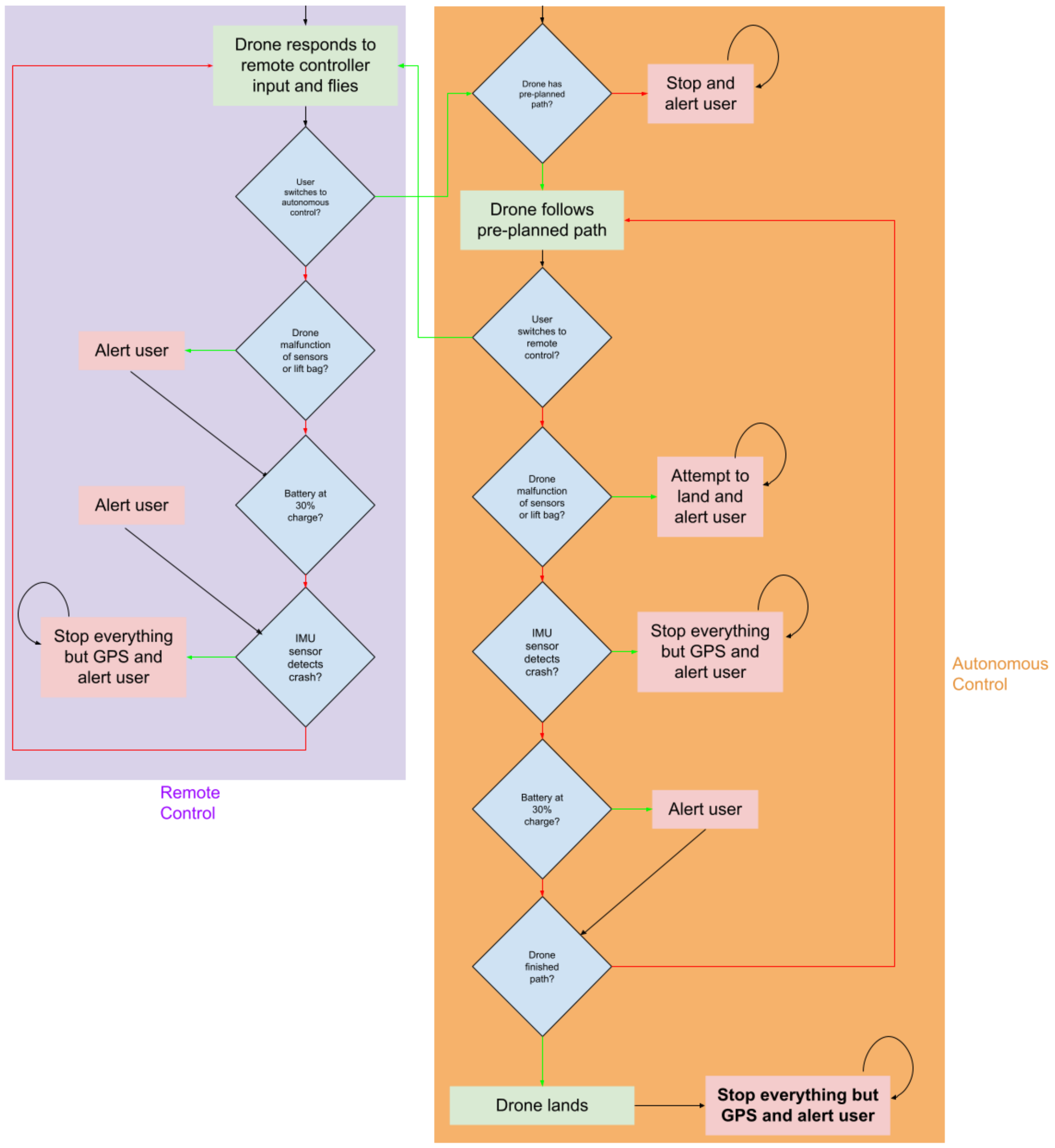

Drone responds to remote controller input and flies
User switches to autonomous control?
Drone malfunction of sensors or lift bag?
Alert user
Battery at 30% charge?
Alert user
IMU sensor detects crash?
Stop everything but GPS and alert user
Remote Control
Drone has pre-planned path?
Stop and alert user
Drone follows pre-planned path
User switches to remote control?
Drone malfunction of sensors or lift bag?
Attempt to land and alert user
IMU sensor detects crash?
Stop everything but GPS and alert user
Battery at 30% charge?
Alert user
Drone finished path?
Drone lands
Stop everything but GPS and alert user
Autonomous Control

Fig. 6.2. System Flow Chart Part 2-Flight Functions

Starting from Fig. 6.1, when the drone is first turned on, it enters the start-up procedure, where it first checks if the sensors, devices, and transceivers were initialized properly. If any of them weren't, the drone stops what it is doing and continuously alerts the user that something is wrong. If they were initialized properly, the lift bag is then checked for 100% capacity, and the drone alerts the user if it isn't but keeps going to check that the battery is 100% checked, where it does the same thing and advances towards checking which mode of control it is in. If the mode of control is remote control, then the drone enters the remote control procedure where it responds to remote controller input and flies. Looking at Fig. 6.2, the drone then checks if the user switched to autonomous control, and if they did, the drone moves to the autonomous control procedure, but if they didn't, the drone then checks if any sensors malfunctioned or if the lift bag was punctured. If it was, the drone alerts the user and keeps going towards checking if the battery is at 30% charge, and does the same thing if it is. Finally, the drone checks if the IMU sensor detected a crash, and if it did, the drone stops what it's doing and continuously alerts the user; otherwise it returns back to responding to the remote controller and flying.

Meanwhile, if the mode of control was chosen to be autonomous control, the drone first checks if it has a pre-planned path as seen in Fig. 6.2. If it doesn't, it stops everything and continuously alerts the user, but if it does, the drone follows the pre-planned path. The drone then checks if the user switched to remote control, and if they did, the drone switches to the remote control procedure, otherwise it goes on to check if any sensors malfunctioned or if the lift bag was punctured. If it was, the drone attempts to land by itself, and continuously alerts the user, but if it doesn't, it checks if the IMU sensor detected a crash, to which it will stop everything but GPS and continuously alert the user if that was the case. If not, the drone checks if the battery is at 30% charge as seen in Fig. 6.2, and alerts the user if it is, and then checks if the drone completed the pre-planned path. If it doesn't, the drone continues on its pre-planned path, and if it does, the drone lands and stops everything but GPS and continuously alerts the user.

6.3 Detailed State Machine Design

Once the system flowchart was designed, it was easier to create a lower-level state machine that would essentially perform the same actions as the flowchart but in greater detail:

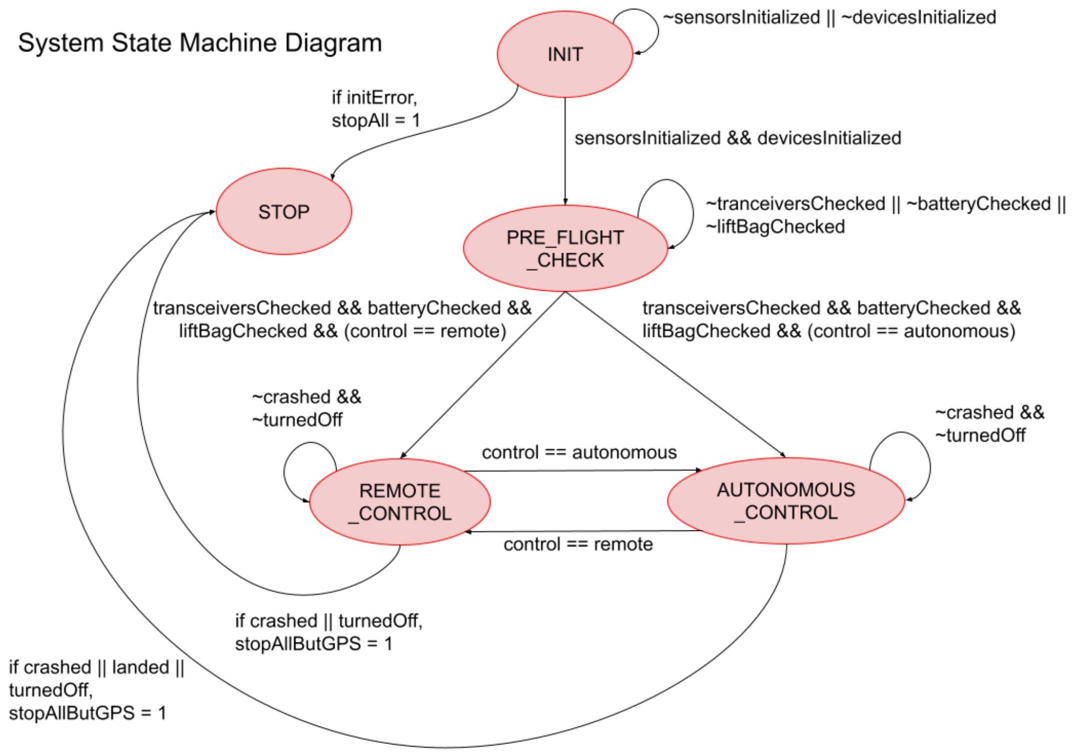

System State Machine Diagram
INIT
~sensorsInitialized || ~devicesInitialized
if initError,
stopAll = 1
sensorsInitialized && devicesInitialized
STOP
PRE_FLIGHT
_CHECK
~tranceiversChecked || ~batteryChecked ||
~liftBagChecked
transceiversChecked && batteryChecked &&
liftBagChecked && (control == remote)
transceiversChecked && batteryChecked &&
liftBagChecked && (control == autonomous)
~crashed &&
~turnedOff
~crashed &&
~turnedOff
control == autonomous
REMOTE
_CONTROL
AUTONOMOUS
_CONTROL
control == remote
if crashed || turnedOff,
stopAllButGPS = 1
if crashed || landed ||
turnedOff,
stopAllButGPS = 1

Fig. 6.3. System State Machine

On start-up, the drone enters the INIT state, and remains there until all sensors and devices are initialized. If there is an error in initialization, the drone enters the STOP state and remains there and alerts the user, but if there isn't the drone enters the PRE_FLIGHT_CHECK state where it remains until all transceivers, the battery, and the lift bag are checked. Once this is true, if the mode of control was remote control, then the drone switches to the REMOTE_CONTROL state, where it remains until it crashes, is turned off, or switched to autonomous control. If the drone crashes or is turned off, it stops all but GPS and enters the STOP state and alerts the user, and if the drone is switched to autonomous control, it enters the AUTONOMOUS_CONTROL state.

Meanwhile, if the mode of control was autonomous control, then the drone switches to the AUTONOMOUS_CONTROL state, where it remains until it crashes, lands, is turned off, or switched to remote control. If the drone crashes, lands, or is turned off, it enters the STOP state and stops all but GPS and alerts the user, and if the drone is switched to remote control, it enters the REMOTE_CONTROL state.

This state machine is only theoretical and has never been written in code and tested, so the system state machine has not contributed anything to meeting STR 3.0.0, RC Control, and 4.0.0, Autonomous Control. If the state machine were to be implemented and tested on the drone during flight, it would have made some progress towards meeting the requirements.

6.4 Startup Procedures

On start-up, the drone performs several tests to ensure it can properly fly, and either prevents the drone from flying or warns them so that they may make their own decision to progress. The first thing the drone checks is if the sensors, devices, and transceivers are initialized correctly. This includes making sure that the ultrasonic, IMU, altimeter, GPS, and air pressure sensors are detected and enabled by the PIC32, the Raspberry Pi and PIC32 establish a connection, and the remote controller receiver and data telemetry transmitter are ready to be ready from/transmit data; if any of the sensors or devices fail to initialize properly, the start-up procedure is stopped by the drone. Once this is completed successfully, the level of helium in the lift bag is checked by the air pressure sensor, and alerts the user to make their own decision whether the flight should continue. Finally, the battery is checked by means of the battery sensor attached to the battery, and lets the user know if it isn't 100% charged.

6.5 Linearization of Equations of Motion

A control method is needed to develop the predictable and stable response of the drone behavior. The creation of a nonlinear control system is complicated and beyond the scope of the team's current abilities. Nonlinear control systems are also more computationally complex, and the delayed control system response can affect the stability of the drone. For design and computational simplicity, a linear control system design is chosen. This section will reference section 2.whatever from this report, and several figures and equations will be repeated here for easy reference, but section 2.whatever will cover their original derivations.

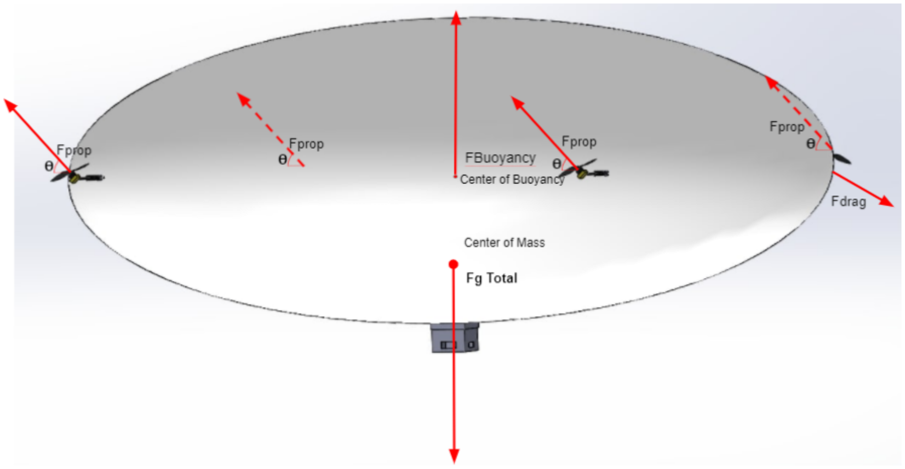


Fig. 2.1. Force Diagram

6.5.1 Translational Movement

The vectorized net force of the drone's control mechanisms, is given by equation equation 2.7

$$F_{Prop} = \sum_{x=1}^{4} M_x P * T_x (2.7)$$

Where $M_x P$ is the vector describing the direction of the propeller force (affected by servo position), and T_x is the thrust provided. To expand the equation slightly, the vector components are written in equation 6.1.

$$\begin{bmatrix} F_x \\ F_Y \\ F_Z \end{bmatrix} = \sum_{motors} \begin{bmatrix} M_{x_x} P \\ M_{x_y} P \\ M_{x_z} P \end{bmatrix} \cdot T_x \qquad (6.1)$$

Where the second subscript denotes the force direction, but can be simplified to equation 6.2.

$$\begin{bmatrix} F_x \\ F_Y \\ F_Z \end{bmatrix} = \sum_{motors} \begin{bmatrix} F_{x_x} \\ 0 \\ F_{x_z} \end{bmatrix} \qquad (6.2)$$

Where the new force components are the result of $M_{x_x} P * T_x$. and $F_{x_y} = 0$since $M_{x_y} P = 0$as designed in the propulsion system. This equation is linear and simple to implement in a linear control system directly calculating the force responses needed. Limitations of this method are discussed in subsection 6.5.3.

Applying Newton's second law, the acceleration of the system from internal forces is calculated in equation 6.3.

$$\begin{bmatrix} A_x \\ A_Y \\ A_Z \end{bmatrix} = \frac{1}{m} \cdot \sum_{motors} \begin{bmatrix} F_{x_x} \\ 0 \\ F_{x_z} \end{bmatrix} \quad (6.3)$$

6.5.2 Angular Movement

Angular approximations are more complicated to set up since positions of the center of mass, center of buoyancy, and propellers need to be known in addition to their forces. Equation 2.12 summarizes the behavior of the moments affected by the propulsion system.

$$M_{mx} = \sum_{x=1}^{4} F_{mx} * (CM_x + M_x P_x * d) \qquad (2.12)$$

To begin, the $M_x P_x * d$ term is a vector that represents the propeller position relative to the motor to determine the accurate propeller position, since the propeller itself is the source of the force. However, due to the drone's large size, $M_x P_x * d$ is small compared to CM_x since the center of mass to the motor position is approximately 55 inches whereas the length d is only a couple of inches, and 2.12 can be approximated as equation 6.4.

$$M_{mx} = \sum_{x=1}^{4} F_{mx} * (CM_x) \qquad (6.4)$$

Once again, expanding the equations out to their vector components will help identify and isolate the individual components of the system, as shown in equation 6.5.

$$\begin{bmatrix} M_{m_x} \\ M_{m_y} \\ M_{m_z} \end{bmatrix} = \sum_{motors} \begin{bmatrix} F_{x_x} \\ 0 \\ F_{x_z} \end{bmatrix} \cdot CM_x \qquad (6.5)$$

The moment caused by buoyancy is given by the equation 2.13

$$M_{buoy} = F_B * (R * CB) \qquad (2.13)$$

Since we are defining the center of the system as the center of mass, the center of mass contributes no moment and simplifies the math a bit. The R term is the rotation matrix and dependent on the angular position of the drone, and has very nonlinear dependencies involving sines, cosines, multiple terms products of each other, and simply is too nonlinear to incorporate.
This is a complicated relationship so several approximations will be made to ensure the system is linear. The primary simplification that is used is the small angle approximation, where the sin of an angle in radians is approximately the angle and the cos of the angle is 1, and the approximation can be applied

since the drone design does not require any tilt to maneuver. The buoyant moment can be simplified and expressed with equation 6.6.

$$\begin{bmatrix} M_{Roll} \\ M_{Pitch} \\ M_{Yaw} \end{bmatrix} = \begin{bmatrix} \phi \\ \theta \\ 0 \end{bmatrix} \cdot F_B \tag{6.6}$$

The moments caused by buoyancy are now given directly by the angular positions of the drone and are linear and can be combined with the moments caused by the motors in equation 6.7.

$$\begin{bmatrix} M_{Roll} \\ M_{Pitch} \\ M_{Yaw} \end{bmatrix} = \begin{bmatrix} \phi \\ \theta \\ 0 \end{bmatrix} \cdot F_B + \sum_{motors} \begin{bmatrix} F_{x_x} \\ 0 \\ F_{x_z} \end{bmatrix} \cdot CM_x \tag{6.7}$$

To find the angular acceleration the drone moments must be multiplied by the inertia matrix for the drone about the center of mass, given in equation 6.8.

$$I = \begin{bmatrix} 1.56 & -.01 & -.05 \\ -.01 & 1.88 & -.001 \\ -.05 & -.001 & 2.75 \end{bmatrix} \tag{6.8}$$

Since the diagonal values are much larger than the other values, *I* can be approximated as a diagonal matrix, which will also simplify the inverse of the matrix to be 6.9.

$$I^{-1} = \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \tag{6.9}$$

The angular accelerations are now given in a linearized form in equation 6.10.

$$\begin{bmatrix} \alpha_{Roll} \\ \alpha_{Pitch} \\ \alpha_{Yaw} \end{bmatrix} = I^{-1} \cdot \left(\begin{bmatrix} \phi \\ \theta \\ 0 \end{bmatrix} \cdot F_B + \sum \begin{bmatrix} F_{x_x} \\ 0 \\ F_{x_z} \end{bmatrix} \cdot CM_x \right) \tag{6.10}$$

The equations of motions are all linearized and ready to be incorporated into a linear control system, however, the assumptions do face limitations that will be discussed in the next section.

6.5.3 Assumptions and Limitations

To linearize the equations of motion to be used in a linear control system, a few assumptions were made.

First, the system uses the small angle approximation. The approximation is only accurate within 0.2 radians of zero. Since our propulsion system and buoyant moment mean that we do not need to tilt to maneuver, the limitation is manageable. To use the approximation, the drone must be kept as close to zero as possible, which is also required to have minimal drag on the drone anyways. The target for tilt of the drone will be within 0.1 radians to provide extra room for the system to operate in case something does go wrong. Additionally, a 2-DOF control system should be implemented to restabilize the pitch and roll angles of the drone if the drone starts to approach the limitations of the approximation, and, since the control system estimations lose accuracy and the drag will have a greater impact on the drone leading to system instability, all other functions should be abandoned to restabilize the pitch and roll angles of the drone.

Another limitation is that we treat Fx and Fz components of the motor forces separate during calculations. On the drone, the Fx and Fz components are linked since there is a single force and the direction is controlled by the servo angle, however, this is nonlinear. The Fx and Fz components can be decoupled, as was done in the system linearization, however, the limitations of the forces need to be passed through a function in order to convert the values back to throttle and angle values for the motors and servos.

Approximations were made to help ensure the system remained manageable and linear. This will result in inaccuracy of the system response, so an integral path needs to be added to the control system to reject the error. The integral path also needs to be present to reject the disturbance of wind and other outside factors, but that's expected. To work around the errors associated with the approximations, the system needs to be tested in detailed simulation, VREP, at ideal conditions, i.e. no wind. The integral values can be recorded and the integral paths can be initialized with the recorded values to prevent error at the start of the flight path and prevent overshoot, since the system will not need time to build up the integral values. Then the adjustments of the integral path can respond directly to wind, with the steady state bias already incorporated.

6.6 Open Loop Remote Control Design

6.6.1 Design of Remote Control Response

STR 3.0.0, Remote Control, requires a remote control implementation of the drone that is capable of responding to user input while maintaining stable pitch and rolls, <-0.1 radians from zero. The drone differs from other drones since it does not need to tilt, so the Remote Commands need to be interpreted. To prevent the need to design a custom remote controller, a standard controller is used, but needs to have inputs modified to fall in line with the needs of the drone.

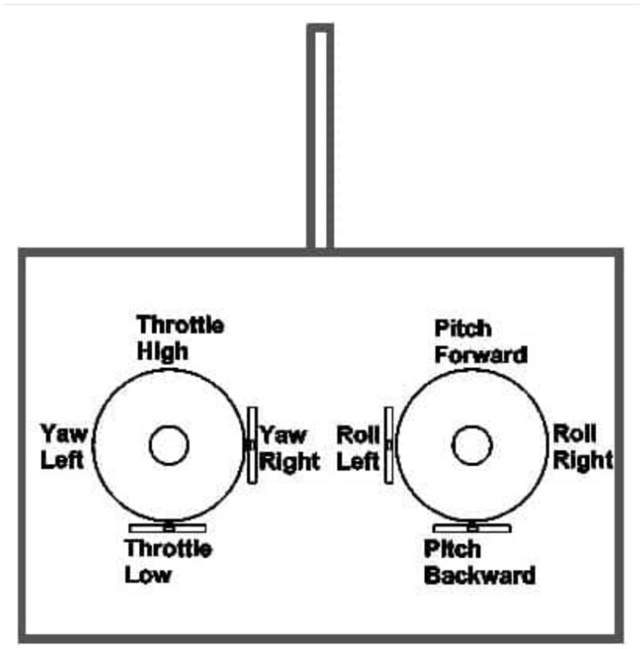


Fig. 6.4. Typical RC Controller Layout

Fig. 6.4 provides the basic layout of most controllers. Pitch forward and backward translate to forward and backward movement. Yaw left and right translate to turn left and right movement. Roll Left and right translate to moving left and right. The throttle translates to power given to the propellers. Since standard drones move by tilting, the throttle just needs to scale all motor commands, and adjusting the pitch and roll angles can translationally move. Also, throttle is how the drone will ascend, since high throttle will move the drone up, and low throttle lets gravity pull the drone down.

The Barone operates differently than other drones so several inputs need to be renamed. Instead of pitch, the command is changed to forward, since we do not change pitch. Roll inputs are not used, since the Barone is not capable of moving sideways with the propulsion setup and the need to not change roll. Yaw is just changed to turn, the drone can change yaw but since pitch is changed to forward turn is just chosen to prevent jumping between greek letters and command names. Throttle high and low is a slightly more complicated setup. Gravity can not be relied on to pull down the Barone. Switching between a negative one value and positive one value on the throttle control to control upwards and downwards movement is confusing when it will always be treated as positive for turn and command inputs, so it is not a great solution. Instead, the throttle will control the throttle, and a button needs to be added to change between ascend and descend. The RC commands become turn, forward, throttle, and a binary input for ascending and descending.

Next the values need to be scaled to servo angles to control forward, turn, and height control to the drone. The drone needs to be able to go full forward, and full turn, but should also have angles scaled if both inputs are maxed to ensure both commands and desired magnitudes are implemented. The behavior is set up in equation 6.11.

$$\alpha = pi/2 * (Forward \pm Turn)/(1 + |Forward * Turn|) \qquad (6.11)$$

The forward and turn commands are combined, positive turn on the right side and negative on the left, to get a combination of inputs. Next, to scale the values to always be less than one, the denominator considers the product of the two terms and adds them to 1 in order to scale the full turn. If either forward or turn are zero, the denominator is one and the servo angles will give max commands. If both are maxed, the denominator becomes 2, as well as the numerator, resulting in a net value of 1 on one side, within the max. The equation also allows for values in between and scales the servo angles accordingly. This also allows for the servos to still allow propellers in the up and down directions so the drone can move up and down. The pi/2 term is just for scaling of the system.

In the max command example given previously, one side is given a full forward command, but the opposite side has a full up command, resulting in an unintentional moment as shown in Fig. 6.5.

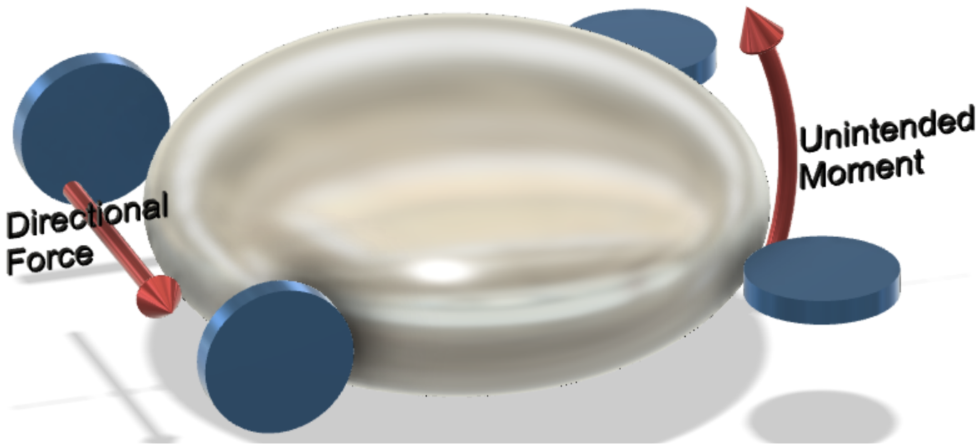


Fig. 6.5. Unintentional Moment With RC Commands

The unintentional moment affects system stability, so the throttle needs to be adjusted. The up and down forces need to remain equal on both sides, so the z components of the thrust values need to equal. The z components of the angles are isolated using the cosine function and the inner value of the turn, the side that would provide the excess force in the z-direction, and that is multiplied by the ratio of the z-component to scale, and is modeled in equation 6.12.

$$T_{inner} = T_{input} * cos(\alpha_{outer})/cos(\alpha_{inner}) \qquad (6.12)$$

6.6.2 Testing of Open-Loop Control Design and Instability

The open-loop control response was simulated in VREP due to the complexity of the system response. Due to the setup, the full simulation is given in Chapter 9, but in short the drone was unstable during flight.

The drone's propulsion system is mounted above the center of mass, so when the propellers apply a force, there is a pitch moment created as illustrated in Fig 6.6.

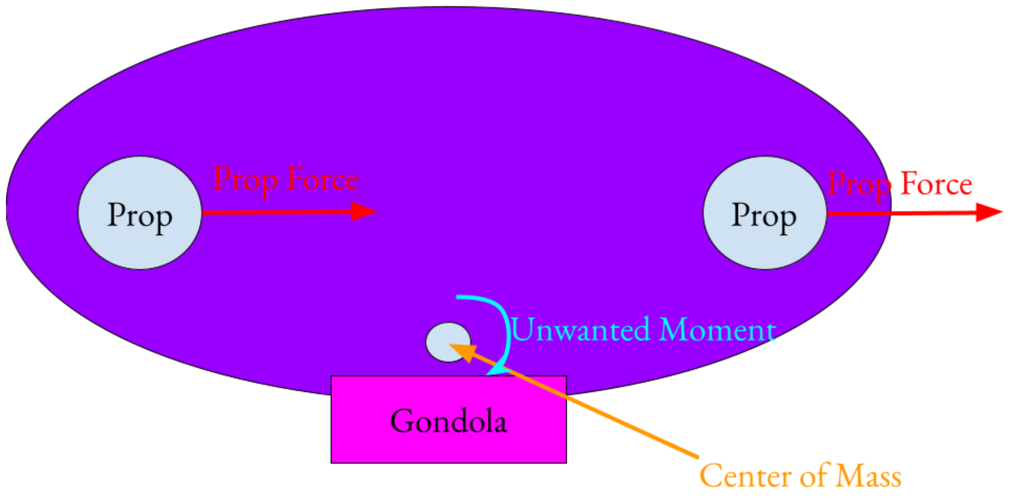


Fig. 6.6. Pitch Moment Caused by Propeller Force

In addition to the effect of the unwanted pitch moment, any other changes on the pitch and roll affect the system, and since the open loop controls don't take the angles into account, the system stability will remain and potentially amplify. The system cannot operate on an open-loop RC system alone, and a closed-loop RC system is needed in order to maintain the system stability.

6.7 Auxiliary Functions

STR 3.1.3, Autonomous Functions, requires multiple functions to aid with Remote Control, including large tilt angle handling, hovering, auto landing and auto take-off. The Auxiliary Functions will be covered before the Closed- Loop RC for a few reasons. Although the auxiliary functions themselves are less important to be designed than the closed loop RC, they provide the basis for the Closed-Loop RC, and the simpler designs of the auxiliary functions enable the system response to be developed in complexity over time, with testing in between functions. To be more specific, the Large Angle Error response only maintains the pitch and roll angles of the drone to be zero and is the simplest closed loop control system on the drone. The Large Angle Error can then have height added to create a hovering system with stable angles. After height and angles are controlled, the auto-takeoff and landing can be implemented by feeding a ramp input to the hovering function, and tests the system against input response. The approach enables all the components of the closed-loop RC to be built and tested incrementally before variable user input is tested, which will be the hardest test the controls will encounter due to the uncertainty of user inputted commands as well as external forces such as drag.

6.7.1 Large Angle Error Response

The large angle error is the simplest and most important control system for the Barone. Pitch and roll stability is pivotal to drone stability, and every control system built for the drone will prioritize pitch and roll above other states. One of the reasons for the importance of maintaining pitch and roll is given in Fig 6.7.

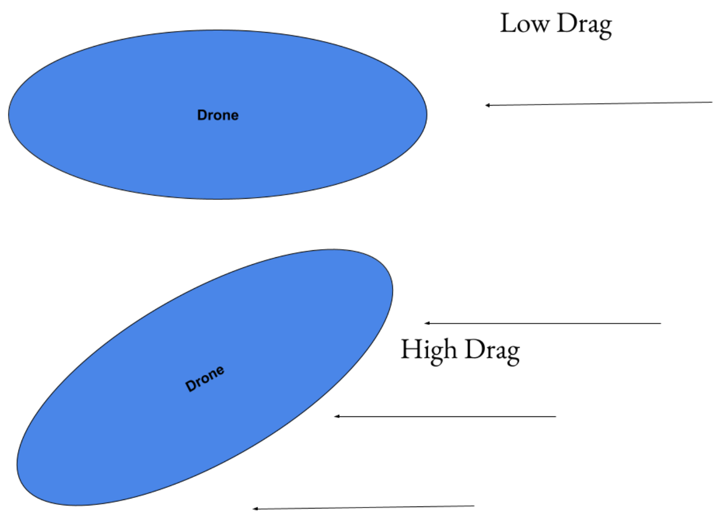


Fig. 6.7. Effects on Drag on Large Tilt Angles

The Barone has an ellipsoid shape, and when the drone's tilt is near zero, has very low drag. As the tilt angle increases, the cross sectional area of the lift bag in relation to the air velocity increases, drastically increasing the drag. In addition to the small angle approximation error that arises at the tilt angle increases, the drag will increase dramatically, and can result in the loss of controllability of the drone and crash--this is probably the worst case scenario for the control system, and it is why the pitch and roll angles are prioritized. As mentioned in the system linearization, the force components of the propellers are decoupled, so the input and state vectors are defined as

$$x = \begin{bmatrix} roll \\ droll \\ pitch \\ dpitch \end{bmatrix}, \ u = \begin{bmatrix} T_{FR_x} \\ T_{FL_x} \\ T_{BR_x} \\ T_{BL_x} \\ T_{FR_z} \\ T_{FL_z} \\ T_{BR_z} \\ T_{BL_z} \end{bmatrix} \quad (6.13)$$

Where x is the state vector, u is the input vector, T is thrust of the propeller, and FR stands for front right, FL stands for front left, BR stands for back right, and BL stands for back left, and the x and z stand for the x and z components of the force they represent. Due to the number of inputs and outputs, as well as the dependencies that exist between each other, a full state feedback system is implemented. The A and B matrices are determined using equation 6.10. The C matrix is given by [1 0 1 0] since the drone only needs stable pitch and roll and their derivatives can vary.

6.7.2 Pitch Roll Height Regulator

To add height to the regulator developed in section 6.7.1, the states only needed to be expanded to equation 6.14. STR 3.0.0 Autonomous Control had a height control added to state that the height must be within 0.15 meters of 1 meter off the ground.

$$x = \begin{bmatrix} roll \\ droll \\ pitch \\ dpitch \\ height \\ dheight \end{bmatrix}, \; u = \begin{bmatrix} T_{FR_x} \\ T_{FL_x} \\ T_{BR_x} \\ T_{BL_x} \\ T_{FR_z} \\ T_{FL_z} \\ T_{BR_z} \\ T_{BL_z} \end{bmatrix} \quad (6.14)$$

And the A and B matrices are expanded to incorporate changes in height using equation 6.3. An integral path is needed to help reject disturbance, so it is added to the system, as shown in Fig. 6.8.

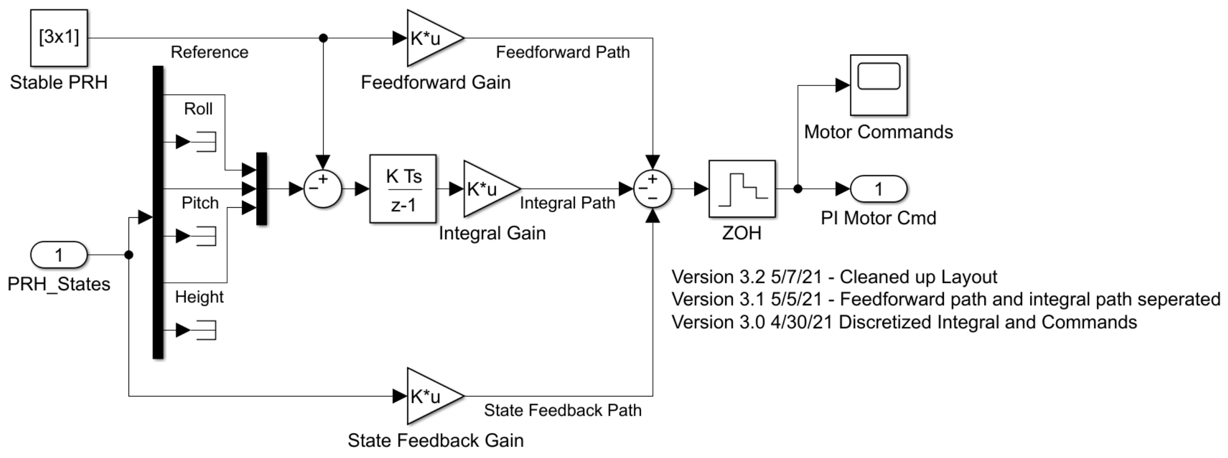


Fig. 6.8. PRH_PI Regulator

Since the regulator controls pitch roll and height, and has proportional and integral gains, it is referred to as the PRH_PI regulator. As shown in Fig 6.8, full state feedback, feedforward gain, and integral gain are all used to control the system response. Pole placement was used to place the poles originally, and somewhat arbitrarily, with the pitch and roll states having the poles farthest from zero since their response speed is more important than conserving energy. The feedforward gains were determined by inverting the DC gains of the state feedback path to scale the system response back to a DC gain of 1. The integral path was adjusted manually to be 1/10th of the feedback gain, and this can be further adjusted with simulation in VREP. The responses of the system at undesirable starting states are given in Fig. 6.9.

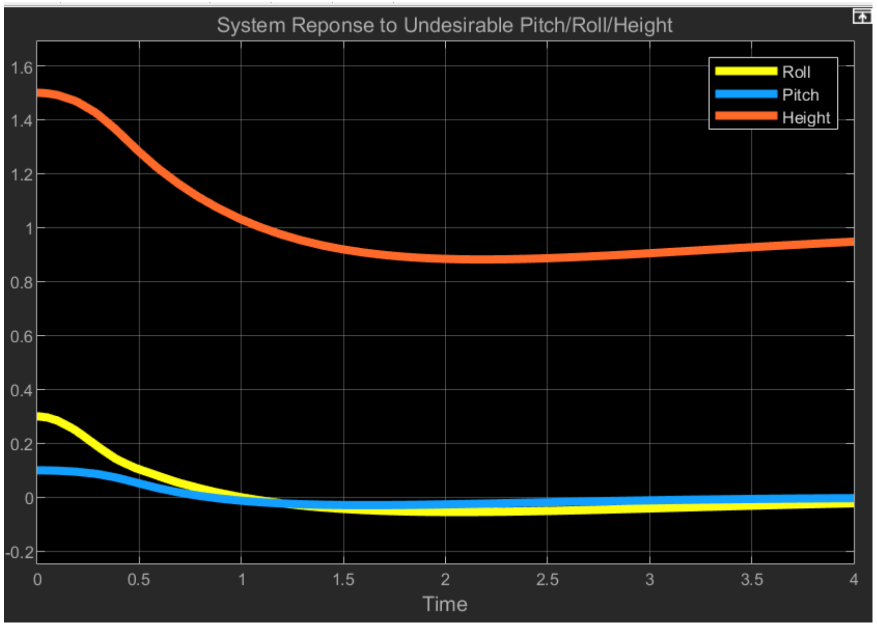


Fig. 6.9. PRH_PI System Response

The system starts 0.5 meters of the desired height, 0.3 radians off the desired roll, and 0.1 radians off the desired pitch. These are all very large errors, since height should be kept within 0.15 m of 1 m and pitch and roll should be within 0.1 radians of zero, which is why they were chosen to test the system response. The height dips to 0.84 in the tests, outside of the desired range of the system, but this is with a large starting height error and due to resources being given priority to the correcting the large pitch and roll errors. The pitch and roll of the system respond quickly and return to zero in less than a second, and stay well within the 0.1 radian error required. The system can be further improved, but tuning should be used with the help of the VREP responses to the increased accuracy of the system responses.

6.7.3 Auto Take-Off and Landing

The auto take-off function is helpful in getting the drone to s safe height before flying, and the auto-landing function can help with a smooth landing. For these functions, the drone only needs to keep the tilt stable and adjust height, so the PRH_PI regulator can be used as a base, and the height value command varies. Since there is a change of a meter in the height, integrator windup is a problem, so instead of a step input, a ramp input can be fed for a smoother transition and prevent overshoot. Since it only manipulates one value, the rest of the setup remains the same. The ramp function was given a slope of 0.25m/s and maxed at 1 m, and the height response is given in Fig 6.10.

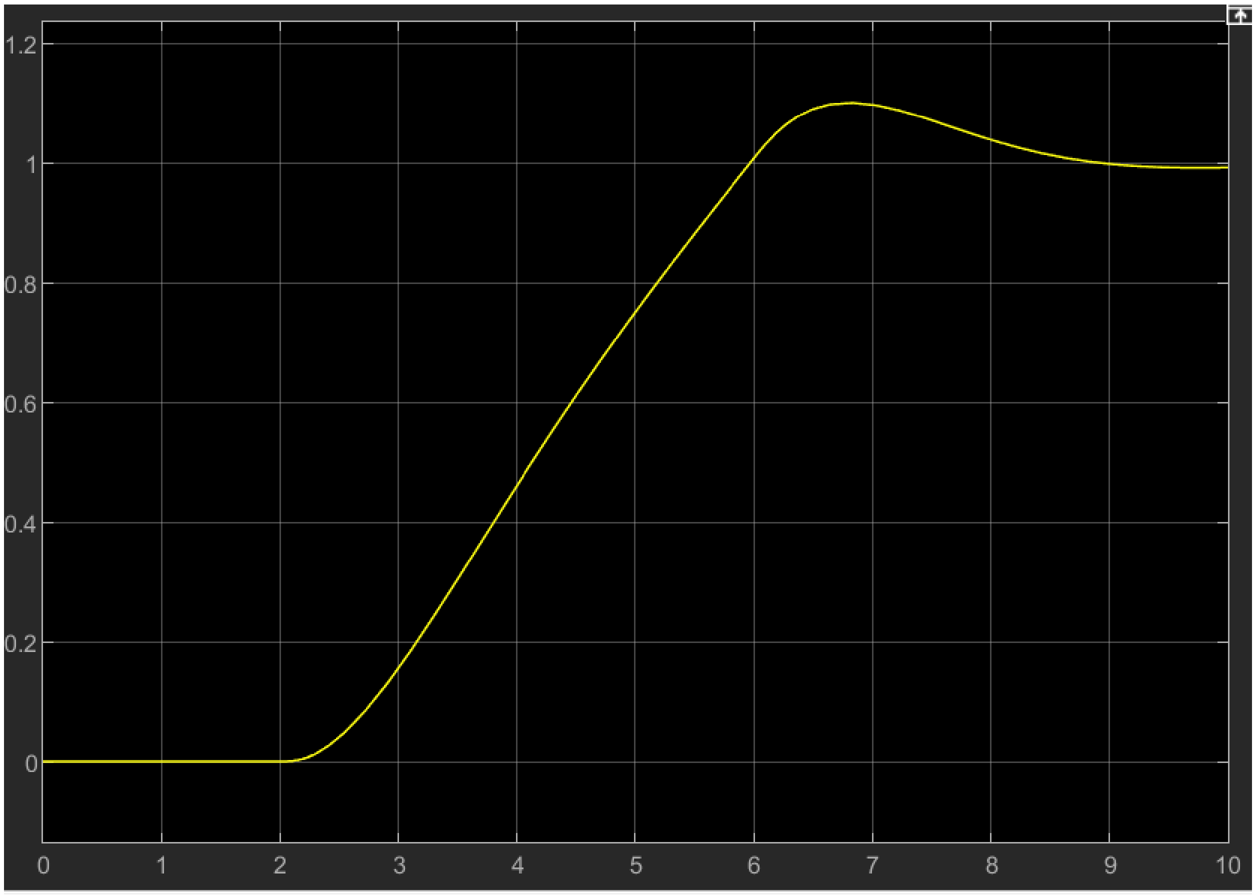


Fig 6.10 Height Response in Auto Take-Off Function

The take off function was delayed slightly for testing purposes, but clearly shows the drone is on the ground, 0 meters, and height increases to 1 meter steadily and with an overshoot of 0.1 meters, within the 0.15 meter requirement. The system response is stable and ready for testing in Vrep.

Similarly, auto landing just uses a ramp function, but goes from 1 meter to zero instead. Also, since the drone can hit something going down, the ramp is given a slighter slope to minimize impact, since overshooting is not possible. The height response is given in Fig. 6.11.

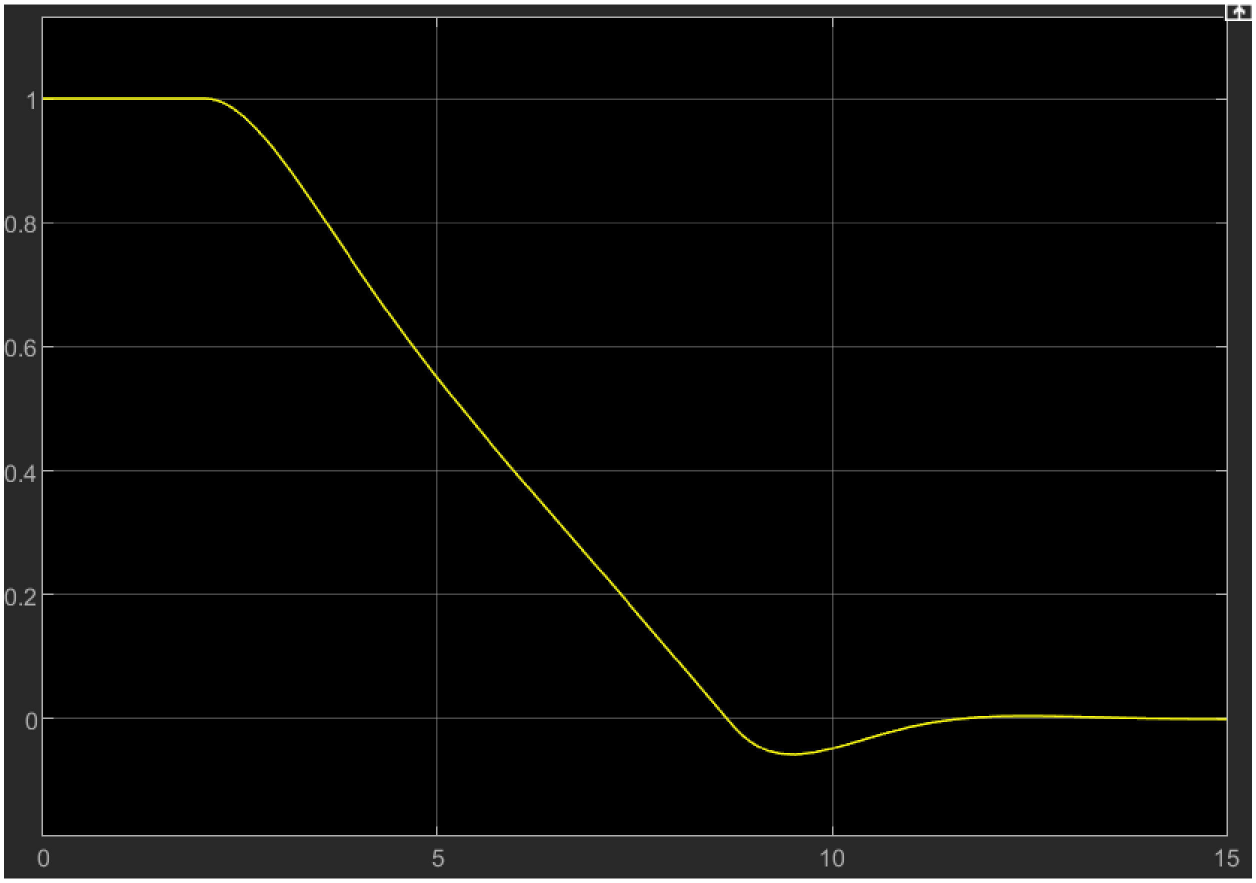


Fig. 6.11 Height Response to Auto-Landing

The drone height command is steady with minimal overshoot. In reality there would not be overshoot but impact instead. Since the velocity before impact is 0.1 m/s, with a 4.5kg drone, the kinetic energy is only 0.05 joules, and not a cause for concern. The system may be tuned more but should be done based on results from a more detailed VREP simulation.

6.8 Closed Loop Remote Control Design

6.8.1 Design Overview

STR 3.0.0, Remote Control, states The drone should have RC control implementation to allow for direct control of the drone. The drone can start in this state, or be switched to from autonomous control. STR 3.1.3, Autonomous Functions, states autonomous functions should be accessible in the remote control state, such as terrain tracking, and tilt control was added after the open loop design proved unstable.

To maintain system stability during remote control flight, a few factors need to be considered. First, the tilt angle, the pitch and roll angles, must be kept near zero for the system linearization to remain accurate, as well as ensure the drag will not increase significantly and cause the drone to lose controllability. Terrain Tracking was also a function that is required of the RC Functionality, so height control is needed as well. The user inputs also need to be interpreted in a way such that the motor can respond predictably, but should also not come at a cost to system stability. The order of priority for the system is

1) Tilt Stability
2) Height Control
3) User Response

To achieve the desired results, there's several functions that need to be incorporated into the remote control design to be effective. First, the PRH regulator developed in the auxiliary functions section can be used to maintain the stability of the identified critical states. An RC command interpreter is also required to translate user inputs. A saturator is needed to limit the commands to realizable values. Lastly, a converter is needed to turn the force values into throttle and servo angle commands. The setup is provided below in Fig. 6.12.

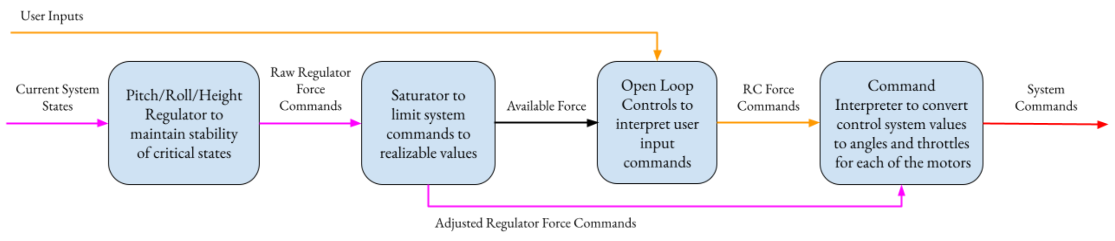

User Inputs
Current System States
Pitch/Roll/Height Regulator to maintain stability of critical states
Raw Regulator Force Commands
Saturator to limit system commands to realizable values
Available Force
Open Loop Controls to interpret user input commands
RC Force Commands
Command Interpreter to convert control system values to angles and throttles for each of the motors
System Commands
Adjusted Regulator Force Commands

Fig. 6.12. Closed-Loop RC Layout

An output is added to the saturator called available force, which tells the open loop controls block to scale the values of the force commands, ensuring that the RC commands only use “leftovers” of the PRH regulator, and the system can respond to user commands, but never at the cost of system stability. The open loop and regulator force commands are then simply added together before being converted to motor and servo commands.

6.8.2 Adjustments to the PRH Regulator

The PRH_PI regulator actually already handles the critical states of the drone on its own. The only adjustment that may be needed is to in testing the response of the system to RC commands needs to be tested since that is now an internal disturbance to the drone, something the regulator was not tested against. Also, the open loop controls will exert a moment as shown in the open loop section, and the integral path needs to ensure that it can compensate against the steady state and rapid changes in user inputs.

6.8.3 Saturator Design

A Saturator is needed to ensure values are realizable and can be scaled for proper motor commands. Since the force values are decoupled into their X and Z components, they need to be recombined and tested against the max force value. The max force value can be given as a constant of the system if there is a max value that can not be exceeded, or if a voltage detector is incorporated on the battery, can be given as an input that varies over time. If any values exceed the max value, the max force command set (X and Z component recombined) is used to scale all the force inputs. All the values are scaled together to prevent an unbalanced input from destabilizing the system. If no pair exceeds the max value, the force commands are passed on without modification. Lastly, the saturator outputs the available force after the saturator commands to help scale the open loop system. The full setup is given in Fig. 6.13.

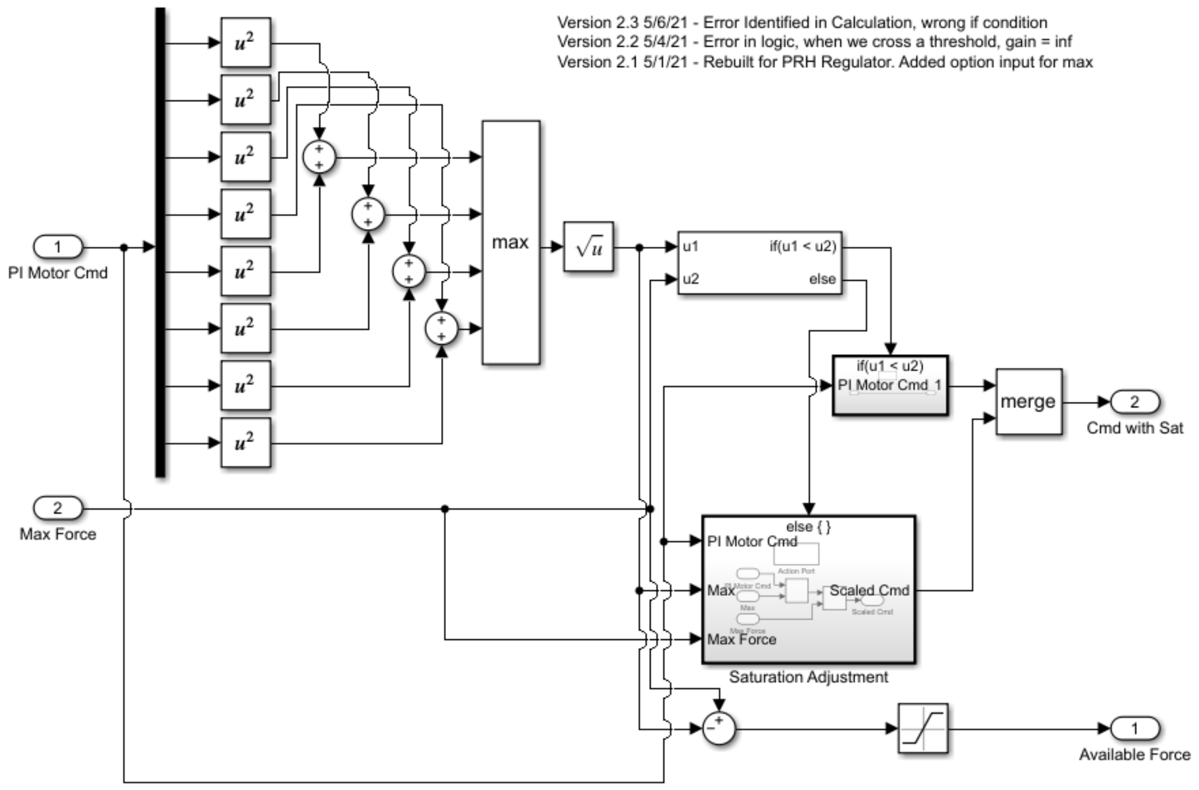

Version 2.3 5/6/21 - Error Identified in Calculation, wrong if condition
Version 2.2 5/4/21 - Error in logic, when we cross a threshold, gain = inf
Version 2.1 5/1/21 - Rebuilt for PRH Regulator. Added option input for max
1
PI Motor Cmd
2
Max Force
u^2
max
$\sqrt{u}$
u1
u2
if(u1 < u2)
else
if(u1 < u2)
PI Motor Cmd_1
merge
2
Cmd with Sat
else { }
PI Motor Cmd
Max
Max Force
Scaled Cmd
Saturation Adjustment
1
Available Force

Fig. 6.13. Saturator Design

6.8.4 Open Loop Control Design for Closed Loop Control

In the original open loop design, ascend and descend commands were needed, however, they are not needed since the PRH Regulator will control height, so the function is adjusted.

First the turn and forward commands are set to be between -1 and 1, a common standard for throttle commands. The Forward vector has all X-direction vectors pointed forward to move the drone forward, and the turn vector has the right X-direction vectors as forward and the left X-direction vectors as backwards to turn the drone. The Z-Components are all kept at zero for three reasons: first, the PRH Regulator will correct for errors in tilt caused by the moment generated by applying force in the X-direction, and second, at steady state, the drag and propulsion systems will apply equal force at the center of the lift bag due to their mounting, creating near zero net moment, and third, it is easier to scale the value to max throttles since the distance formula won't be needed like in the saturator.

Next, the commands should be combined to create a single command. To combine, first the throttles can be added, and the combination is scaled by the inputs so they're balanced. The issue now is the value can be greater than 1, not a standard throttle command. The motor values then have their max values checked, and if any value is greater than 1, all the values will be scaled so the largest value is equal to 1. All the values will be scaled the same to preserve the forward and turn ration as provided by the user.

Finally, since the max value is one, the throttle can be directly multiplied by the available force input to scale the force commands of the open loop system to the max force that can be used, and the value can be passed to the rest of the system.

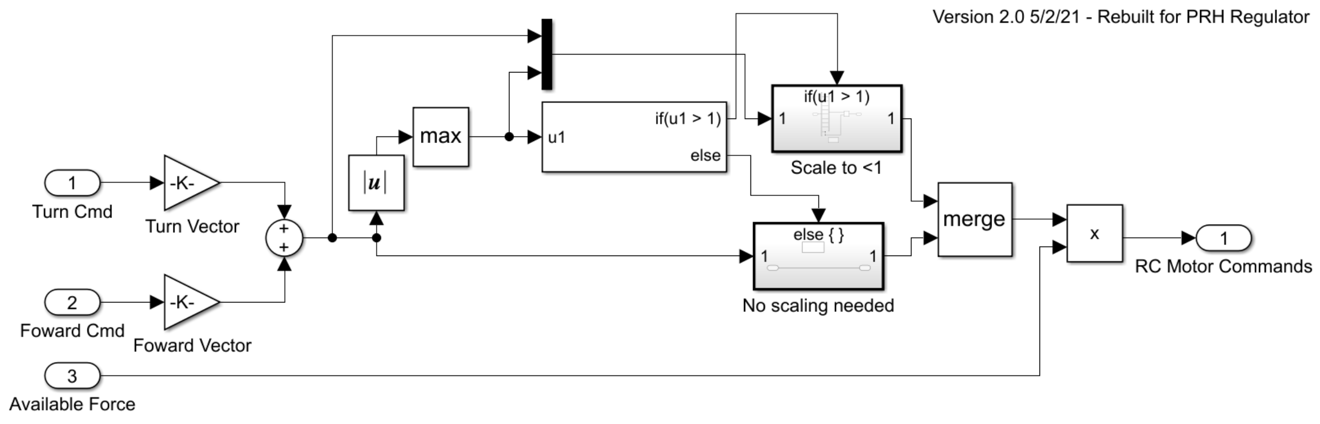


Fig. 6.14. Open-Loop RC Design

6.8.5 Force to Throttle/Angle Converter

To help linearization of the system, the components were decoupled and the servo angle was ignored. To output commands to the system, the force commands need to be converted to throttle and servo angle commands.

Since the saturator and scaled open loop commands are already within the force capabilities of the system, the vectors can be used to calculate the servo angles using the arctangent function with atan(Fx/Fy), and the throttle value determined by comparing the ratio of the max force of the system and the combined force values of the X and Z components. The only limit is the atan function has a range between -pi/2 and pi/2, so the X-component is checked to check the angle quadrant, and a value of pi can be added to shift the servo angle to the proper quadrant. Due to the size of the block, and since the math itself is simple, it is included in the appendix for reference, but does not offer much more information.

6.8.6 Closed-Loop RC Integration

All the components are finally combined to generate the full Closed-Loop RC system as shown in Fig. 6.15.

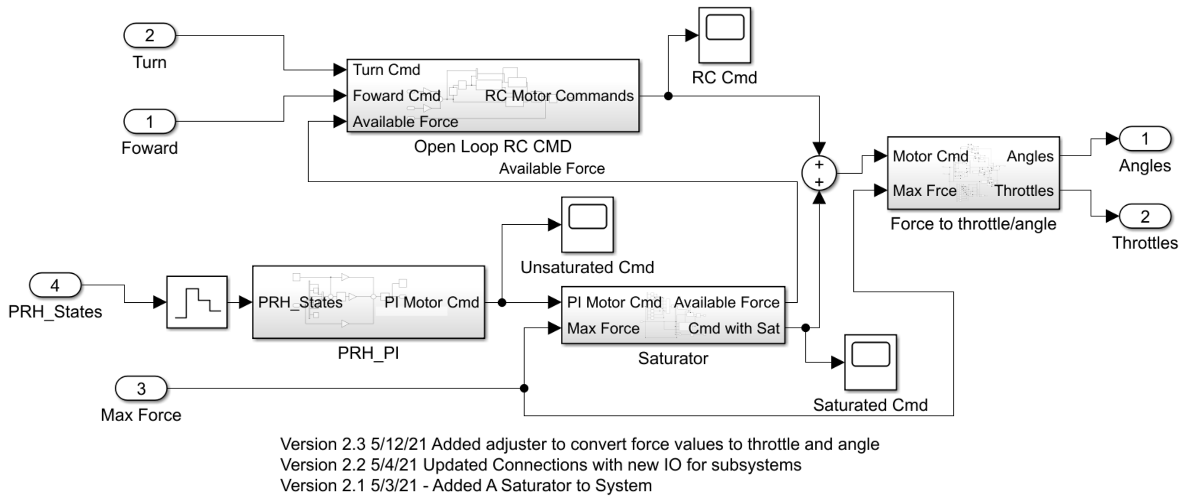


Fig. 6.15. Closed-Loop RC, Full System

The Full system follows the general setup as laid out in Fig. 6.12. The states are fed into the regulator to get the first set of force commands. The values are passed into the saturator block along with the max force value to scale the commands to realizable forces, and returns the available force to the Open-Loop RC block. The Open-Loop RC then interprets turn and forward commands into a throttle to be scaled with the available force input. Then, the regulator and open-loop RC commands are combined and pass through the Force to Throttle/Angles block to be converted into commands for the system.

6.8.7 Closed-Loop RC Verification in MATLAB

Since the open loop design failed to meet design specifications, STR 3.4.0, Closed-Loop RC was added, and requires a system that maintains pitch of roll error of <0.1 radians and a height error of less than 0.15 radians. The system was verified through matlab simulation, however, it is not fully detailed like the Vrep simulation environment. To be tested, the force to throttle/angle converter was removed and the force commands were tested with the State Space Matrices defined in section 6.7.2, Pitch/Roll/Height Regulator.

The PRH regulator and its stability was already tested in the auxiliary functions section, so the system stability response to the RC commands was what needed testing. The worst input response from users would be full stop to full forward, since the pitch moment would be greatly affected from the propulsion systems positioning onboard the drone. To test, the drone is given a forward throttle command of 1, and after 1 second, the command is immediately changed to -1, and the effects on the pitch roll and height are given in Fig. 6.16. At one second, the pitch value jerks as expected, and the pitch never exceeds 0.06 radians, well within the defined 0.1 limit. The height seems like a small fluctuation but almost imperceptible, especially when compared to the 0.15 m error allowed.

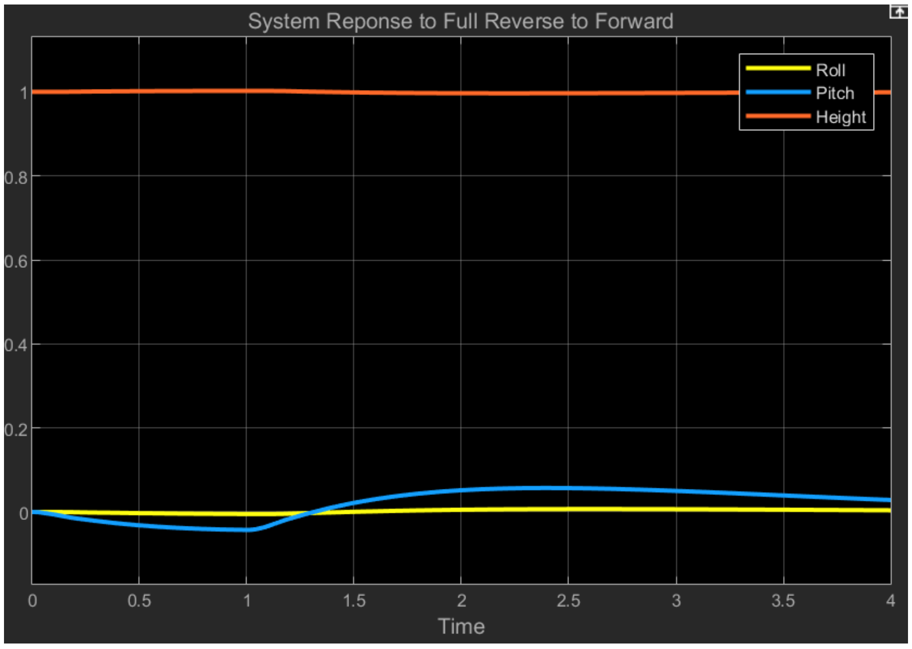


Fig. 6.16. Critical States Response to RC Input

The system maintains full stability even with the worst case scenario tested in MATLAB. The values can be tuned to have a better response with some trial and error, but will be fine tuned with using the system responses in VREP as more accurate guides to the system response.

The system should be tested and verified in VREP, especially since the physical model is more accurate than what is used in MATLAB, but the simulation communication is incomplete, so it is not verified in the environment. The simulation layout and testing plan is laid out in section 9.whatever

6.9 Autonomous Design

Autonomous design was required as stated in STR 4.0.0, Autonomous Control. The system plants were defined by adding yaw and north east components using equations 6.3 and 6.10, and the small angle approximation was used to simplify the rotation matrix, but only for pitch and roll since the yaw angle could not use small angle. The yaw angle included using gain scheduling to allow for the full 360 degree use, but did not seem like a good approach. Some work was started on instead continually updating the next waypoint to be defined in the drone frame coordinates, which would require constant updates, but gain scheduling would be vastly simplified to only a few sections instead of a full 360 degrees, but no significant progress was made. The requirement was not met.

6.10 Error Identification and Response

6.10.1 Crash Detection

The drone has an 9-DOF IMU sensor to detect the orientation of the drone, which is required by STR 4.3.1, Error Handling By the IMU Sensor. The IMU is constantly sending back information about the drone's orientation, and can detect if a sudden change in orientation occurs, meaning that the drone has begun tumbling. The IMU sensor has been implemented and verified as seen in Chapter 5, so it meets STR 4.3.1, Error Handling By the IMU Sensor.

6.10.2 Deflated Bag

Onboard the drone is also a pressure sensor to determine if the lift bag deflates. STR 4.3.3, Popped Balloon Error Case requires the drone to make an emergency landing if the lift bag rapidly loses volume. Currently, the pressure sensor is not implemented in software and has not been tested, so STR 4.3.3, Popped balloon was never verified. Ideally, the drone would be able to keep track of the pressure of the lift bag, and be able to detect a drastic change in pressure. However, with the current physical design of the drone, the propulsion modules are held in place by the lift bag inflation, and if the bag deflates, there is no way to controllable land since there is no way to know the propeller positions. The loss of rigidity of the propulsion modules is shown in the flight test section, and the drone is too uncontrollable, so the drone fails STR 4.3.3, Popped balloon.

6.11 Conclusion

The controls design was verified in matlab for STR 3.4.0, Closed-Loop RC, and STR 3.1.3, Autonomous Functions, but not functions were fully verified in VREP. The autonomous functions were not completed or tested. The error detection and automatic responses were also not completed. Based on the MATLAB verification, the drone is controllable and a full state feedback PI controller is possible to incorporate on the drone, but controls as a whole failed to meet the system requirements.

Chapter 7: Power Analysis and Flight Time

Power management and analysis are crucial to understanding the achievable flight times of the drone based on the performance of the system. Power estimates are gained through individual testing of each part that is compared to the estimated power specified on each component's datasheet. Through analyzing each component of the system and the power that will be required for each one, the flight time requirement of a greater than 30 minute normal autonomous flight time was theoretically achieved by ensuring the minimum flight time is greater than 30 minutes, fulfilling STR 1.0.0, Flight Time.

Switching and linear voltage regulators were used to achieve voltage levels required for different electrical components used. Even with power losses and replacement motors that were less power efficient than the intended motors for the same thrust, the system theoretically always results in at least a 30 minute flight time, but cannot be verified until experimental tests are done.

7.1 System Components

To properly analyze the power required of the entire system, we must look at each component individually. Separating the power costs by part allows us to create a power budget that lists the amount of power needed for every part to add together for the total power required for a flight. The total power used will largely depend on the activity of the motors and servos, which amount to greater than 90% of the total system power. Most of the parts of the drone, including the sensors and internal control mechanisms, have been tested to draw a constant amount of power when the drone is in flight to simulate the constant sensor readings, data transmitting, data receiving, and control functions which are necessary for the drone to respond to user controls and to process the data it is receiving. The following data from the main power budget is divided into three categories separated by general functions. These sections include the communication mechanisms, the sensors, and the actuators. The data telemetry kit and internal pressure sensor could not be tested. However, since these parts amount to less than 1% of the total power used by the system, the estimated power values were used instead, since it will minimally affect the flight time of the drone. The power summation between every part is stated in section 7.2.2.

7.1.1 Communication Mechanisms

The communication mechanisms include the uC32 microcontroller, Raspberry Pi 3B+ microprocessor, AKK KC03 camera transmitter, a Sparkfun serial telemetry radio kit, and an FS-iA6B RC receiver on the drone. The power requirements of each of these parts were taken from their respective datasheets. These power requirements were then tested through measuring the current through an ammeter while the parts were in operation. The tests done for the control components are seen in the following system block diagrams, Figures 7.1 and 7.2. Table 7.1 describes the estimated power required for each component mentioned based on the typical power required for each part to function while in flight, as stated in their respective datasheets, and the power that was actually tested for each part individually.

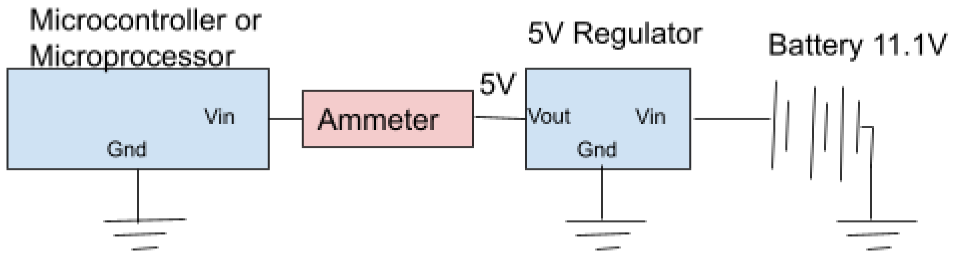


Fig. 7.1. Microcontroller and Microprocessor Power Test

Microcontroller/Microprocessor Test Process:

1. Ensure Microcontroller or Microprocessor is running the code produced for their operation
2. Connect Battery to Vin of 5V Regulator
3. Connect 5V regulator Vout to an ammeter and the other end of the ammeter to the microprocessor or the microcontroller.
4. Ensure each part has a common ground
5. Wait 5 seconds for current to stabilize and record current on the ammeter

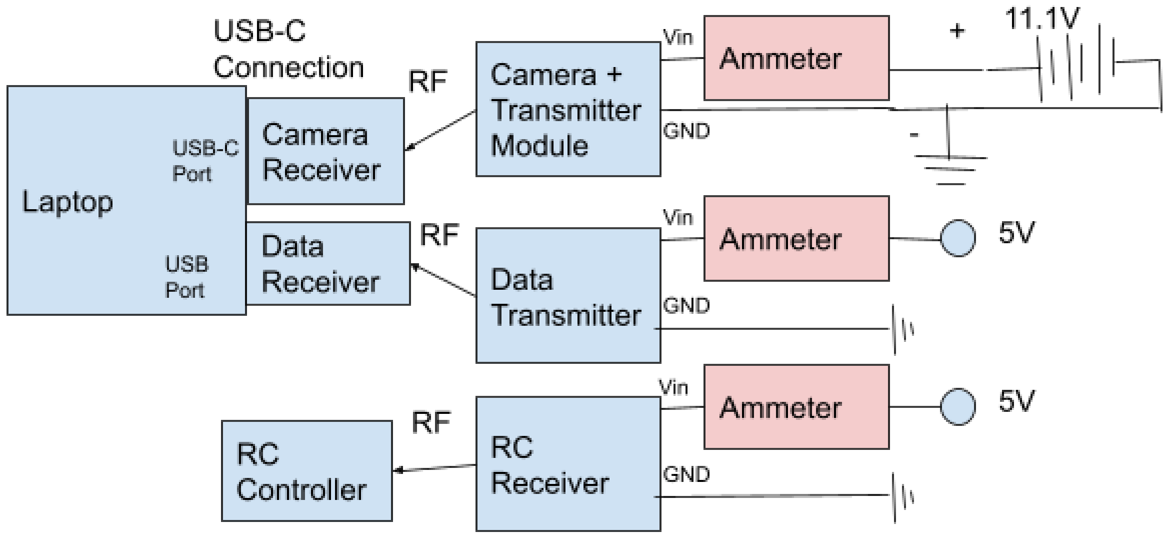


Fig. 7.2. Camera, RC, and Data Transmitter Test

Transmitters/Receiver Test Process:

1. Ensure Battery is connected to camera module with an ammeter in between and with a common ground
2. Plug in camera receiver to a USB-C port on a computer
3. Make sure camera receiver is showing output on the receiver
4. Wait 5 seconds for current to stabilize and record current on the ammeter
5. Connect any 5V source to the data transmitter and RC receiver with a common ground for both
6. Turn on the RC controller and ensure it reads the RC receiver is active
7. Record current needed for moving the joysticks on the controller
8. Connect Data receiver to a USB port on a computer
9. Ensure data is being transmitted by the data transmitter
10. Record current needed by data transmitter

Table 7.1
Estimated and Tested Power Required for Control Components in Flight

Part Name	Nominal Voltage (V)	Estimated Current (mA)	Tested Current (mA)	Estimated Power (mW)	Tested Power (mW)
uC32 Microcontroller[31]	5	75.5	60	378	300
Raspberry Pi 3B+ Microprocessor[32]	5	1200	500	6000	2500
AKK KC03 Camera/Transmitter[33]	11.1	340	312	3770	3460
FS-iA6B RC Receiver[34]	5	20	34	100	170
Serial Telemetry Radio Kit Data Transmitter[35]	5	100	N/A	500	N/A
Total				10748	6930

The power used by the uC32 microcontroller was expected using the datasheet current required for executing code that is stored in the microcontroller. The microcontroller's typical amperage estimate of 75mA is added to a small increase of 0.5mA of power required to maintain I2C communication protocol, as specified on its datasheet. The microcontroller was then tested to run the code required to move the servos and motors, where the current was read through an ammeter, measuring the tested value of 60mA. The current is lower than the estimated datasheet current since not all pins are being used

The Raspberry Pi 3B+ microprocessor's estimated power requirement follows the maximum total current draw on the datasheet of 1200mA. While running the code necessary to read from sensor and control inputs, the current measured was variable between 300-500mA. Since it was unstable, the 500mA max was recorded as the tested current. The lower current is due to the processing power being used that is less than the microprocessor is capable of.

The AKK KC03 camera transmitter is connected to the camera module itself and receives the power for both to function. The provider stated the working current for the camera to be 135mA at 5V and the transmitter to work at 280mA average from 7-20V, which was assumed to be the current for a 11.1V power supply. The total current was adjusted for use at 11.1V for both, giving us a current at 340mA. This was tested by viewing the camera transmitter output on a screen with an ammeter in between the camera and our 11.1V power supply, giving us a lower current of 312mA. This difference can be attributed to the wide voltage input range of the transmitter and the vague working current stated by the provider.

The FS-iA6B RC receiver was stated to have a maximum current of 20mA at 5V by the provider. However, when tested, this part consumed 34mA at 5V, regardless of the input on the joysticks of the controller while connected. This is due to the RC controller continuously sending a signal to the receiver no matter the contents of the signal

The final control part is a Sparkfun telemetry radio kit data transmitter, which could not be tested due to the lack of data from sensors to transmit. Therefore, the transmitting current given by the provider of 100mA at 5V was kept in the power budget.

Overall, the control components only use about 64% of their estimated power, or 6.9W, due to the components not being used to their full operating capacities.

7.1.2 Sensors

A variety of sensors were implemented in the system, including an MPL311512A external pressure and temperature sensor, four HC-SR04 ultrasonic proximity detectors, an MTK 3339 GPS module, an ICM-20948 IMU orientation sensor, and an MPRL S0001 internal pressure sensor. These parts were also estimated to have a constant current requirement, which was first estimated using data from the sensors' datasheets and then tested using an ammeter while they were on and recording data. Figure 7.3 describes a block diagram for the tests done with the sensor modules. Table 7.2 shows the results of these estimations and experiments.

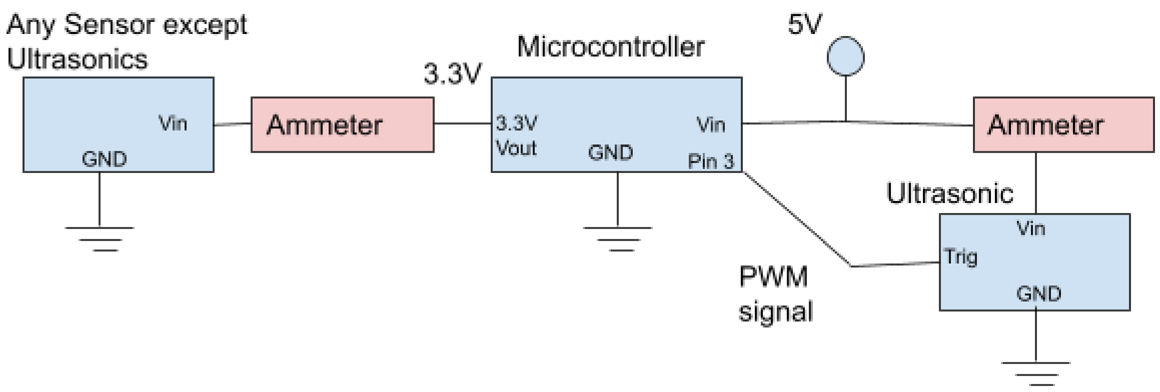


Fig. 7.3. Sensors Power Test

Sensors Test Process:

1. Connect the Vin pin of the microcontroller to any 5V source with a common ground
2. Connect connect an ultrasonic sensor to the 5V source with an ammeter in the middle, with a common ground and with the trigger pin connected to pin 3 on the microcontroller, which is running the ultrasonic code
3. Record current while the ultrasonic sensor is running after 5 seconds
4. Connect the Vin of any other sensor to the 3.3V Vout pin of the microcontroller with an ammeter in the middle and with a common ground
5. Record current through the sensor after 5 seconds have passed
6. Repeat step 4-6 until no more sensors are left to test

Table 7.2

Typical Power Required for Sensors in Flight

Part Name	Nominal Voltage (V)	Estimated Current (mA)	Tested Current (mA)	Estimated Power (mW)	Tested Power (mW)
MPL3115A2 External Pressure/Temperature Sensor[36]	3.3	0.2	0.16	0.875	0.528
HC-SR04 Ultrasonic Sensor[37](4 total)	5	15	2	300	40
MTK 3339 GPS Module[38]	3.3	20	22	66	73
ICM-20948 IMU Sensor[39]	3.3	3	3	10	10
MPRL S0001	3.3	4	N/A	13.2	N/A

Internal Pressure Sensor[40]					
Total				390	136

The MPL3115A2 external pressure sensor datasheet had a current estimate of 0.265mA at 2.5V, meaning if powered by 3.3V, it would be 0.2mA. The tested current was 0.16mA when connected to power. This difference can be due to the amount of times the sensor will be read, which was estimated at about 1 times per 2 seconds, while the datasheet says the expected current value is 1 time per second.

There are four separate HC-SR04 ultrasonic sensors, each also being used constantly throughout flight. Power for the ultrasonics is taken from its datasheet, which specifies a 15mA current at 5V. However, when one sensor was tested and returned readings, the current was only 2mA. This can be due to a 16hz rate the sensor is being used at, as the datasheet does not specify the max response rate of the sensor.

The MTK 3339 GPS module datasheet power was specified to typically be 20mA at 3.3V when a GPS signal has been acquired and is being tracked. It also had an estimated 25mA current at 3.3V when trying to acquire a signal. When tested, the current while acquiring a signal was correct at 25mA, but the tracking current was 22mA. Since the signal will be acquired within a minute, the average current was stated to be 22mA for a greater than 30 minute flight time.

The ICM-20948 IMU sensor's activity and expected power draw were taken from the datasheet at 3mA at 3.3V. When tested, the value was exactly 3mA, as stated on the datasheet.

The internal pressure sensor was unable to be tested since the code for it had not been implemented yet, but was estimated to be 4mA from the datasheet.

The sensors in total only require about 35% of the power estimated from datasheets, or 0.136W, owing to the low refresh rate and use of each sensor comparatively with what they are capable of.

7.1.3 Actuators

The actuators, which are the motors and the servos, are different from the other components because the difference in their own power requirements are large when based on their performance and are either individually testable or stated in their datasheets. The servo power was used from its datasheet for idle, working, and stall currents and then tested to see if the stall current was reached and to verify the other current values. The motor and ESC power requirements were calculated using a simulation test with our initial motors that gives thrust per RPM when the motors were combined with the propellers. This was done since the motors amount to more than 90% of the total power usage of the system. Since replacement motors needed to be found late in the project, simulation results were not done with them. Figures 7.4 and 7.5 show the system block diagrams of the tests done for the motors and servos. Table 7.3 summarizes the findings from the simulation and tests, where the low state for the motors will be hovering at 5N total thrust from the 4 motors and the 4 servos will be idle, while the high state for the motors will be 10N total thrust and the servos will be turning without stall.

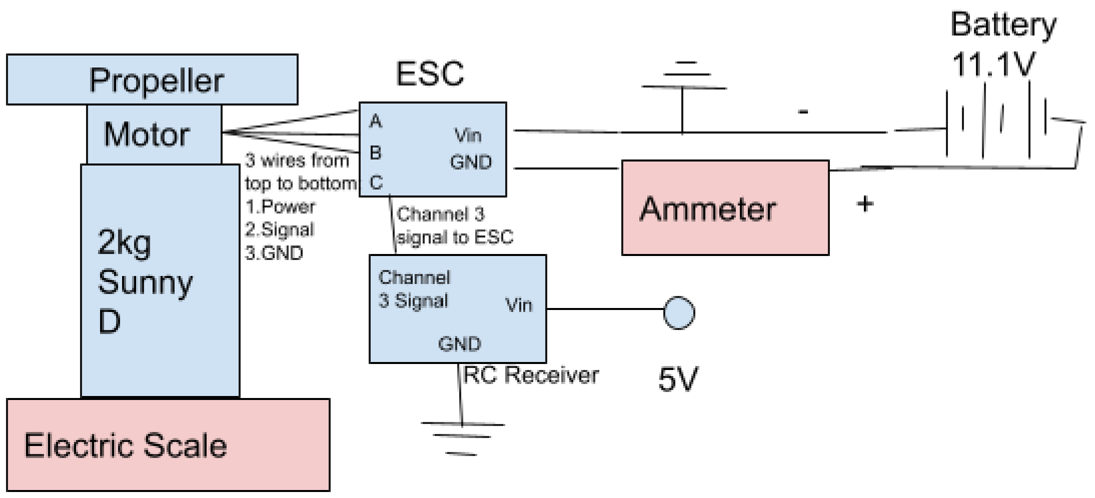


Fig. 7.4. Motor Power Test

Motor Test Procedure:

1. Tape a bottle greater than or equal to 2kg, in this case a sunny-D bottle to an electric scale
2. Tape the motor and propeller to the top of the bottle
3. Connect an ESC's 3 wires to the motor as shown in the diagram, with Power to A, signal to B, and Ground to C
4. Connect the ESC to an 11.1V battery and ground with an ammeter in between the power supply and the ESC
5. Connect the RC receiver to a 5V source to Vin, ground it, and connect the channel 3 of the RC receiver to the ESC
6. Turn on the RC controller, the motor should now beep once and it will be ready to spin
7. Increase the throttle 1% and wait 5 minutes, then record the weight changed by the motor and the current required after 5 seconds of continuous throttle
8. Repeat step 7 until the Ammeter cannot handle the current.

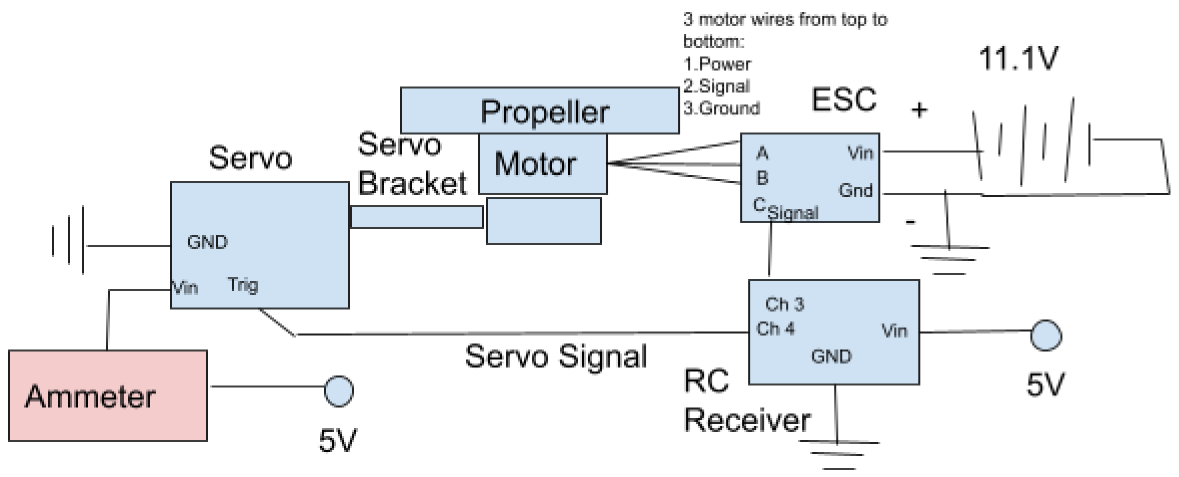


Fig. 7.5. Servo Power Test

Servo Test Procedure:

1. Tape Servo to a secure place horizontally with the motor and propeller on the motor bracket connected to the servo
2. Connect the servo Vin with a 5V supply and an Ammeter in between, with the trig pin of the servo connected to channel 4 of the RC receiver
3. Connect the motor with three wires to an ESC, as described in the diagram, with Power to A, signal to B, and Ground to C
4. Connect the ESC to the 11.1V battery supply and with a common ground with the servo and all other parts
5. Connect the RC receiver to a 5V power supply and ground, with the channel 3 pin being connected to the signal pin of the ESC
6. Run the motor at full throttle(29% for the replacement motors) and turn the servo continuously
7. Observe the Ammeter and record the highest current achieved while the servo is spinning

Table 7.3
Power Required for Motors and Servos

Part Name	Nominal Voltage (V)	Low State Current (mA)	High State Current (mA)	Low State Power (mW)	High State Power (mW)
SK3 2822-1275kv Motor[41](Intended)	11.1	1900(Expected) 1144(Tested)	3900(Expected) 3044(Tested)	84,000(Expected) 51,000(Tested)	173,000(Expected) 135,000(Tested)

D2822-2600kV Motor[42](Replacement)	11.1	1787(Tested)	4086(Tested)	79,000(Tested)	181,000 (Tested)
RC Sail Winch Servo[43]	5	3(Expected) 1(Tested)	350(Expected) 350(Tested)	60(Expected) 20(Tested)	7000(Expected) 7000(Tested)

The servo power was taken from its own datasheet. When not in operation, the servos require power equal to the low state power consumption since they are idle and not working. This is a result of the angle detection that our servos utilize, both when they are moving and when they are stationary. When they are moving, which happens when the drone is turning or correcting itself, it will then require the high state power consumption for the duration the servo it is moving.

These values were tested for the servo, requiring 1mA instead of 3mA from the datasheet while idle. The current while the servo was turning was tested and did not exceed 350mA as stated on the datasheet, although the current read through the servo with an ammeter was unsteady. While the motor was tested with the maximum thrust of 2.5N while on the servo, the servo did not stall due to opposing torque. This would have required a current of up to 1A, but did not occur since the motor torque is about 0.9Nm, while the torque created by a 2.5N thrust 6 inches from the servo would have a torque of about 0.3Nm.

The motor power was calculated using Figure 7.6, which is a thrust simulation of our motor and propeller at different rpm values. The two thrust values that were focused on were each of the 4 motors providing 1.25N of thrust for a 5N drone to hover and 2.5N of thrust per motor to oppose maximum drag of 8N and weight of 5N combined using trigonometry, which equals about 9.3N. Since the thrust is greater than the expected max drag at our maximum drone speed, this fulfills STR 2.0.0, Drone Speed, but this cannot be verified until tested in an outdoor environment.

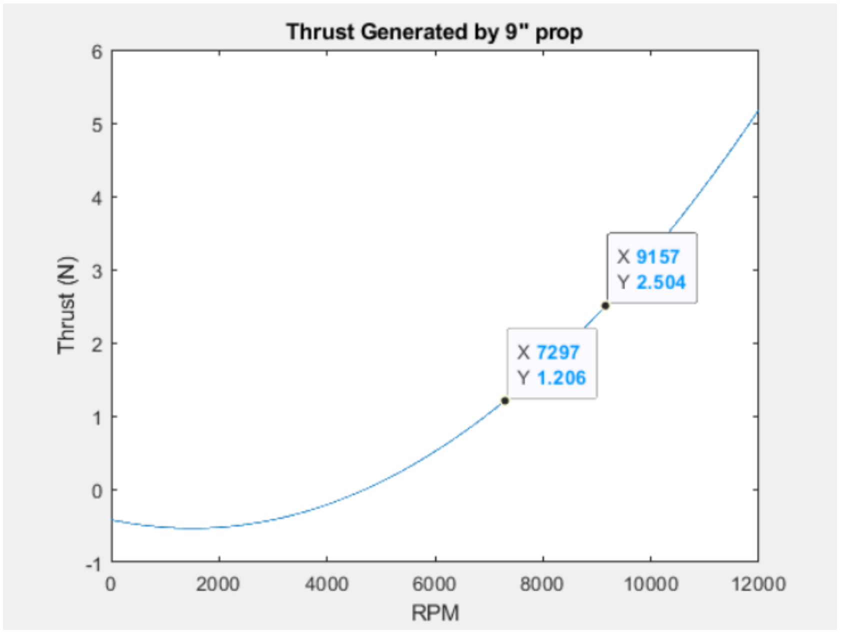


Fig. 7.6. Results of simulated propeller and motor thrust at different rpms using the intended motors.

Using equation 7.1, the power is estimated using the torque needed to turn the motors at a certain rpm. Power is equal to torque times angular velocity, which means for power, the Ω would be cubed instead of squared.

$P = Mx * \Omega = \frac{\rho * \Omega^3 * D^5 * Cq}{4\pi^2}$ Equation 7.1[44] Power of each motor; Torque is labeled Mx

Where ρ is approximately 1.225kg/m^3, or the density of air at room temperature and at sea level. Ω is the angular velocity of the propeller that is dependent on 0-100% throttle and was simulated from 0-12000rpm, D is the diameter of the propeller or 0.2286m, and Cq is the propeller torque constant that is dependent on the wind speed flowing through the propeller. Cq ranges from 0.00184 at still winds to 0.0019 at a wind speed of 8.9ms, which is the maximum windspeed the drone should be able to fly against based on our system technical requirements. For maximum power needed, the wind speed is assumed to be max. Using this equation while varying the throttle of the propeller gives us Table 7.4 and Figure 7.7 for power required for a certain throttle by multiplying the torque by the angular velocity expected from the motor and propeller based on the results of our simulation.

Table 7.4
Power Required Per Throttle in Simulation for Intended Motor

Throttle	Power (W) (20mph wind)	Power (W) (0mph wind)
0%	0	0
10%	0.1	0.1
20%	0.7	0.7
30%	2.5	2.2
40%	5.7	5.3
50%	11.1	10.4
60%	19	18
70%	29.9	28.5
77%	38.9	38
80%	44.4	42.6
90%	63	60.7
100%	86.1	83.2

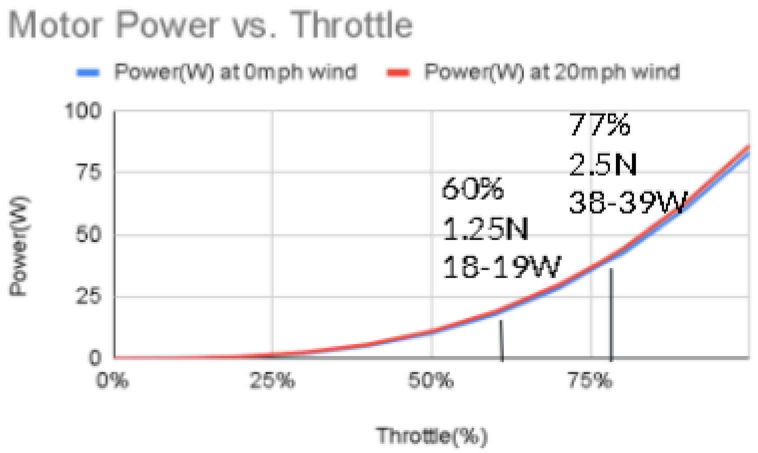


Fig. 7.7. Power required depending on throttle applied to the motors

In this chart, the limits of the motor throttle is decided by the amount of thrust required of the motor. Around 60% of this throttle is necessary to hover with 1.25N of thrust being generated by the motor and propeller. Multiplied by four, this gives us enough force to counter our 5N effective weight of the buoyant drone. This power level was chosen as the idle state power due to the drone having to hover if it is idle. 77% throttle would give us twice the amount of force, which was chosen as the limit for the max acceleration of the drone at 10N total in order to meet our system technical requirement of a drone speed of 5mph in 15mph winds, giving a max of 8N thrust as done in a drag estimation simulation. This was chosen as the maneuvering state power requirement, since the actual throttle used for maneuvering will be arbitrarily controlled by the user. Instead, the minimum and maximum power needed to hover and at max performance, respectively, are used to give a range of throttle and power that will be needed by the system. The limitations of this simplification are that the drone will not always be moving with maximum thrust, however this simplification allows us to calculate the absolute minimum and maximum flight times while an actual flight time can be found by integrating the amount of throttle used over time, as discussed later in section 7.2.3 in this chapter.

The results of testing the motors are found in Table 7.5 for the intended motors and Table 7.6 for the replacement motors. These tables show the thrust and power needed at a certain throttle level. The data is graphed in Figure 7.8 for the intended motors and Figure 7.9 for the replacement motors.

Table 7.5

Thrust and Power Needed by Intended Motors with Different Throttle Values

Motor Throttle	Motor Thrust (g)	Motor Thrust (N)	ESC Current (A) (at 11.34V)	ESC Power Required (W)
0%	0	0	0.04	0.4536
5%	1	0.0098	0.055	0.6237
10%	10	0.098	0.097	1.09998
15%	26	0.2548	0.176	1.99584
20%	43	0.4214	0.293	3.32262
25%	66	0.6468	0.478	5.42052
30%	94	0.9212	0.708	8.02872
35%	120	1.176	1.01	11.4534
37%	128	1.2544	1.12	12.7008
40%	149	1.4602	1.33	15.0822
45%	176	1.7248	1.69	19.1646
50%	208	2.0384	2.17	24.6078
55%	238	2.3324	2.67	30.2778
58%	251	2.4598	2.98	33.7932
60%	263	2.5774	3.16	35.8344
65%	294	2.8812	3.8	43.092

70%	305	2.989	4.47	50.6898
75%	345	3.381	5.24	59.4216
80%	371	3.6358	5.92	67.1328
85%	385	3.773	6.55	74.277
90%	415	4.067	7.47	84.7098
95%	430	4.214	7.84	88.9056
100%	452	4.4296	8.6	97.524

Table 7.6
Thrust and Power Needed by Replacement Motors with Different Throttle Values

Motor Throttle	Motor Thrust(g)	Motor Thrust(N)	ESC Current(A) (at 11.34V)	ESC Power Required(W)
0%	0	0	0.046	0.52164
5%	0	0	0.046	0.52164
10%	74	0.7252	0.986	11.18124
15%	114	1.1172	1.61	18.2574
16%	125	1.225	1.75	19.845
17%	131	1.2838	1.9	21.546
20%	158	1.5484	2.37	26.8758
25%	215	2.107	3.23	36.6282
28%	250	2.45	4	45.36
29%	257	2.5186	4.25	48.195
30%	266	2.6068	4.44	50.3496

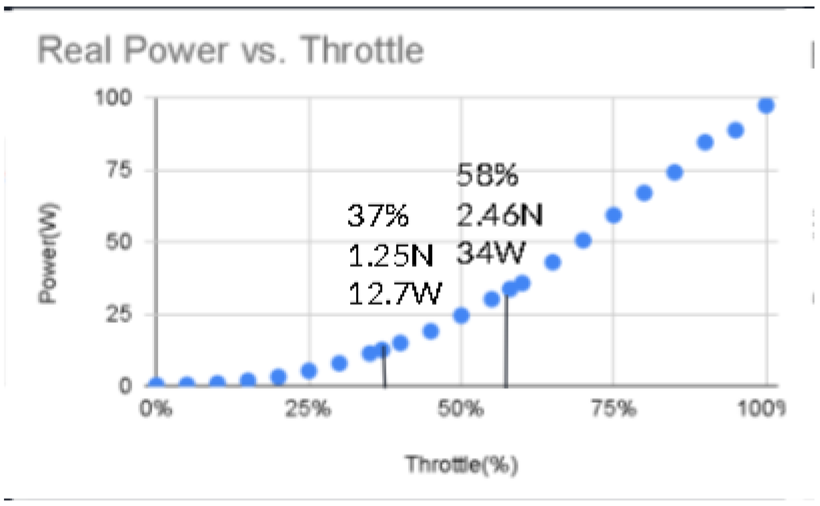


Fig. 7.8. Throttle over Power Needed as Tested with the Intended Motors.

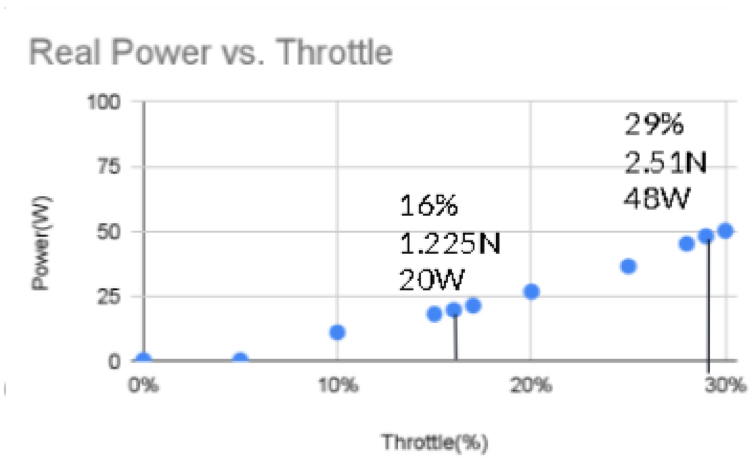


Fig. 7.9. Throttle over Power Needed as Tested with the Replacement Motors.

The results of the motor tests show that the intended motors are more power efficient for the same throttle. It also shows that the replacement motors are less efficient than the expected and tested values for the old motors, owing to the twice as high kV rating as the intended motors. Due to the higher kV rating, the motor will push harder to achieve a higher RPM with a certain level of voltage and when combined with the prop we have, will lose more power to heat due to the higher torque applied with the same RPM to get the same thrust as the intended motor.

The results from the tests and simulations are combined to give a power range that the motors will need in between the minimum and maximum power requirements of the motors, which is based on the thrust the motor will provide at that power up to the maximum thrust of 2.5N. This is summarized in Table 7.7.

Table 7.7
Power Ranges for Simulated, Intended and Replacement Motors

Motor & Verification	Power Range (W)
Simulated Power For Intended Motors	84.3-173
Actual Power For Intended Motors	50.8-135
Actual Power for Replacement Motors	79.3-181

7.2 Powering & Distribution

With all the components of the system decided, it is necessary to divide the power required by their nominal voltages, giving us 3.3V, 5V, and 11.1V power rails. When parts require the same voltage, they can be interconnected on one power line. For the battery selection before selecting the voltage, we have chosen a Lithium-Polymer(LiPo) battery due to its high energy density and popularity with drone usage[46]. When parts were defined by 11.1V nominal voltage, this meant they were specified for use with a three cell LiPo battery, for example, that was the max amount of cells stated to work with our motors. The voltage rails lower than this must be connected to the battery using voltage regulators to get a lower voltage. These regulators have a rated efficiency that means energy must be used to convert the voltage to a lower value and that energy is lost to heat. The total power required by the components as stated in section 7.1, added together with the heat losses in section 7.2.1 will give us the total power needed for the system to give us a 30 minute minimum flight time in order to meet the system technical requirement for a 30 minute normal autonomous flight.

7.2.1 Voltage Rails and Regulators

For the 3.3V, 5V, and 11.1V power rails, each one must be supplied by the battery. This means regulators must be used to drop down that voltage. The 11.1V nominal rail is straight from the battery, but will have heat dissipation from the 7.5ft wires that are run from the gondola to the motors themselves. The 5V rails will be achieved through a switching regulator to save power due to the high current needed and the variable voltage was allowed in each 5V part when tested, while the 3.3V rails will be gained from a linear regulator due to the very low current needed and the stable voltage needed for sensor operation. A summary of these rails and their power losses are shown in Table 7.8.

A switching regulator was chosen for the 5V rail due to the high current, which means a high power loss from heat if the regulator were linear, and because each part that is powered by 5V was specified that a variable supply voltage is allowed and it was tested that each part worked without error while powered by the regulator. Two Pololu 5V switching regulators were used, with a minimum efficiency of 85% at our battery voltage range of 9.6-12.6V.[45]. Two were used because the servos have a stall current of about 1A each and the regulator is only rated for 5A. Based on the power required for these rails, the efficiencies of the regulators were taken into account and the power loss due to heat was added onto the power required for usage of the system.

A linear regulator was used for the 3.3V power rail because of the low current required, and because our microcontroller has this function built in. Even though the 3.3V regulator is powered by the 5V microcontroller, the switching regulator that powers the 5V microcontroller will dampen the efficiency losses from dropping down from the battery voltage, while the 3.3V linear regulator will only have to drop from 5V. Due to the low current, there will be a low power loss, verifying that the 3.3V linear regulator on the uC32 microcontroller can be used.

Table 7.8
Voltage Rails and Heat Dissipation

Rail Voltage (V)	Max Current (mA)	Max Current Allowed by Regulator (mA)	Max Heat Dissipation (mW)	Regulator Datasheet and Comments
11.1 Nominal (12.6-9.6V)	12906	N/A	471	Heat comes from 4ft wires to and from each motor 1.588 ohms per 1000 feet for 14 AWG at 3.044A max
5	1400	5000	1235	5V switching regulator for servos, 85% minimum efficiency[45]
3.3	41	1000	70	3.3V Regulator used on microcontroller[31]
1.8	750	5000	662	Same 5V Switching Regulator used for every other 5V part 85% efficient[45]
Total			2600 Max	

7.2.2 Battery Selection

Now that we have each component's power, we can add them all together to get the total power needed for the system, which we can use to calculate flight time. The total power of the system is shown in Table 7.19. This table shows the expected power from the intended motor simulation and datasheet power values, while also detailing the actual power from the tests conducted. However, the initial battery selection was done with the expected power and will be discussed at the end of section 7.2.2.

Table 7.9
Power Ranges for System and Each Part Category

Part	Sensors	Control Mechanisms	Servos	Motors	Heat Loss	Total
Expected Power(W)	.419	10.75	.12-7.0	84.3-173 (Simulated)	.94-2.8(1275kV)	100-194(1275kV)
Actual Power(W)	.136	6.86	.04-7.0	50.8-135(1275kV) 79.3-181(2600kV)	.94-2.4(1275kV) 1.3-2.6(2600kV)	58.8-151(1275kV) 87.3-197(2600kV)

With the total power, our system technical requirement for flight time requires a normal autonomous flight time of 30 minutes. Due to the arbitrary nature of this requirement, which was given to us by our client, the safest way to fulfill this requirement would be to ensure the minimum flight time would have to be greater than 30 minutes. This means that the maximum power required for a 30 minute flight must be below the capacity of our battery. This power amounts to the maximum expected power of 194W times half an hour, which equals 97Wh of energy required.

Selecting a battery that is capable of 97Wh discharge starts with selecting the type of battery. We have chosen a Lithium-Polymer(LiPo), due to the high energy density, low weight, and higher safety compared to Lithium-Ion batteries[46]. For LiPo batteries, each cell has a 3.7V nominal voltage and they can range from 3V-4.2V depending on the amount of energy left in the battery. The motors chosen for the drone are able to handle up to 3 cells of a LiPo battery, giving us a maximum voltage of 12.6V, a minimum voltage of 9.6V and a nominal voltage of 11.1V. The supplier for our LiPo battery was chosen as the company of MaxAmps Batteries, due to the highly variable selection of batteries and their high safety rating despite their higher cost.

LiPo batteries are rated by milliAmp-hours(mAh), this means that it is rated for releasing that amount of current for one hour, which will cause the voltage of the battery to drain from 4.2V per cell max to 3.0V per cell minimum. However, releasing the entire capacity is very harmful to the battery if a cell falls below 3.0V and continues to discharge[47]. For this reason, research into this issue concluded that discharging the battery fully had minimal effects on the long-term lifetime of the battery compared with discharging it partially[48]. A linear depth of discharge was chosen as an approximation due to the lack of a depth of discharge curve. This led to keeping the voltage of each cell above 3.2V per cell as a safe buffer to enable the drone to find a spot to land and to conduct emergency landing procedures without going below 3V[49]. Assuming a linear depth of discharge, this will allow us to discharge 83% of the battery without going below 3.2V per cell. Another assumption for a depth of discharge was researched and concluded that about 90% of a LiPo battery can be discharged when approaching 3.2V[49], but this was not able to be tested or verified with the 83% estimation due to not being able to measure the current used over a long enough time to estimate the energy discharged from the battery and the voltage that the discharge resulted in.

Since the current being used in our system is at different voltages, it was necessary to measure the battery in milliWatt-hours(mWh) instead, which is multiplying the mAh by the nominal voltage of the battery. Therefore, with these restrictions that 83% of our battery capacity must be above the required minimum flight time power of 97,000mWh, the battery that was selected was a 3 cell, 11000mAh battery[50]. 83% of the mWh capacity of the battery is 101,750mWh, which is above the required battery capacity required. There is about 4,750 mWh of excess power because it was the smallest battery size our provider had that would meet our minimum flight time power requirement.

7.2.3 Flight Time

With our battery chosen, we can compare the battery capacity with the power required for keeping the drone in the air. This will allow us to calculate the minimum and maximum flight times. By calculating the maximum power and the minimum power required for the expected power with the intended motors, the tested power with the intended motors, and the tested power with the replacement motors, we can get the minimum and maximum flight times in Table 7.10, where the maximum energy required will be the motors always opposing maximum drag and the minimum energy required will be while the drone is only hovering.

Table 7.10
Minimum and Maximum Flight Times & Power Requirements with 101750mWh Battery

Motors & Verification	Always Opposing Max Drag Flight Time	Hovering Only Flight Time
(Intended)1275kV Simulated	31.8 minutes	66 minutes
(Intended)1275 kV with 20A 4 in 1 ESC	40.2 minutes	103.2 minutes
(Replacement)2600 kV with 4 single 30A ESCs	30.6 minutes	69.6 minutes

A limitation of these times in Table 7.10 are assuming the drone is at maximum and minimum motor and servo performance, respectively, meaning that for the minimum time, the drone is always turning and pushing the motors to their limit while the maximum time is for the drone hovering in place. Calculating the actual flight time of a flight that is in between the minimum and maximum performance will require integrating the power needed for the throttle the motors are receiving over the time it is used. A chart that shows the equations used is displayed in Figure 7.8 for the intended motors and Figure 7.9 for the replacement motors. Integrating by the throttle used over time plus the time the servos are used over time for turning and also when they are not turning will give us the variable power to be added to the constant component power drain for the sensors and controls to give us the power required for a time and performance variable flight, and therefore its flight time if compared with our battery capacity.

7.3 Conclusion

Even though this system has not been fabricated completely, the individual component power tests for each part justifies that theoretically the drone can reach a 40 minute flight time at the minimal 20mph airspeed. Since the minimum flight time, verified through power draw analysis, is above 30 minutes, STR 1.0.0, Flight Time, can be met. The requirement cannot be experimentally verified until a flight test has been completed with their flight time measured above 30 minutes.

Chapter 8: Full System Simulation and Validations of Drone Design

A drone simulation was needed to best replicate real world conditions within a controlled environment, so we chose V-rep as our simulation platform. Within this chapter, we will explain how we designed our system and the functions we implemented to apply internal buoyancy, external drag, and propeller forces. Using these fundamental forces, we were able to create a full system design of our drone with a rudimentary open loop remote control system, which allowed us to verify whether we would meet our STR 2.0.0, Minimum Drone Speed. Though we were not able to implement the closed loop remote control, we have listed out the tasks we were able to perform and possible future testing that could be done on a completed RC system. Overall, the simulation chapter also covers physical responses to remote controller inputs, noisy sensor inputs, and control system responses to identify necessary flight responses before physical testing to inform of any prerequisite design changes.

8.1 Choosing a Full System Simulation Environment for the Buoyant Drone

Before simulation began we looked at a variety of options for simulation environments where we would be able to test our drone to its full capacity. We ended up finding two programmable simulation environments that were open source and the most recommended for beginners, since nobody on our team had any simulation engineering experience. As our first primary choice, Gazebo had many of the great features V-rep included, such as sensor integration in robotic hardware, testing multiple robotic controls simultaneously, and an active developing community for newer users to be able to ask questions. But the drawbacks that kept us from using Gazebo were primarily that the installation of Gazebo required Ubuntu packages making it very difficult to run on Windows and CAD files could not be imported into Gazebo, whereas Unity was used instead.

On the other hand, while a week was spent trying to get Gazebo to install on our device, V-rep had taken only a single day. Although V-rep utilizes a robust real-time physics engine to simulate a diverse multitude of robotic designs, its support for aerodynamic features, an essential parameter our design takes advantage of, is fairly lacking. Due to the constraints of time spent on our simulation design, we ended up deciding on V-rep as our platform of choice, and so our implemented solution to the lack of aerodynamic support is provided in the following section.

8.2 Necessary Building Blocks for a Practical Simulation Environment

Within the simulation we chose the following forces to closely represent real world conditions: buoyancy, gravity, thrust, and drag. The implementation of each of these forces will be explored in further detail within this section.

8.2.1 The Buoyant Moment

Buoyancy is the essential lifting force of our drone, derived from the lift bag. However, one specific keynote to take into consideration when adding buoyancy is that the physics engines in V-rep were not built to handle the complicated nature of aerodynamics principles, such as the density of a fluid, volume of liquid displaced, and local acceleration of uneven surfaces due to buoyant factors. Simply stated, an atmosphere in V-rep does not exist.

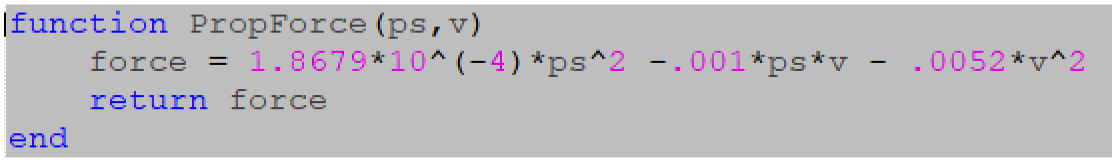
Due to the inaccuracies in our ability to simulate these properties of real-life fluids, we rely on the most simple, symbolic form of the buoyant force, a single upward force vector. And to apply this buoyant force, a lift bag object is mounted within the larger envelope to imitate the physical design. The purpose of this design feature is not only to keep the masses of the lift bag and envelope separate, allowing their own local areas of gravitational forces, but also to shift the buoyant force higher than the center of mass, creating a buoyant moment in the correct location.

8.2.2 Individual Gravity Forces and Momentums

Given a working CAD model with various interconnected parts, masses, and individual orientations, individual gravity forces can be applied locally on a full system model. Assuming evenly distributed parts, these individual gravity forces are applied directly to the center of that object. All parts are assigned masses to them, which include the envelope, lift bag, gondola with all electronics and hardware within it, ultrasonic, propulsion system mounts along with the embedded servo weights, D-shafts, and the propellers with their attached motor weights. In addition, these parts are provided with surrounding respondable objects that react to collision according to the momentum they are moving with.[54]

8.2.3 Implementing Propeller Forces

Propeller forces are implemented as a secondary lifting force we are able to control using the propulsion system. Therefore to make the propeller forces realistic,the propeller forces written are non-linear and adjusted entirely by the propeller speeds they are provided. We use this equation to calculate the forces required, simplifying all coefficients and adjusting the coefficients to scale alongside the max thrust force. The equation in figure 8.1 is based on$F_x = \rho n^2 D^4 c_T$(4.1), where the constants are defined by our propellers dimensions. For more information about the propeller force equation see Chapter 4.



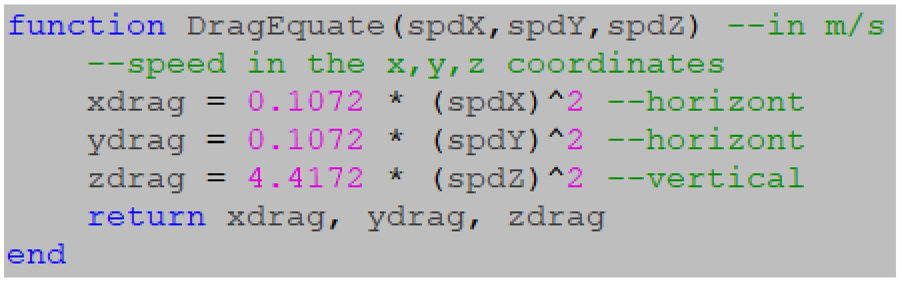
```
function PropForce(ps,v)
    force = 1.8679*10^(-4)*ps^2 -.001*ps*v - .0052*v^2
    return force
end
```

Fig. 8.1. Propeller Force Function where 'ps' Represents the Propeller Speeds and 'v', the Wind Velocity

The forces are applied to the respondable propeller objects and speeds are applied to a non-respondable propeller object separately, since once again buoyancy is not inherently within the sim so these forces must be manually coded to work as intended. To test that these forces were working, we turned the propeller speeds to maximum on individual propellers so we would know that they provided an accurate torque to the overall drone. When tested, each corner provided the appropriate amount of torque to rotate the drone in the direction that a propeller pointed, along the curvature of the envelope, from one corner to the opposite.

8.2.4 Implementing Drag Forces

As a factor limiting our drone's capabilities, drag forces were implemented as two distinct forces in two distinct areas, drag forces due to a wind's vector force and drag forces due to the air displaced by the movement of the drone. In order to change the wind's direction to point in any direction we chose, the coordinates were aligned on a spherical axis: wind speed changes along the radius, theta along the x-y plane similar to yaw, and phi along the z-axis much like pitch. The vector in spherical coordinates is then converted to rectangular coordinates and applied to the drag equation, $Drag\ Force = F_d = \frac{C_d \rho A U^2}{2}$ (2.2) and as shown in Figure 8.2, providing individual drag forces in every direction. The coefficients were based on the dimensions found in section 3.1.4.



```
function DragEquate(spdX,spdY,spdZ) --in m/s
    --speed in the x,y,z coordinates
    xdrag = 0.1072 * (spdX)^2 --horizont
    ydrag = 0.1072 * (spdY)^2 --horizont
    zdrag = 4.4172 * (spdZ)^2 --vertical
    return xdrag, ydrag, zdrag
end
```

Fig. 8.2. Drag Equate Function Showing the Drag Force Equations in Cartesian Coordinates.

The drag force equations in the 'DragEquate' function, figure 8.2, are simplified to a single coefficient and variable of airspeed as shown above. Drag force, due to air displaced in the drone's direction of motion, uses the same equations to calculate drag force applied by the wind; however, the variables of velocity in three directions are replaced using the velocities of the drone and the force is applied opposite only to the direction the drone is moving. By applying both of these forces separately, three dimensional forces can be applied on cartesian coordinates without the assistance of a rotational matrix at whichever direction drone or wind is specified. By operating both drag forces and propeller forces together, the drone demonstrates capability of moving forward within a specified quantity of headwind.

8.3 Open Loop Remote Control Implementation

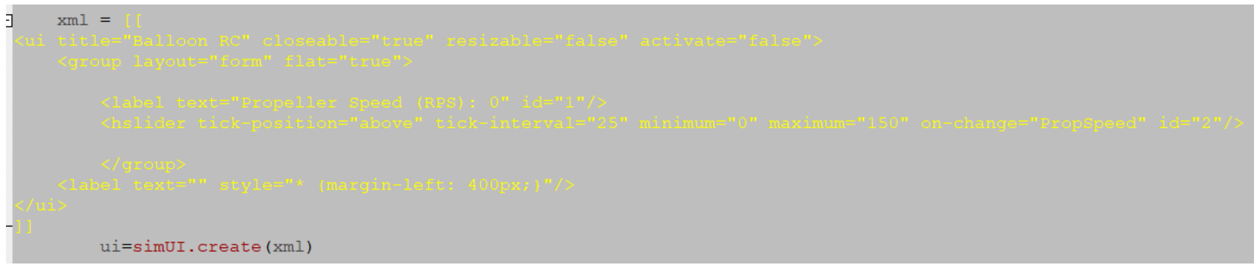
In this section the attempted verification of the following STRs will be discussed:

2.0.0 Drone shall be able to fly at least 5mph in winds up to 15mph.

3.0.0 The drone should have RC control implementation to allow for direct control of the drone. The drone can start in this state, or be switched to from autonomous control.

8.3.1 Open Loop Remote Control GUI Explained

In an open loop remote control, propeller speeds were adjusted manually and for all motors the same propeller speeds were applied.



```
    xml = [[
<ui title="Balloon RC" closeable="true" resizable="false" activate="false">
    <group layout="form" flat="true">

        <label text="Propeller Speed (RPS): 0" id="1"/>
        <hslider tick-position="above" tick-interval="25" minimum="0" maximum="150" on-change="PropSpeed" id="2"/>

        </group>
    <label text="" style="* {margin-left: 400px;}"/>
</ui>
]]
        ui=simUI.create(xml)
```

Fig. 8.3.a. Example Code Snippet, Which Shows how a GUI Slider for "Propeller Speed" is Created Using XML Code in V-rep.

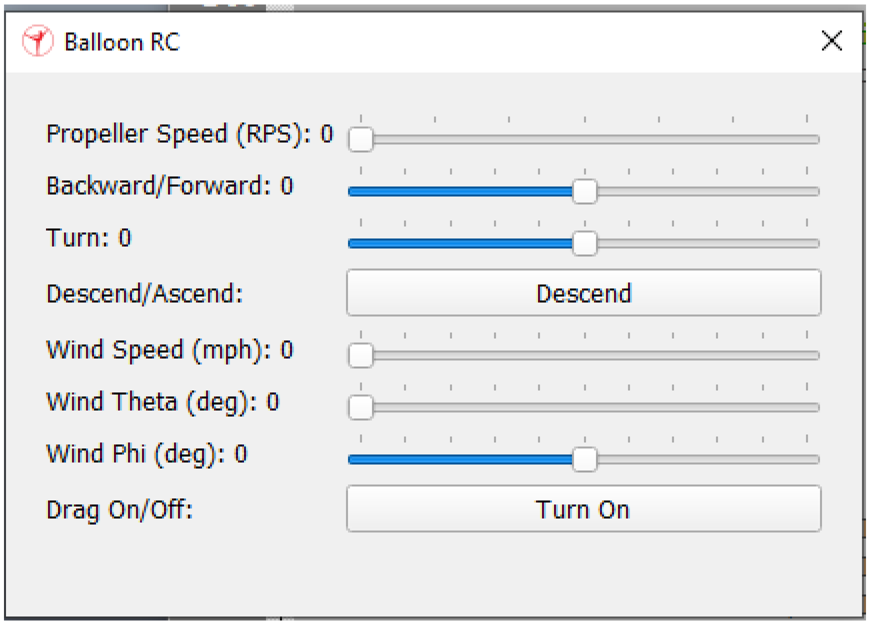


Fig. 8.3.b. Input Layout of Open Loop Control is Generated in GUI Format as Shown Above.

As mentioned in the previous section 8.2, the propeller speed and wind controls, illustrated above, control the propeller forces and wind vector, respectively. However, the propeller forces are not fully completed without adjusting the propeller force directions. Therefore, an open loop control must be created to test the drone within varying flight conditions. As previously mentioned in the controls system Chapter 6, open loop control was designed to change the direction of the propellers, by using factors for thrust and turn to rotate the servos in pairs.

8.3.2 Open Loop RC Testing

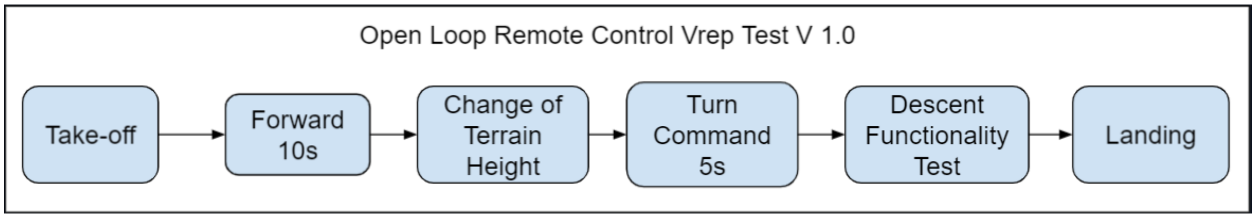

Open Loop Remote Control Vrep Test V 1.0
Take-off
Forward 10s
Change of Terrain Height
Turn Command 5s
Descent Functionality Test
Landing

Fig. 8.4. Block Diagram of a Series of System Level Tests

The block diagram, shown in figure 8.4, shows the order in which separate tests are performed to verify both a working drone model and open loop control system. Before conducting a test, drag should be turned on. Starting at the left side of the simulation platform, take off is initiated and propellers speeds are set to max throttle until a height of 5m is reached. Using the forward slider, the propellers are tilted forward at around 70 degrees at maximum propeller speeds for optimal lift capacity, when the propeller force's vertical acceleration cancels out its gravitational acceleration.

$$4F_{max\_thrust} * sin\theta = F_g - F_{buoyancy} \quad (8.1)$$

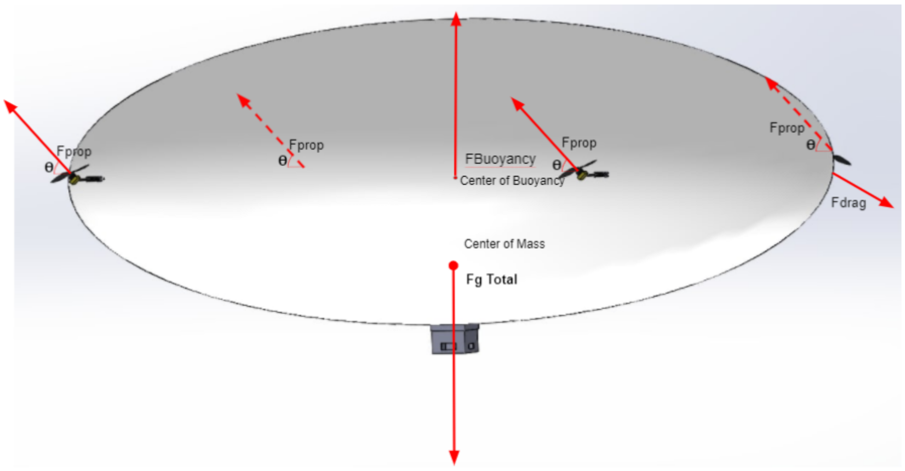
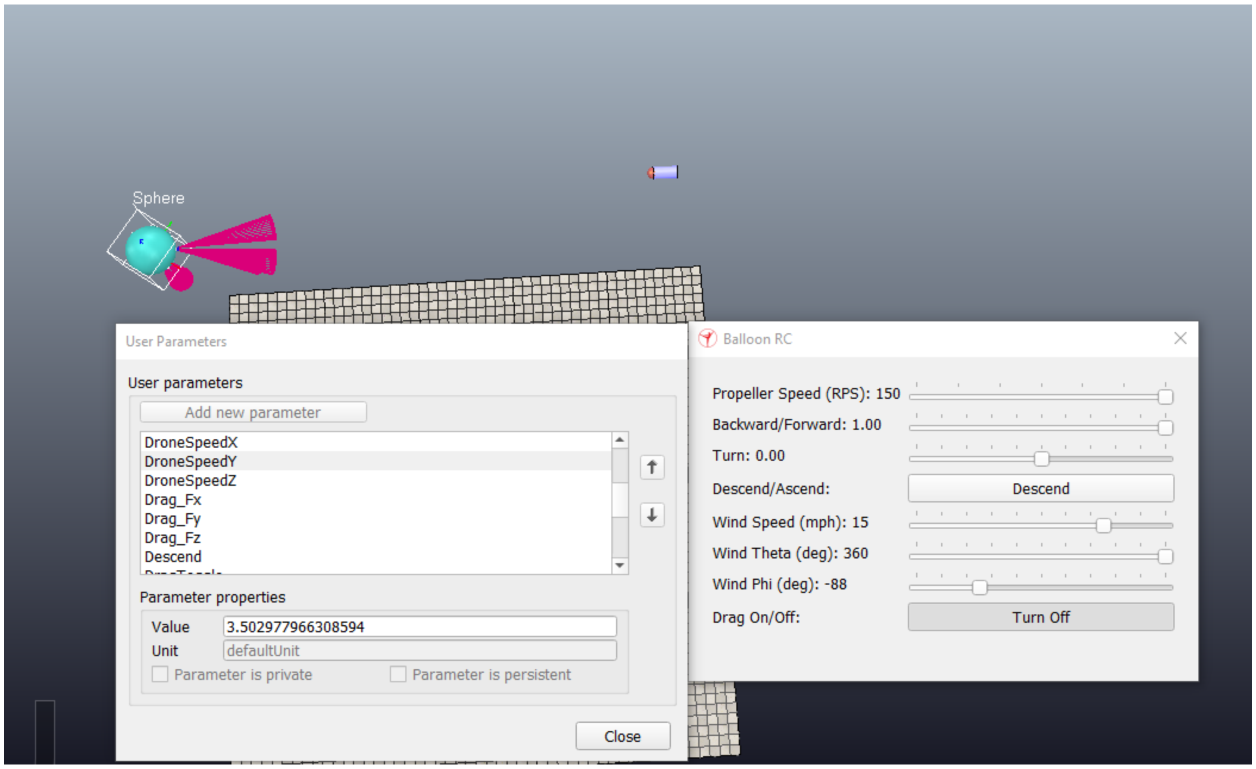


Fig. 2.1. Force Diagram

The drone is then presented with a change in terrain to test its ability to overcome obstacles in a drone's flight path, either by adjusting propeller speeds or propeller direction. A turn command is then issued by increasing or decreasing the turn factor slider, which causes the balloon to rotate either clockwise or counterclockwise. Descent functionality flips the propeller controls across the x-y plane, which could be used if the drone encounters heavy updrafts of wind. And at the end of a test, landing the drone requires utmost precision when adjusting propeller speeds in order to reduce vertical velocity as close to zero as possible before landing.

8.3.2 Open Loop RC Analysis of Tests

Under no wind conditions, drone flight was seen to be wobbly due to an uneven distribution of mass between the gondola and envelope. The drone was capable of making wide turns with slight difficulty, which was made possible with multiple turn commands.



DroneSpeedX
DroneSpeedY
DroneSpeedZ
Drag_Fx
Drag_Fy
Drag_Fz
Descend

Parameter properties

Value 3.502977966308594
Unit defaultUnit
Parameter is private
Parameter is persistent

Propeller Speed (RPS): 150
Backward/Forward: 1.00
Turn: 0.00
Descend/Ascend: Descend
Wind Speed (mph): 15
Wind Theta (deg): 360
Wind Phi (deg): -88
Drag On/Off: Turn Off

Fig. 8.5. Drone Speed is Shown to be 3.502 m/s on the left column reading 'User Parameters' during a flight test with drag turned on

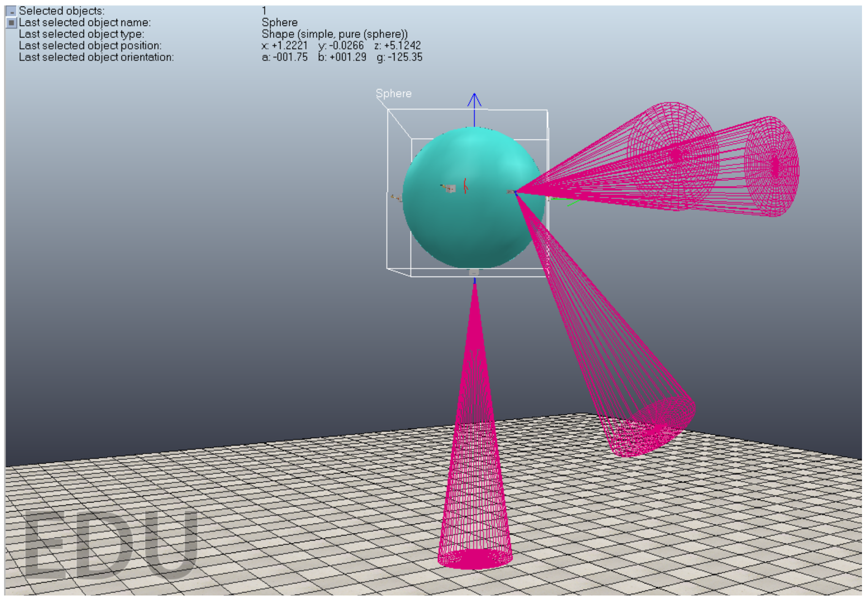
With the throttle and wind speed set to 1.00 and 15mph, respectively, on the right hand RC GUI panel, it was possible to verify that our drone could fly at least 5 mph within a 15 mph headwind, This verified STR 2.0.0, Drone Speed. for our drones optimal design. At the same time with maximum velocity wind conditions, steering the drone using entirely open loop controls was very difficult. Since the drone relies on its ability to move in the direction of either forward or backward, slight misdirections due to wind can cause the drone to veer away from its intended direction. With a large central mass, rotational momentum can be difficult to correct. Therefore, once the drone begins rotating, propellers will need to correct by applying a torque in the opposite direction of the drone. Though it is possible to correct for drone movement using an open loop remote control system, closed loop remote control needs to be implemented in order for the drone to be able to withstand faster wind conditions and make wider turns with less corrective measures needing to be issued.

8.4 Sensor Array Implementation

In this section the attempted verification of the following STRs will be discussed:

4.2.1 The sensors shall be able to monitor the area in front of the drone in order to maintain a constant height of 1 m.

4.2.2 The sensor shall be able to monitor altitudes above 4m for drone altitude awareness.



```
Selected objects:                  1
Last selected object name:         Sphere
Last selected object type:         Shape (simple, pure (sphere))
Last selected object position:     x: +1.2221   y: -0.0266   z: +5.1242
Last selected object orientation:  a: -001.75   b: +001.29   g: -125.35
```

Fig. 8.6. Drone in V-rep Simulation Shown with Ultrasonic Sensors.

Sensor data was recorded for the drone, which included data from the barometer, GPS, ultrasonic, and IMU. In order to account for the noise during actual flight and test our RC response to the varying tolerances of the sensors, we decided to add noise to these sensors by applying a Gaussian distribution centered around their ideal values to simulate more realistic data for our control system. Provided a variance and mean, our function outputs a randomly generated number, which is located along the gaussian curve. The variances we found were determined by data sheets with the following values: Barometer: ± 0.4 kPa; GPS: ± 50m (CEP); Ultrasonic: ± 3mm; and IMU: ± 1.5%.

Two standard deviations were recorded for each value meaning pseudo noise provides values 95% of time within the given values of noise. Once a sensor array is completed with the appropriate considerations for noise, sensor data could be collected and employed within a closed loop remote control system to maintain further stability.

8.5 Closed Loop Remote Control Implementation

In this section the attempted verification of the following STRs will be discussed:

3.0.0 The drone should have RC control implementation to allow for direct control of the drone. The drone can start in this state, or be switched to from autonomous control.

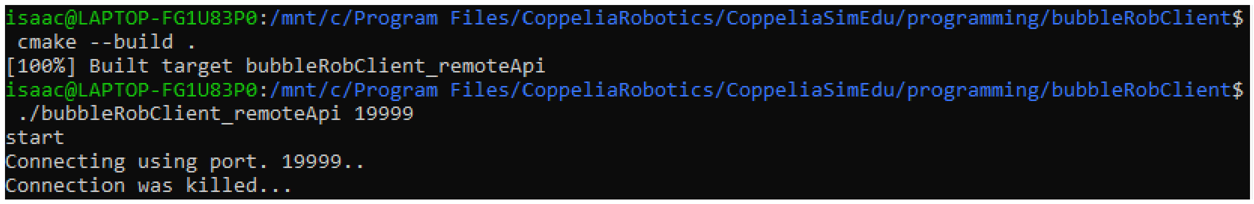
3.1.0 The software shall be fast enough to respond quickly to all user commands and error handling.

A closed loop remote control would allow our drone the ability to navigate itself with greater controllability in high velocity wind conditions, due to the self correcting characteristics of PID controllers. Although we were not able to complete the closed loop remote control implementation , we have listed out the tasks we were able to perform and possible future testing that could be done on a completed closed loop RC system.

8.5.1 Implementing a Communication Network Between V-rep and Externally Written Code

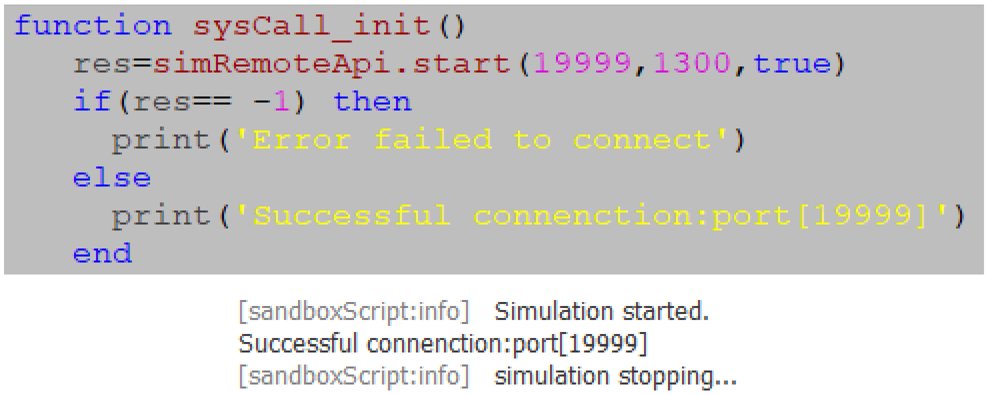
One of the first tasks to implement a closed loop remote control in V-rep is establishing a communication network using remote application programming interface(API) so that V-rep could interact with our control system code written outside V-rep. In our case, the external code was written in Matlab. However, interfacing V-rep with Matlab code proved to be a much more daunting task than we had anticipated. Simulink, the native language of Matlab, is incompatible with remote controlled code and therefore needed to be converted to C or C++ for its capability to compile with the greatest run speed on our microcontroller. Before compiling C++ on a Windows OS, first of all, an Ubuntu terminal is required to run gcc commands. However, because the Ubuntu OS does not run on Windows, Linux is required to be installed prior to running any terminal commands.

Once all packages are installed and Ubuntu is able to launch, in order to connect the server side, V-rep, to our client program, Visual Studio Code(VS Code), we needed to include a number of required V-rep files into VS Code. We found that due to vague or unspecified instructions on outdated V-rep user manuals[52], when the required files were found, further files from further directories within V-rep needed to be included as well. To solve the problem of missing file inclusions, we ended up editing an example file, which utilizes a remote API, written directly in the V-rep source folder, which is why the file retains its former name, "bubbleRobClient". Finally, by using Cmake to build a makefile, our program is compiled directly by calling the file and specifying the port number, to which V-rep has connected to, while it is running. Figure 8.7 and figure 8.8, shown below, demonstrate an example of a successful remote API communication thread between V-rep and VS Code:



```
isaac@LAPTOP-FG1U83P0:/mnt/c/Program Files/CoppeliaRobotics/CoppeliaSimEdu/programming/bubbleRobClient$
 cmake --build .
[100%] Built target bubbleRobClient_remoteApi
isaac@LAPTOP-FG1U83P0:/mnt/c/Program Files/CoppeliaRobotics/CoppeliaSimEdu/programming/bubbleRobClient$
 ./bubbleRobClient_remoteApi 19999
start
Connecting using port. 19999..
Connection was killed...
```

Fig. 8.7. Code Snippet Taken from an Ubuntu Terminal Showing Compilation of Code Written in VS Code and a Connection to the Port, 19999.



```
function sysCall_init()
   res=simRemoteApi.start(19999,1300,true)
   if(res== -1) then
     print('Error failed to connect')
   else
     print('Successful connenction:port[19999]')
   end
```

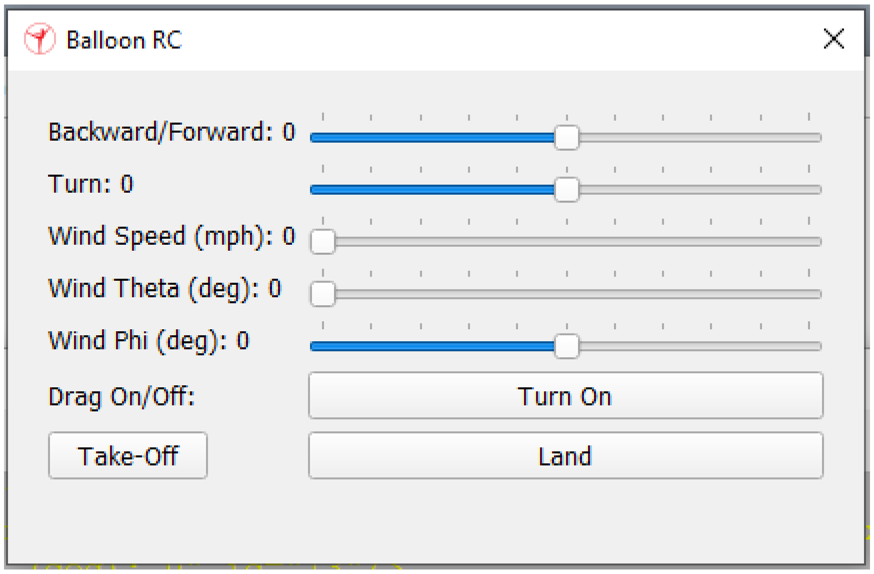
```
[sandboxScript:info]   Simulation started.
Successful connenction:port[19999]
[sandboxScript:info]   simulation stopping...
```

Fig. 8.8. Code Snippet and Terminal Output Shown in Lua, V-rep's Embedded Scripting Language, Managing the Connection to Port 19999.

8.5.2 Direct Interface of the Closed Loop Remote Control Function

For the purposes of interfacing V-rep with a closed loop remote control system, a fundamental understanding of the closed loop RC is necessary to be established.

The closed loop function, described in further detail in Chapter 6, takes in the four inputs: forward; turn; maximum force of propellers; and an array of six variables listed as pitch, roll, height and their corresponding derivatives, alternatively known as "PRH states". And the output of the closed loop RC function provides the necessary angles and throttles of each individual propeller for the drone to operate using.



Balloon RC

Backward/Forward: 0

Turn: 0

Wind Speed (mph): 0

Wind Theta (deg): 0

Wind Phi (deg): 0

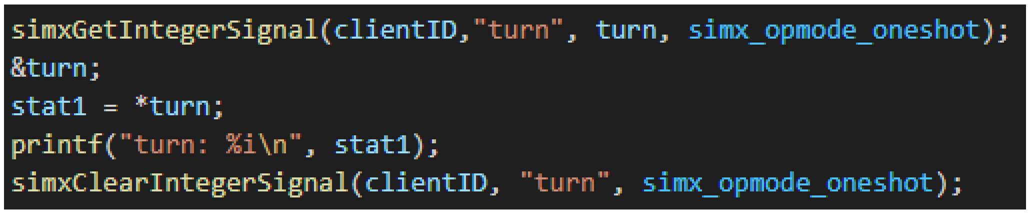
Drag On/Off: Turn On

Take-Off

Land

Fig. 8.9. Controller Input Layout of a Closed Loop Remote Control in GUI Format is Shown Above.

A completed closed loop remote control removes the need for a propeller speed variable, as seen in figure 8.9 which shows the current implementation of the GUI in V-rep. This system allows the drone to move with only two commands, forward throttle and turn factor, and corrects for external factors that cause the drone to tilt at unspecified angles.



```
simxGetIntegerSignal(clientID,"turn", turn, simx_opmode_oneshot);
&turn;
stat1 = *turn;
printf("turn: %i\n", stat1);
simxClearIntegerSignal(clientID, "turn", simx_opmode_oneshot);
```

Fig. 8.10. Remote API Functions are Written to Pass Data Between Server and Client.

In figure 8.10, a single turn command is being retrieved from V-rep with a non-blocking function call, allowing data to be sent without waiting for a reply.[55] These remote API function calls are essentially the fundamental building blocks of transferring data between server and client programs.

Although the theory behind developing a closed loop remote control system was generally understood, performing these function calls required debugging the errors we found in much greater detail. So it is at this stage of our design where the practical simulation engineering section concludes and the beginning of our hypothetical tests as well as analysis of our overall simulation design begins.

8.5.3 Hypothetical Testing of Closed Loop Remote Control

If the implementation of closed loop RC were to be completed, STR 3.0.0, Remote Control, would be verified in this section.

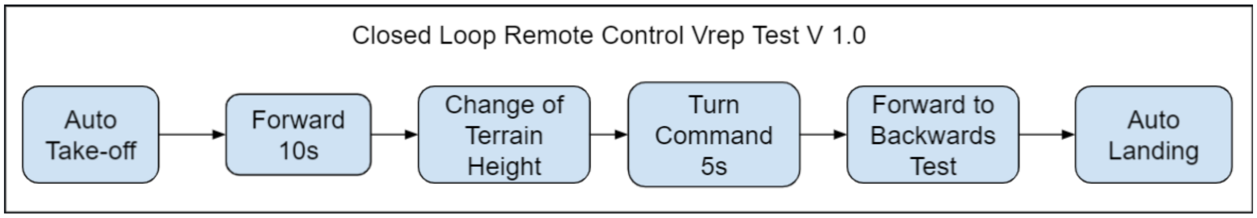


Fig. 8.11. The Block Diagram Above Shows How a Test Would be Conducted Using a Closed Loop Control System.

Firstly, to take the drone off the ground, an automatic take-off sequence is engaged with the press of the Take-off feature on the GUI. This feature causes the propellers to start up and increase thrust to maximum. Then, the forward throttle is increased to maximum; much like open loop remote control, the propellers face in the 'forward' direction and move so, adjusting in mid-air for any slight misdirections, caused by any uneven distributions of mass, thus torque. A change of terrain height is then presented to the drone and, with the barometer and ultrasonic sensors constantly feeding back data of height and altitude, an increase in throttle upward allows the drone to be able to avoid all obstacles that may potentially collide with the drone. After clearing a provided terrain height, the drone is issued a turn command for five seconds to test a range of narrow to wider turn radiuses. Next, we have the forward to backwards test. This test assesses a drone's RC response to a significant amount of unforeseen pitch and pitch velocity, by switching the throttle back and forth in quick succession, to account for collisions, heavier than normal winds, or simple piloting errors. Finally at the end of an experiment, the autonomous landing feature is engaged with the press of the button on the GUI. The propellers, given the data of height and vertical velocity, correctly adjust propeller speeds to zero before the drone touches the ground, so as to prevent any high speed collisions with expensive hardware.

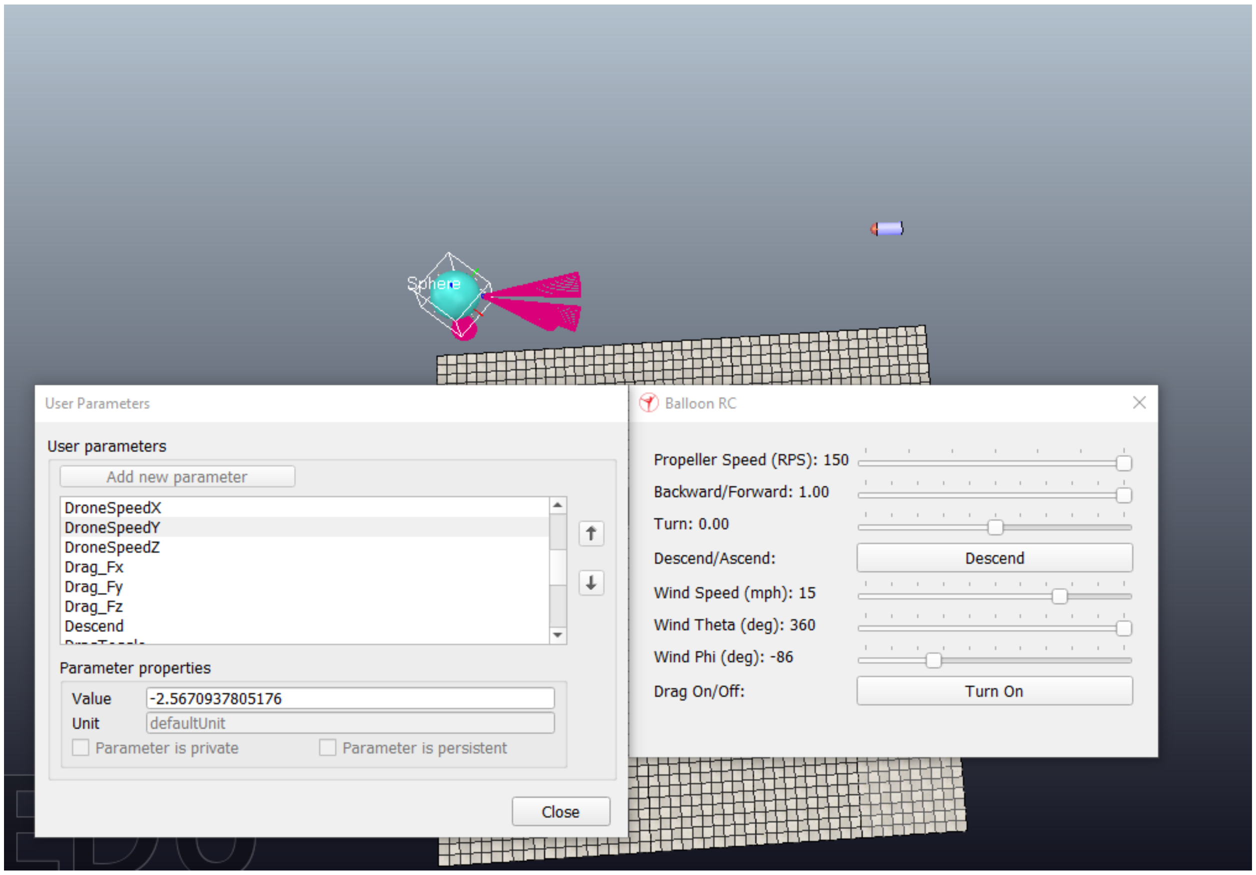
8.5.4 Further insights into the Closed Loop Remote Control Implementation

Due to our limited programming knowledge in C++, inexperience with communication networks, or in the absence of a fully developed or updated V-rep user manual for remote API[53], closed loop remote control implementation was unable to be completed on time. Given more time to work with interfacing V-rep with VS Code, closed loop remote control may have been completely completed and tested. However, for reference, when seeking help from our TA, Alexey Munishkin, who has had previous experience with remote application programming in V-rep, interfacing with Matlab, he had told us that learning remote API in V-rep had taken him close to a year to do. In retrospect, Gazebo, an alternative robotic operating system we had in mind when deciding on a simulation environment, may have been a better option for operating our drone model due to its support of aerodynamic principles, specifically buoyancy and fluid density. The primary justifications we had in mind when choosing V-rep over Gazebo was modularity and the fact that Gazebo could not operate under a Windows operating system and was required to be installed through Ubuntu packages on Linux. The irony of this decision lies in the fact that in order for our client program(VSC) to compile C++, Ubuntu had to be installed anyway.

8.6 Verification of the Prototype's Fabricated Dimensions

In this section the attempted verification of the following STR will be discussed:
2.0.0 Drone shall be able to fly at least 5mph in winds up to 15mph.

Based on the newest fabricated model, discussed in further detail in Chapter 3, V-rep simulations were run to test potential flight speeds we would expect simply based on the model's dimensions.



User parameters

DroneSpeedX
DroneSpeedY
DroneSpeedZ
Drag_Fx
Drag_Fy
Drag_Fz
Descend

Parameter properties

Value	-2.5670937805176
Unit	defaultUnit

Parameter is private   Parameter is persistent

Propeller Speed (RPS): 150
Backward/Forward: 1.00
Turn: 0.00
Descend/Ascend: Descend
Wind Speed (mph): 15
Wind Theta (deg): 360
Wind Phi (deg): -86
Drag On/Off: Turn On

Fig. 8.12. Drone Speed is Shown to be -2.567 m/s on the Left Column Reading 'User Parameters'

With the throttle and wind speeds set to maximum, the fabricated V7 envelope is calculated to fly with 40N drag in a 15 MPH head wind. Simply stated, the drone cannot move forward in the x-y plane against the maximum headwind at maximum throttle. As shown in figure 8.12, our drone is unable to fly within 15 MPH of headwind. Although horizontal drag had immensely overshot our expectations, vertical drag was significantly decreased, which would mean vertical flight testing was still within reasonable expectations to be achieved in physical testing.

8.7 Conclusion

Overall, the simulation discussed in this chapter provided results that we did not fully expect to get. At the same time, we were not able to see the conclusion of all the tests we expected to complete, specifically those of closed loop remote control and autonomous. However, we were able to verify STR 2.0.0, Drone Speed, in simulation tests that we were not able to retrieve in physical testing such as drone speed and controllability. Due to the failure to implement remote API, STR 3.0.0, Remote Control could not be verified.

Chapter 9: Testing in a Controlled Environment

Testing in a controlled environment was unable to be completed due to errors in the handling of the drone and equipment. Two flight tests were attempted but both of them failed before flight time data could be gained. Further tests could not be done due to shipping delays when ordering new lift bags. Procedure for completed, attempted, and planned controlled environment testing are introduced and discussed in this chapter. STR 8.0.0, Helium Leakage, STR 10.0.0, Noise Level, and STR 1.2.0, Effective Weight, all tested but failed to meet their requirements. STR 1.0.0, was not able to be verified due to flight test failures.

9.1 Tests to Conduct

In this section we will introduce each of the tests we planned to perform in a controlled environment. This included noise testing, helium loss testing, system weight testing, and flight time testing. It also introduces additional tests that would have been performed in a controlled environment if the control system had been completed and verified in VREP.

9.1.1 Conducted Tests

The first test that was planned to be conducted was a noise level test. This was in order to determine if the system met STR 10.0.0, Noise level, this test would take measurements of the dB level at 5 feet away around the drone in order to determine if the noise level was under 65dB at max throttle.

The next test we planned to conduct was the system weight test to determine if our system met STR 1.2.0, Weight, this test would simply measure the weight of the system when fully inflated with helium in order to determine if the effective weight was within the required 0-5N range.

A helium loss test was to be performed In order to determine if our system met STR 8.0.0, Helium Loss. This test would measure the weight of the system when filled with helium at several time intervals in order to determine if the loss rate of the helium would be less than 10% within a week.

The final test to be performed was a flight time hover test. This was in order to test if our system would meet our STR 1.0.0, Flight Time, stretch goal. This test would have the drone system hover until the voltage alarm sounded, determining the flight time of our system when hovering. This would determine if our power draw analysis seen in Chapter 7 was accurate, as well as determine if our system met the 1 hour hover time stretch goal. This test could not be completed due to fabrication errors. For more information about the fabrication errors see section 3.3.3.

9.1.2 Unconducted Tests

Some additional tests that were planned to be performed but were not able to be done due to parts of the system being incomplete.

First a flight time test at maximum motor throttle to determine if the STR 1.0.0, Flight Time, primary goal of 30 minute flight time would be reached while satisfying STR 2.0.0, Drone Speed. This would function similarly to the hover test except a fan would be used to generate a 20mpg wind, until the voltage alarm sounded. This test would have our system "flying" with a 20 mph airspeed. If the system can maintain its position for 30 minutes, both requirements are met.

A closed loop control flight test would be conducted in order to determine if STR 3.0.0, Remote Control, was met. This would have our system fly using our closed loop control system, it would determine if our system was successfully able to maintain stable pitch and roll angles within ±0.1 radians, while maintaining a stable height of 1±0.15m. additional test could be conducted in wind in order to test its stability in harsher conditions.

An Autonomous controlled flight test would determine if STR 4.0.0, Autonomous Control, was met. This would test the miscellaneous auto take off and landing functions, as well as its ability to fly between waypoints. Additional tests could be conducted in wind and with obstacles to test its stability and collision avoidance.

A Magnetometer interference test would determine if our STR 6.0.0, Interference, was met. This test would take data with the magnetometer with the drone off, on and in various stages of flight in order to determine if the interference created by our system at any stage become greater than 10nT.

9.2 Procedures and Results

This section will talk about the procedures of both conducted and proposed tests. For the conducted tests the errors and adjustments that were made will be discussed. The results will also be introduced and analysed.

9.2.1 Noise Testing

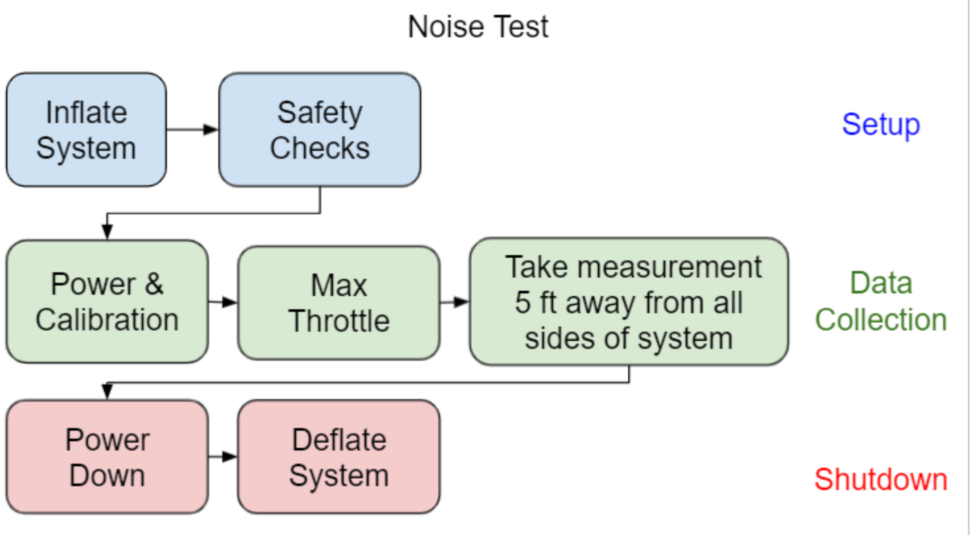


Fig. 9.1. Procedure of Noise Test

The first Controlled environment test that was conducted was the noise test. This test was to determine if the noise level at 5 feet away from the drone was less than 65dB as required by STR10.0.0, Noise level This test was done in the week of 5/17/21 by Dylan Harutoonian, Jeremy Germenis, and Leonid Shuster inside Leonid's garage. A high level procedure can be seen in Figure 9.1.

Noise level test detailed procedure:

1. Layout envelope so that all motor brackets are on the outskirts and the envelope is flat on that ground, leaving the gondola bracket in the middle of the envelope covered by the top
2. Inflate the system entirely with air until the motor brackets are held taught
3. Perform safety checks
    a. Assure that all motor and signal wires are attached in the gondola and to the motors
    b. Ensure that motor wires will not be hit by the propellers by using zip ties and electric tape to secure the wires to the brackets and envelope.
4. Perform the calibration of motors
    a. If ESCs are uncalibrated
        i. Set the throttle curve on controller to max at 100%
        ii. Push the throttle to 100% with the system off
        iii. Turn the system on
        iv. Wait for 2 beeps followed by third beep from each of the motors
        v. Turn the throttle to 0% within 2 seconds
        vi. Wait for 3 more beeps from the motors

vii. Set throttle curve to max allowed throttle at 29% with replacement motors

b. If ESCs are calibrated

i. Turn the power on

5. Run the motors increasing the throttle 1% every 2 seconds until 29% is reached
6. Have a member stand 5 feet away from the drone, measured with a 5ft tape measure, and observe the decibel meter on an apple watch while slowly walking along the radius of 5 feet away from the drone
7. Record the highest decibel seen on the watch
8. Turn the motors to zero throttle and turn the power switch off
9. Deflate the system

These procedures were followed and several iterations of the test were done, Figure 9.2 shows one of these tests being conducted. The data in these tests were collected using the on board noise level meter in a Series 6 Apple Watch, according to third party testing the on board noise level meter has an accuracy of 1%[56].

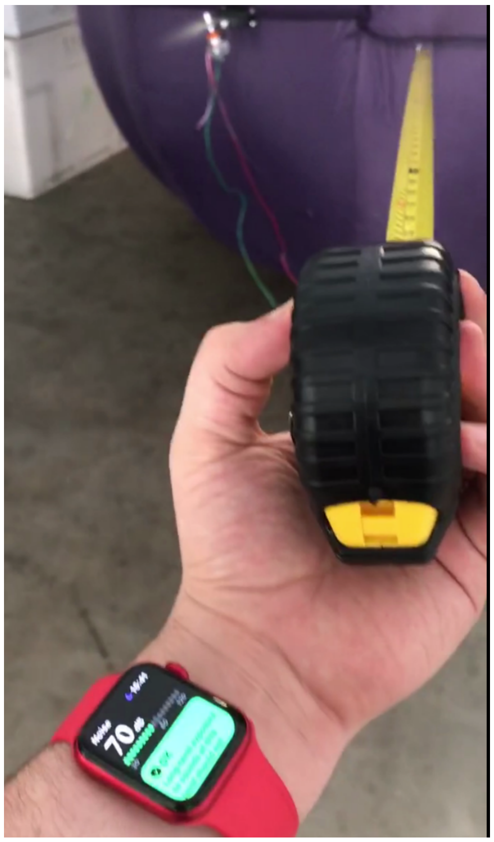


Fig. 9.2. Noise Test Being Conducted.

The test was conducted first with the garage door closed. The result of the test was that the highest decibel rating had a peak of 76dB at the highest motor throttle. However, the test was done again with the garage door open due to the possibility of constructive noise interference. With the garage door open, the highest decibel rating had a peak of 72dB at the highest motor throttle. Both of these tests were conducted several times with consistent results. This verifies that we did not meet STR 10.0.0, Noise Level, which stated the drone should be quieter than 65dB. However, we wish to conduct the same test in an outdoor environment, since the noise level is lower with the garage open it is expected to be even lower when the system is fully outside. This is important as the system is intended to be operated primarily outside once complete. We believe that a noise test outside could meet STR10.0.0, Noise Level, but the test would have to be conducted in order to confirm this. Additionally, it can be noted that the noise level was below 85dB, which is the noise level at which chronic hearing damage starts to occur[76]. Although this does not impact STR 10.0.0, Noise Level, this does have an effect on STR 7.0.0, Drone Safety. For more information about Drone Safety, see Chapter 11.

9.2.2 Helium Loss and Weight testing

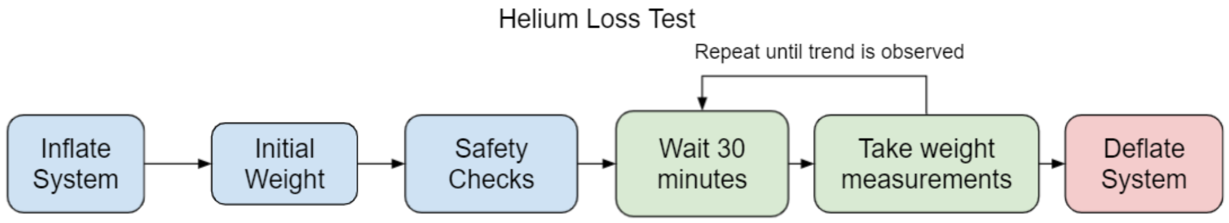


Fig. 9.3. Procedures of Helium Loss Test.

The next controlled environment test was helium loss and system weight. These were conducted at Westside Research Park under supervision of graduate student Gordon Keller. These tests were performed by Dylan Harutoonian and Jeremy Germenis during the week of 5/24/21. The tests were conducted simultaneously with the attempted Flight Time test discussed in section 9.2.3. The goal of the weight test was to verify STR 1.2.0, Weight, while the goal of the helium loss test was to verify STR 8.0.0, Helium Loss First, the drone system had to be transported to the location with the motors and gondola that will be used to hover. A helium tank then had to be bought and delivered from Praxair in Watsonville, CA to the testing room with the safety considerations detailed in Chapter 11. This involved securing the helium tank to a metal fastening in the room and ensuring it was locked with a chain. The testing at Westside Research Park was then approved to be conducted. Before inflating the system the weight of all components parts in the tests were weighed. This included a 1kg weight function as the payload for the purposes of this test. The initial weight without helium was estimated to be 4.444kg. After this the test was ready to start. The high level procedure can be seen in Figure 9.3.

Helium Loss Test Detailed Procedure:

1. Layout envelope so that all brackets are on the outskirts.
2. Inflation
    a. Since the envelope is large then anticipated first the lift bag will be inflated until it is sphere with 1.5m diameter or 1.8m^3 in volume (for more information on the envelope fabrication error see Chapter 3.)
    b. The rest of of the system will be filled with 4m^3 of helium
3. Check weight of system
4. Safety Check
    a. Assure that all attachments are securely
    b. Attach tether to weight on ground
5. Wait for 30 minutes
6. Check weight of system
7. Repeat steps 5 & 6 several times to see if weight loss if linear or exponential
8. Deflate system
9. Find weight difference over time to estimate helium loss for 1 week

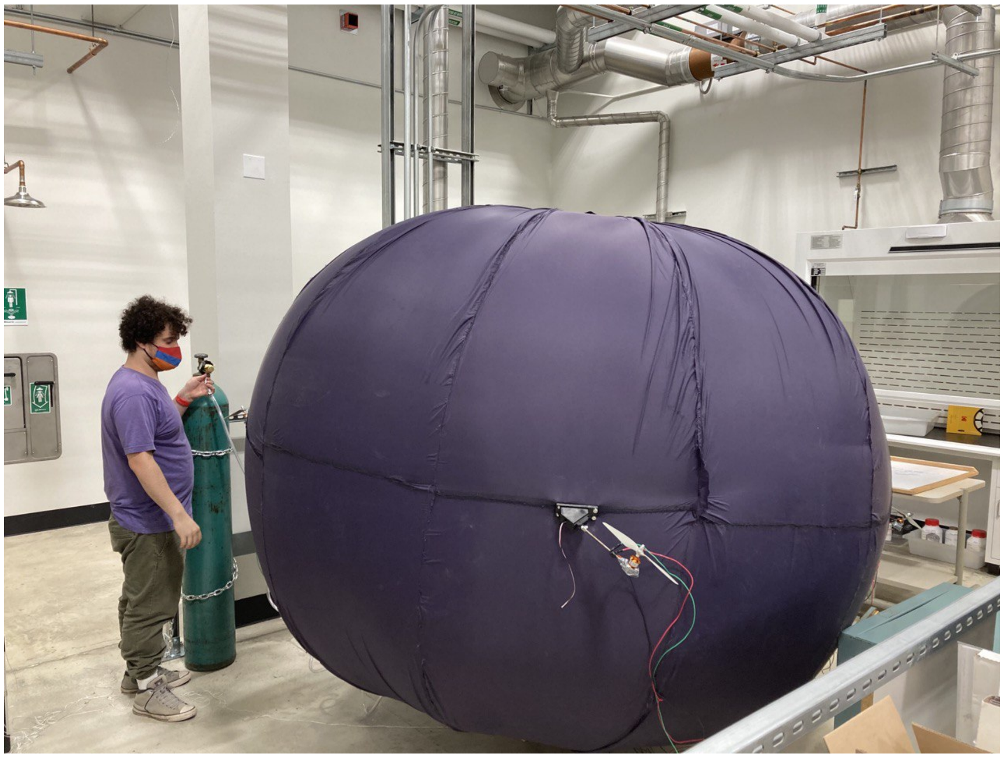


Fig. 9.4. Full System Inflated with Helium

Once the System was fully inflated as seen in Figure 9.4 the system was weighed to be 1.3kg.

This was higher than expected and does not meet STR 1.2.0, Weight, this was likely due to too much air being added. The air added additional weight to the system that the helium did not compensate for. Unfortunately the system was punctured during transport to do the hover test discussed in section 9.2.3, so helium loss

A second attempt was made at this test. In this test less air and more helium was to be added, however during the second attempt of the inflation test the inflation hose disconnected from the lift bag. Although the hose was eventually reattached significant helium was lost during the reconnection process. Due to this helium loss we ran out of helium before the system was fully inflated. Additionally when the hose was reattached it was moved inside the envelope and the lift bag was not able to be properly sealed once inflation was finished. However, the data was collected anyway. This time the initial weight of the system was 798g, closer to system STR 1.2.0, Weight, but still not meeting the required <500g. However it is believed that if we had not run out of helium due to the leak during inflation the requirement likely would have been met. 30 minutes later a second data point was taken showing that the system now weighed 960g, As shown in table 9.1 this estimated a loss of 4.2% of the system's helium, meaning that the system would lose 10% of its buoyancy in just over an hour. This verified that we would not meet STR 8.0.0, Helium Loss. The system was once again punctured before any more data could be collected.

Table 9.1

Data Collected in Second Helium Loss Test

Time	Weight	Estimated helium in system	Percentage of helium lost
0min	798g	4.50m^3	0%
30min	960g	4.356m^3	4.2%

9.2.3 Hovering Flight Time Testing

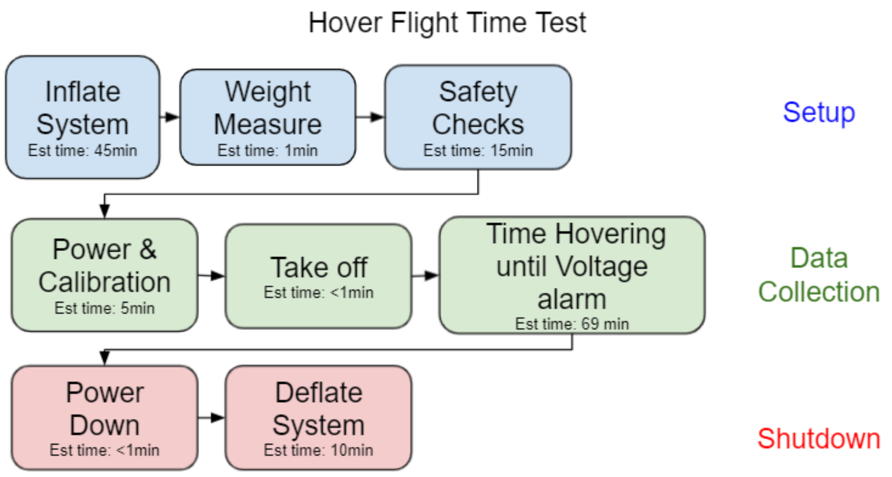


Fig. 9.5. Flowchart of the Hover Test to be Performed in the Westside Research Park Flight Room

The Hovering Flight time test in a controlled environment was done simultaneously, to the weight and helium lost test discussed in section 9.2.2. This test aimed to determine if our system would meet the 1 hour hover flight time stretch goal from STR 1.0.0, Flight Time. The high-level procedure can be seen in Figure 9.5.

Hover Test detailed procedure:

1. Ensure battery is fully charged with 4.2V per cell and 12.6V total
2. Layout envelope so that all motor brackets are on the outskirts and the envelope is flat on that ground, leaving the gondola bracket in the middle of the envelope covered by the top
3. Inflate the system
    a. Since envelope is larger than anticipated first the lift bag will be inflated until it is sphere with 1.5m diameter or 1.8m^3 in volume
    b. The rest of of the system will be filled with 4m^3 of helium or until lift bag is taught enough so the motor brackets are unable to hit the envelope
4. Check the weight of the inflated system and record the weight in kilograms
5. Perform safety checks
    a. Assure that all motor and signal wires are attached in the gondola and to the motors
        i. Ensure that motor wires will not be hit by the propellers by using zip ties and electric tape to secure the wires to the brackets and envelope.
    b. Attach 2 tether strings to a larger than 10kg weight on ground
6. Perform the calibration of motors
    a. If ESCs are uncalibrated
        i. Set the throttle curve on controller to max at 100%
        ii. Push the throttle to 100% with the system off

iii. Turn the system on
iv. Wait for 2 beeps followed by third beep from each of the motors
v. Turn the throttle to 0% within 2 seconds
vi. Wait for 3 more beeps from the motors
vii. Set throttle curve to max allowed throttle at 29% with replacement motors

b. If ESCs are calibrated
   i. Turn the power on

7. Perform the flight test
   a. Ensure the servos are turned to upright position
   b. Turn throttle to between 16% and 29% to get system off the ground
   c. Reduce the throttle to 16% for hovering
   d. Keep hovering until voltage alarm sounds
   e. Reduce the throttle to 0%
8. Turn off the power switch
9. Record the amount of time the drone was in the air
10. Deflate the system

The hover test was attempted twice but failed to achieve lift off either time due to mishandling of the equipment.

As stated in section 9.2.2, during the first attempt the system was successfully fully inflated until taut. After this, the safety checks for wiring, attached parts and tether connections were completed. However, when trying to transfer the system from the inflation room into the flight room, the system was punctured due to communication error. Therefore the system was not able to attempt flight during the first test.

The second attempt was done inside the flight room to avoid another transportation accident. We didn't inflate it there the first time because we were unsure if we could move the helium tank from where it was stored. After we reserved approval to move the helium tank inside the flight room, we started inflating the drone in the netted testing room. However, as stated in section 9.2.2, during inflation, the lift bag entrance was slowly moved up the side of the envelope during inflation. This meant the tube entrance being used to insert the helium was unreachable. After trying to move the lift bag, the tube was taken out of the lift bag and helium leaked as we tried to keep the lift bag closed and move the entrance back towards the opening for the gondola. This loss of helium meant the second inflation was not full enough and the propulsion system mounts were not taut as a result. The significant slack on the propulsion system can be seen in Figure 9.6.

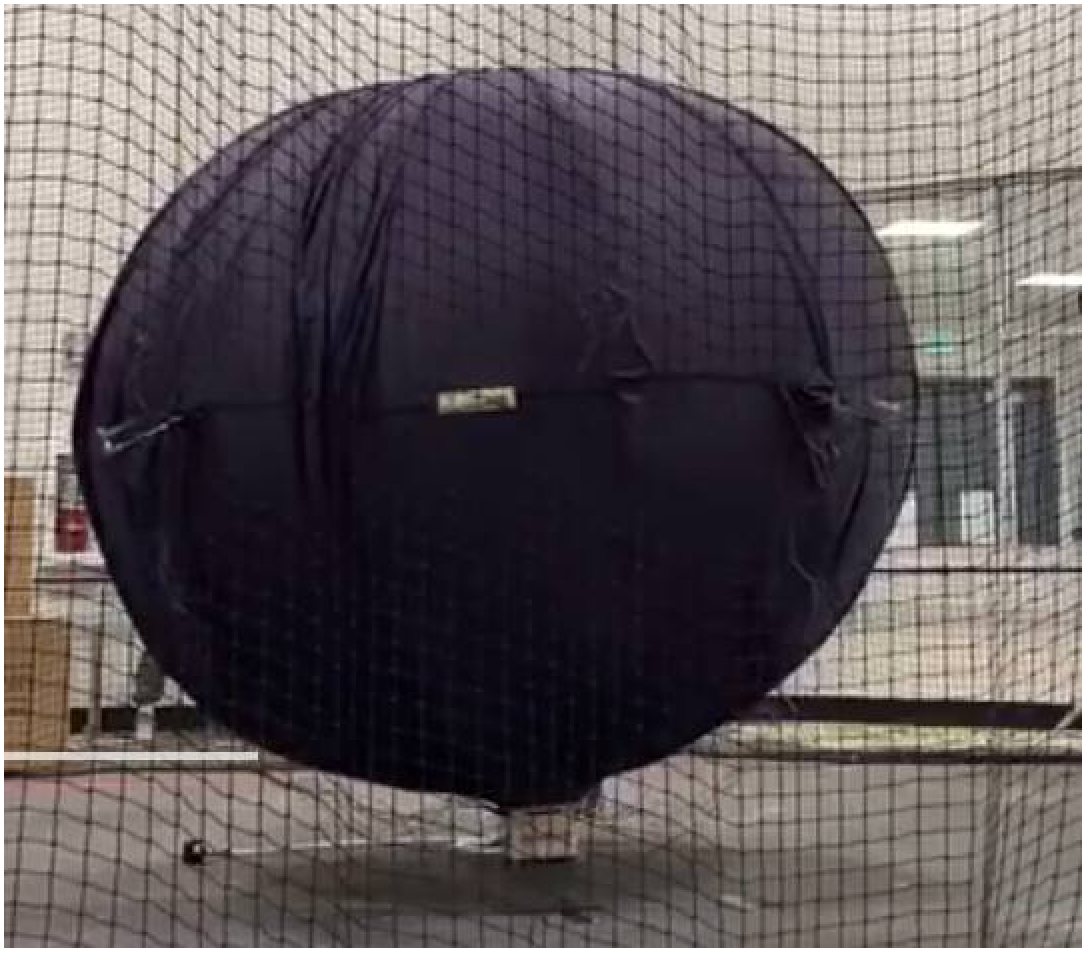


Fig. 9.6. Inflated System with Slack on Hover Test Attempt 2

When we saw that the slack in the system we considered our options. The first option we considered was trying to add more air to the system to make it taut. The benefits of this would be that the motors would be less likely to hit the system due to the slacked envelope not holding them properly outright. However we decided against this as the risk of popping the balloon and not getting any data during this attempted inflation was too great. So after the system was properly tethered, the motors were then turned on and calibrated. However, when approaching 10% throttle, one of the propellers started hitting the envelope. After several attempts to remove the slack from the envelope the test was eventually aborted due to this. We decided to try our first option of adding more air to the lift bag to remove the slack in the envelope but due to the friction of the inflated lift bag inside the envelope the lift bag ruptured and the test was not able to be continued. Therefore, STR 1.0.0, Flight Time, was not verified by thes flight tests.

Although this test was a failure the tests did help us verify our helium safety and inflation techniques. If the helium leak had not caused slack in the envelope the system would have likely been able to hover, however the time of hovering cannot be determined from these results. Power testing, as described in Chapter 7, remained our best flight time estimate while we expected the hover time of the drone to be 69.6 minutes. Another attempt of this test is planned in the future once a new helium lift bag is received.

9.3 Unconducted Test Procedures

The proposed procedures for the unconducted tests are shown here. Since these tests were not conducted the errors and results during these tests cannot be discussed.

The procedure for our flight speed test with controlled wind to determine if our system meets STR 2.0.0, Drone Speed, and STR 1.0.0, Flight Time shown in Figure 9.7

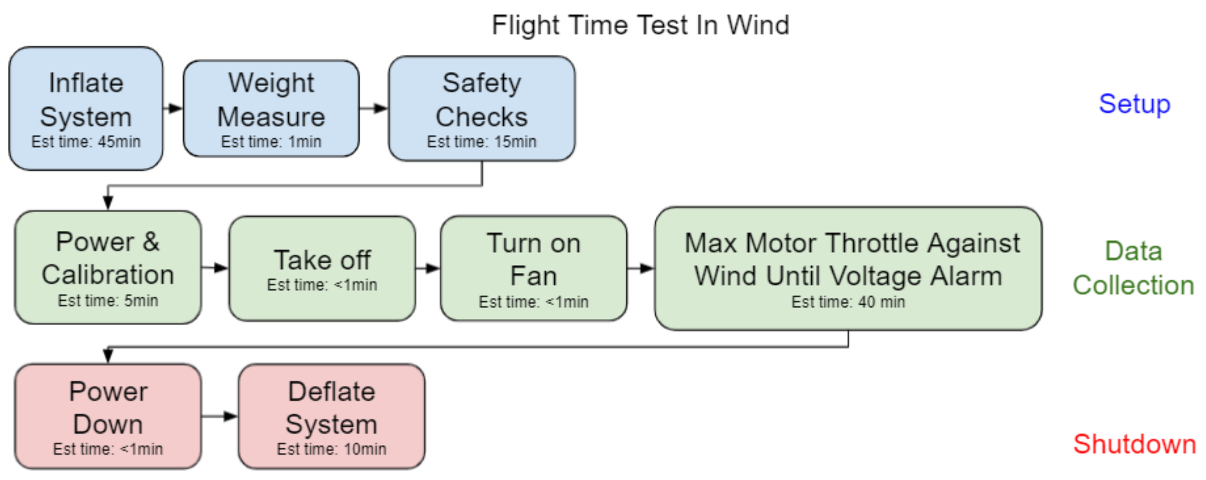

Flight Time Test In Wind
Inflate System Est time: 45min
Weight Measure Est time: 1min
Safety Checks Est time: 15min
Setup
Power & Calibration Est time: 5min
Take off Est time: <1min
Turn on Fan Est time: <1min
Max Motor Throttle Against Wind Until Voltage Alarm Est time: 40 min
Data Collection
Power Down Est time: <1min
Deflate System Est time: 10min
Shutdown

Fig. 9.7. Wind Speed Test

The procedure for the closed Loop control flight test the would be conducted in order to determine if STR 3.0.0, Remote Control, was met is shown in Figure 9.8

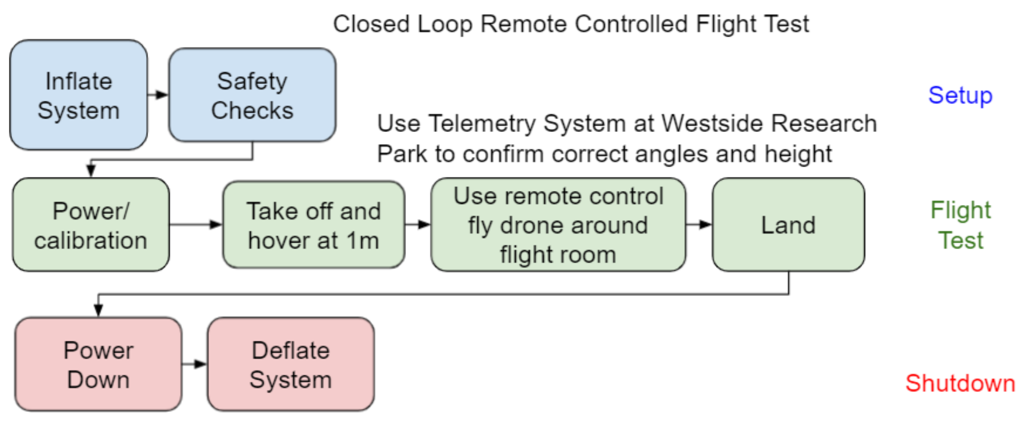

Closed Loop Remote Controlled Flight Test
Inflate System
Safety Checks
Setup
Use Telemetry System at Westside Research Park to confirm correct angles and height
Power/ calibration
Take off and hover at 1m
Use remote control fly drone around flight room
Land
Flight Test
Power Down
Deflate System
Shutdown

Fig. 9.8. Closed Loop Remote Controlled Flight Test

The procedures for the autonomous controlled flight test that would determine if the STR 4.0.0, Autonomous Control, was met iss shown in Figure 9.9

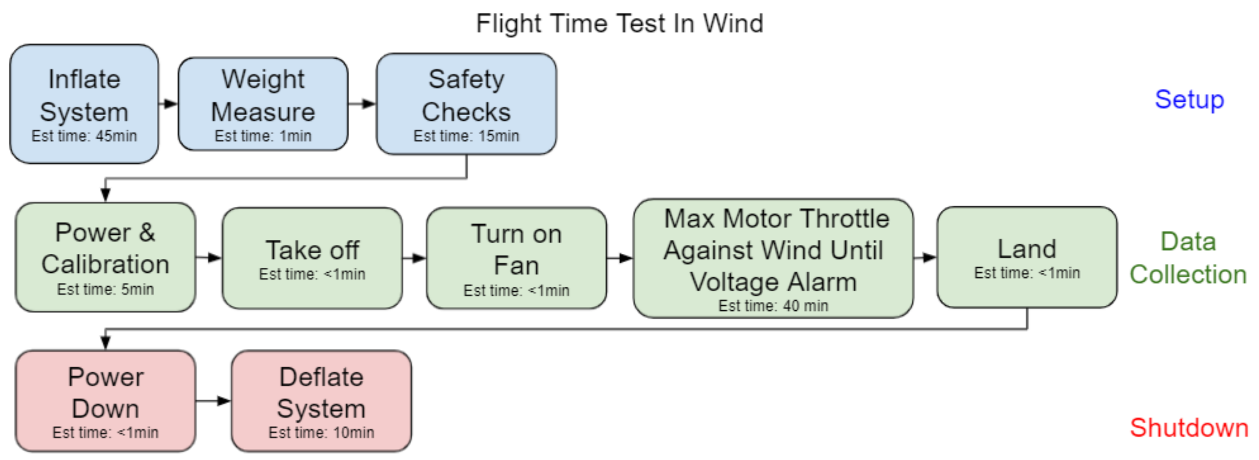

Flight Time Test In Wind
Inflate System Est time: 45min
Weight Measure Est time: 1min
Safety Checks Est time: 15min
Setup
Power & Calibration Est time: 5min
Take off Est time: <1min
Turn on Fan Est time: <1min
Max Motor Throttle Against Wind Until Voltage Alarm Est time: 40 min
Land Est time: <1min
Data Collection
Power Down Est time: <1min
Deflate System Est time: 10min
Shutdown

Fig. 9.9. Autonomous Flight Test

Finally the procedures for the magnetometer interference test that would determine if our STR 6.0.0, Interference, was met is shown in Figure 9.10

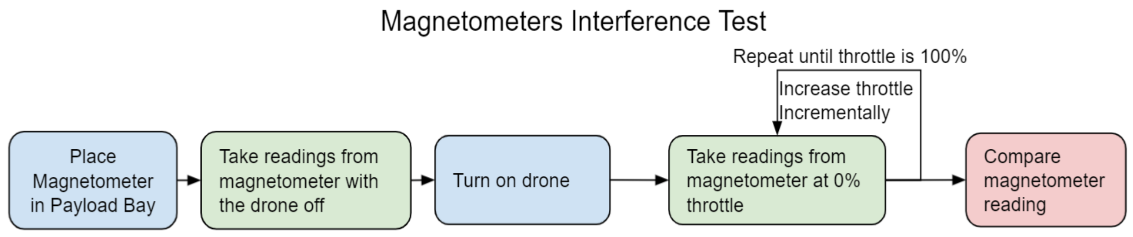

Magnetometers Interference Test
Repeat until throttle is 100%
Increase throttle Incrementally
Place Magnetometer in Payload Bay
Take readings from magnetometer with the drone off
Turn on drone
Take readings from magnetometer at 0% throttle
Compare magnetometer reading

Fig. 9.10. Flow chart of Magnetometer Test procedures

Although these tests have not been conducted we believe that these procedures would effectively verify the requirements once the system is ready for the tests.

9.4 Conclusion

Through the controlled tests conducted we were able to verify that our drone did not meet several requirements. First we found that the STR 10.0.0, Noise Level, was not met in an indoor environment. We found that STR 1.2.0, Weight, was not met but likely may have been if the system was fully inflated. We also found that STR 8.0.0, Helium Loss, was not met. Lastly due to the failure of the flight testing, STR 1.0.0, Flight Time, was not able to be verified. Additionally we designed procedures for testing for STR 2.0.0, Drone Speed, STR 3.0.0, Remote Control, and STR 6.0.0, Interference.

Chapter 10: Testing in an Uncontrolled Environment

Testing in a variable environment was unable to be conducted due the incomplete state of the system. We wanted to fully test the system in a controlled environment in order to make sure we had complete control and verification of many of our systems functions before moving on to a variable environment. Since the controlled environment testing was not completed testing in an variable environment was never attempted, however This chapter will go discuss the tests and procedures we would have used to verify our systems use in a variable environment.

10.1 Variable Environment Tests

In this section we will introduce the proposed tests that we would have attempted in order to verify our systems use in a variable environment, namly outside. These tests would mostly be focused around stress testing our closed loop R.C. and Autonomous functionality and their ability to verify their respective technical requirements STR 3.0.0, Remote Control, & STR 4.0.0, Autonomous. Additionally these tests would have helped confirm that STR1.0.0, Flight Time, and STR 2.0.0, Drone Speed, are achievable in an uncontrolled environment.

10.1.1 Closed Loop R.C. Testing

Once verified in a controlled environment such as the flight room at Westside Research Park, A closed loop control flight test would be conducted outside in order to determine if STR 3.0.0, Remote Control, could be met when we have no control over the weather or obstacles that the system may encounter. This would have our system fly using our closed loop control system, it would determine if our system was successfully able to maintain stable pitch and rolla angles within ±0.1 radians, while maintaining a stable height of 1±0.15m. While dealing with the natural terrain and wind patterns of the area, the test is conducted. The uncontrolled wind conditions would help us further verify the STR 2.0.0, Drone Speed, as the wind direction would vary and test conditions that we had not tested in our controlled test. Additionally the closed loop control may respond in unique ways to the uncontrolled wind and terrain. This could vary the power draw of our system in unforeseen ways, so this test would further help STR 1.0.0, Flight Time. Since this test will be conducted outside the length of the tether used to hold the drone would be increased, so that drone could maneuver over a large area while still maintaining the security against the loss of the system a tether provides. The procedure of this potential test is shown in Figure 10.1.

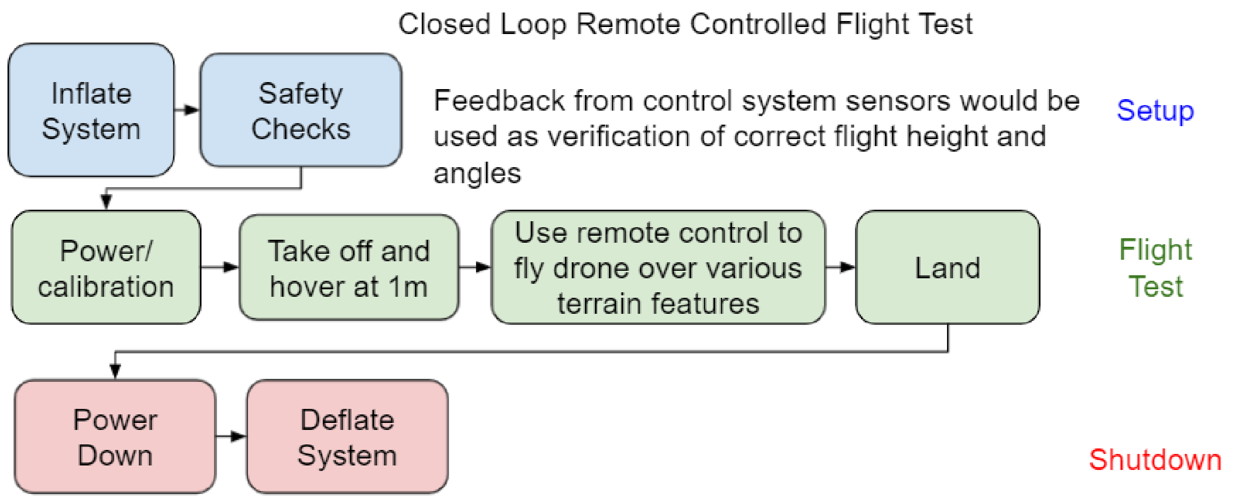

Closed Loop Remote Controlled Flight Test
Inflate System
Safety Checks
Feedback from control system sensors would be used as verification of correct flight height and angles
Setup
Power/ calibration
Take off and hover at 1m
Use remote control to fly drone over various terrain features
Land
Flight Test
Power Down
Deflate System
Shutdown

Fig. 10.1. Procedure Flow Chart for Closed Loop RC Flight Test

Since the telemetry system available at Westside Research park will not be available for this test, Subsystem requirements STR3.1.0, System Response, and STR 3.2.0, Data Feedback, must be fully verified in order to maintain safe and controlled flight as well as collect valid from the control system sensors as these will provide us with our systems telemetry data allowing us to verify the success or failure of the test flight.

10.1.2 Autonomous Test

After the closed loop control of the system is verified in an uncontrolled environment, a similar test can be performed with our drones system on autonomous control. In addition to the closed loop remote control flight test confirming the ability of our system to fly in an uncontrolled environment, it is critical that switch from autonomous to RC STD3.3.0 is fully verified, as in the event of the loss of control of our system, it is essential that we regain control of system as soon as possible. Similarly to the closed loop remote control test this test will help verify both STR 1.0.0, Flight Time, as well as STR 2.0.0, Drone Speed, by allowing us to see how our autonomous flight system responds to uncontrolled environmental obstacles, terrain, and wind. We can determine if these requirements are still met even in the uncontrolled environment that may cause our drone to respond in ways not found in controlled testing. This test will also be used to further verify STR 4.0.0, Autonomous by seeing that our drones required telemetry holds even in uncontrolled conditions. For this test to be conducted STR 4.4.0, Data Feedback, must be verified before hand, as similar to the closed loop control test the telemetry system at Westside Research park will not be available, so receiving data feedback about the systems telemetry during the flight test is essential to the verification of if the test was success or failure. The procedures of this proposed test can be seen in Figure 10.2

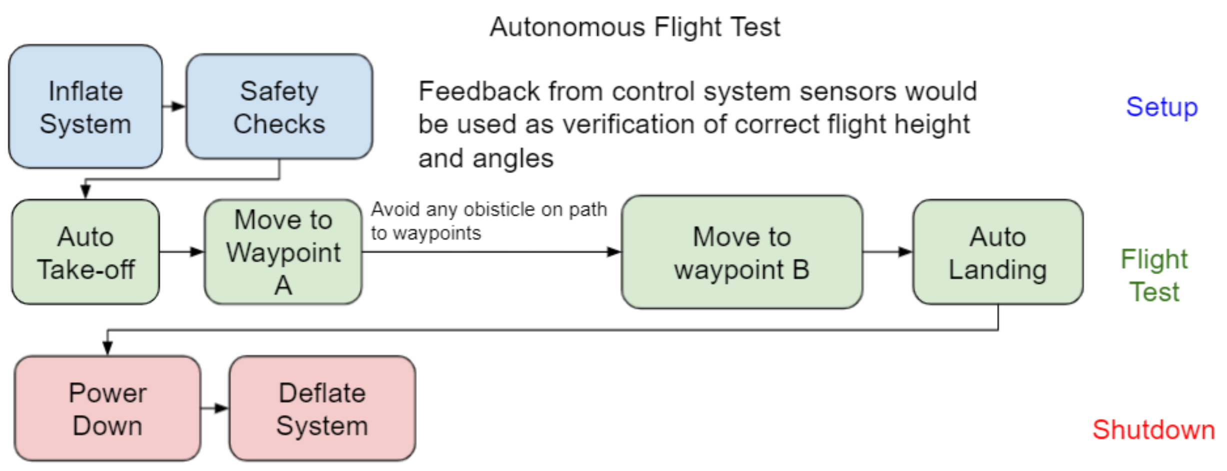


Fig. 10.2. Autonomous Flight Test

10.2 Conclusion

Although these tests were not conducted within the duration of our project, The verification methods of our requirements were planned out, as well as the steps needed to be taken before the verification process could start. This is important as it gives a clear path of how work on this project could be continued.

Chapter 11: Legal and Safety Requirements

A number of legal and safety considerations were made for the system. STR 9.0.0, Legal Compliance, required legal actions that were investigated but could not be verified, since the drone could not be registered with the Federal Aviation Administration(FAA). However, several different laws and regulations were studied and confirmed that our drone would follow the requirements. A number of safety precautions were also taken to ensure the safe flying of the drone and the safety of those around it, including collision, electrical, and helium safety to fulfill STR 7.0.0. These safety implementations all fulfill one or more requirements of our system, but most could not be verified due to no complete flight tests being done.

11.1 Legal Requirements

Since the drone was unable to be completed, FAA registration could not be undertaken since it requires a fully functioning drone. Also, a number of steps would have had to be taken to fly the drone outside on campus, but since we were able to get permission from Professor Mircea Teodorescu and the West Side Research Park faculty to use their indoor flight room for a controlled flight test, these steps were able to be done before testing in an uncontrolled environment.

11.1.1 FAA Compliance

To meet requirement STR9.1.0, FAA Compliance there are two ways to register drones with the FAA, which are the Part 107 and the Exception for Recreational Flyers processes. Our drone would always weigh less than the 55 pound limit when flying under Part 107[57]. However, even though the Exception for Recreational Fliers specifies that it is only allowed for flying for fun, we may operate under the Exception for Recreational Flyers since we will be using our drone for educational and research purposes under an institution of higher education. This is confirmed with an FAA statutory provision[58]. This registration would cost $5 and would expire after 3 years.

Additional rules that need to be followed for flying the drone under the Part 107 and Exception for Recreational Fliers would have included marking the drone with the registration number assigned. Other rules to be followed during flight include ensuring a line of sight to the drone is always available, making way for any manned aircraft that the drone may interfere with, flying below 400 feet in both controlled and uncontrolled airspace, and not interfering with any emergency response vehicles or personnel. Since we plan to fly our drone under these conditions, we can fulfill SRT 9.1.2, Statutory Provision and SRT 9.1.0, Part 107 Compliance, but cannot verify them until an outdoor flight test is done.

To register under the Exception for Recreational Flyers, Title 14 Code of Federal Regulations Part 47 needs to be followed[59]. Title 14 requires submitting an aircraft registration application AC form 8050-1 or through registering at the FAA Drone Zone website[60]. Also a notarized affidavit showing the full legal name of builder, model designation, serial number, number of engines, type of engines, max takeoff weight, class, and evidence of purchase must be shown. Since our drone was built by the team, it needs to be stated that we cannot provide a bill of sale document or most of the information on the affidavit, requiring another AC form 8050-88, Affidavit of Ownership for Amateur-Built and Other Non-Type Certificated Aircraft, that must be completed instead of giving model designation and serial number[61]. Form 8050-88 will require us to give specific motor information, the number of engines, and the specific class of aircraft, which is a small unmanned aircraft system. The drone must also not be registered in another country. After providing the following information and receiving the certificate of registration in the mail after an unspecified amount of time, the certificate must be carried at all times when flying the drone[60].

This registration process, broken down, would fulfill STR 9.1.2, FAA Drone Registration, but it could not be completed due to an incomplete drone. This means STR 9.1.0, FAA Compliance was not verified but could be fulfilled with the steps given to register the drone and fulfilled through a successful outdoor flight test once all the legal steps are done.

11.1.2 University Permittance

While flying a drone on campus, insurance and liability under the Unmanned Aircraft System Liability Policy may be provided by the university up to $25,000 if the drone is sanctioned by the university, if a line of sight is always kept, if the drone stays below 400ft, and if the aircraft weighs less than 55 pounds at the time of takeoff[62]. Since our drone is only meant to be 1 meter above the ground, weighs less than 55 pounds and will be flown within sight at all times, our drone follows all of these except for getting the drone sanctioned by the university.

In order to be sanctioned by the university, the drone must be registered with the FAA and the user must have a small Unmanned Aircraft System License, as stated in section 7.1.1. The drone must also be given permission to fly through contacting the university's Unmanned Aircraft System or through the UC Drones Web app. Once permission is granted from the app through a flight request form, which would require registration with the FAA, the drone would be available to fly for the requested flight times inputted. Since our drone cannot be registered with the FAA yet, an outdoor flight test could not be performed.

11.1.3 Helium Handling

Since our drone cannot be registered with the FAA, it cannot be flown outside on campus. Before testing outside, we were able to get access to an indoor flight room at Westside Research Park. We got in contact with Professor Mircea Teodorescu, who was able to give us access to use the indoor flight room at Westside Research Park. Only one of our team members, Dylan Harootunian, could get direct access to the lab, and was able to bring in one other unregistered team member as long as they were personally observed by either Professor Mircea himself or his graduate student Gordon Keller.

We first had to get helium to test our system, so we made contact with Praxair in Watsonville to get a T-size 4.8 helium tank with a purity rating of 99.998%. The helium did not have to be industrially pure, as this kind of helium is commonly used in commercial balloons and the effect of further purity would be miniscule to the buoyancy of the helium since the purity grade is already over 99.99%. This tank was transported using a truck while also ensuring it was strapped down for transport to West Side Research Park. Laura Ciravolo, the facility manager, and Ben Coffey, the facility coordinator, were then contacted and they confirmed that the helium tank could be stored at an adjacent room from the flight testing room for storage as long as it was strapped down to a secure metal fastening with two sets of chains and locks. For testing, the tank was then moved into the flight room, where a nozzle was connected to an vinyl airtight tube in order to fill the lift bag with helium. While filling up the helium, someone always had to be holding the helium tank for safety while they released the helium into the balloon. The nozzle itself was secured using rubber bands around the lift bag entrance. The lift bag inside the drone envelope was then able to be filled for testing while inside a netted testing area for flight.

The measures here are proof of verification of STR 7.3.0, Helium Safety, since it was verified to be sufficient safety measures when preparing for the incomplete flight test with the drone. The flight test was unable to be completed due to errors handling the drone itself instead of the helium that was used.

11.2 Safety Requirements

Some additional safety requirements had to be followed in order to avoid complications during testing and later functions. Popping the balloon was an issue during initial inflation of the lift bag within the envelope that was caused by sharp edges of the 3D printed parts. The puncturing of the liftbag during a test flight could be potentially dangerous to those in the surrounding area as control of the system would not be able to be maintained.By adding a layer of bubble wrap to the inside of the 3D printed mounting plates the issue of puncturers caused by the parts was solved. Over-discharging the battery was to be avoided by including a voltage alarm that would give the user a loud alarm when any cell of the battery went below 3.2V, since damage occurs to the battery if below 3V per cell was reached. However, in future iterations of the drone, a remote battery monitoring process will be needed for long distance flights.Hearing damage to bystanders from exposure to over 85dB was verified to not be a concern through a noise level test of the drone with motors at full throttle. For more on the noise test see Chapter 9. STR 7.1.2, Propeller Safety was not met because propeller guards were not implemented with the system. This was due to an oversight when deciding on the motors needed and not finding a propeller guard that would fit in time for testing, resulting in not fulfilling STR 7.1.0, Collision Considerations. The rest of the safety requirements are described in the following subsections.

11.2.1 Lift Bag Safety

During various inflation tests of the lift bag inside the envelope, sharp edges of the 3D printed parts had to be softened to avoid popping the lift bag, as had happened previously. The inside of the envelope where the 3D parts were attached, such as the propulsion attachment, gondola mount, and ultrasonic mount, were lined with bubble wrap on the insides so the sharp edges would not pop the lift bag. Additionally hot glue was also used on the edges of the outsides of the servo shafts to help soften the edges. Once these measures were taken, the lift bag no longer popped during the inflation while inside of the envelope. The measures here are proof of fulfillment of STR 7.1.1, Body Design Safety, since the balloon was verified to not pop when these measures were taken for multiple inflation tests. However, this requirement was unable to be verified with regards to the safety of the user due to the flight test being incomplete.

11.2.2 Electrical Safety

To avoid the draining of the battery while the drone was not in use, a mechanical switch was decided to be implemented onto the drone. This switch was placed in series just before the battery ground. When the switch was on, the circuit was open and effectively turned everything off. When on, the system would run again, where the ESCs would drain approximately 40mA each while the motors were idle. During testing, it was suggested that there should have been a remote switch on our remote controller as well to ensure the motors can be turned off from a distance if they malfunctioned. In future iterations, the mechanical switch would be kept to avoid the 40mA constant drain from our ESCs, but a kill switch would also have to be programmed to our remote controller to turn off the RC receiver so the motors would be stopped without having to reach for the mechanical switch at the bottom of the drone.

Another precaution that was taken was avoiding the over-discharging of our battery. The LiPo battery cells used can range from 4.2V-3V, where 4.2V is max charge and 3V is minimum charge. However, discharging a battery below 3V on any cell can damage the battery[63]. 3.2V was chosen as the cutoff voltage of the battery due to the emergency landing feature our drone was designed to have, as well as the steep discharge curve once Lipo batteries reach that level[64]. The voltage alarm implemented will emit a loud alarm when it detects any cell below that 3.2V limit, which will signal the user to land the drone. In future iterations, the drone will need to be able to detect any of the cell voltages remotely by itself and be programmed to enter the auto landing sequence if the voltage of any cell falls below 3.2V.

The electrical safety here suggests that our drone has met STR 7.2.2, Electronic Safety, through the implementation of the manual kill switch and voltage battery alarm. STR 7.2.1, Protected Electronics, was fulfilled through using electrical tape and zip ties to ensure the motor wires were properly secured to the lift bag to make sure the propellers do not make contact with the wires. Therefore, the overarching STR 7.2.0, Electrical Safeguards, was fulfilled but was unable to be verified since a complete flight test could not be completed that would prove the electronic safety.

11.2.3 Noise Test

To ensure our drone met the noise requirements of 65 decibels or lower, it was tested in an indoor environment through walking around the drone at 5 feet away while holding a decibel meter on an Apple Watch. The experiment procedures are shown in section 9.3. This test resulted in a decibel level of about 76 decibels, which does not meet our requirement but is also below the 85 decibel limit that will cause hearing damage over time. This test must also be completed in an outdoor test environment, since the indoor testing environment can have an effect on the decibel count through constructive interference of sound waves. However, since we are unable to test our drone outside due to legal reasons, this test was not able to be completed in time. This means that STR 10.0.0, Noise Level, was verified to not be met since the indoor testing of the noise level was greater than the 65 decibel limit. This can also be verified again in an outdoor test when FAA registration and university permittance is acquired by measuring the decibels of noise 5ft from the drone, since the indoor testing may have resulted in constructive noise interference causing the noise level to be higher than it actually was.

11.3 Conclusion

All of the subsystem technical requirements fulfilled within STR 7.0.0, Drone Safety except for STR 7.1.2, Propeller Safety. There are additional actions to take to ensure the safety of the drone, such as a remote shutoff switch instead of a manual shutoff switch on the drone, in addition to the implementation of propeller guards. However, a majority of the safety considerations have been fulfilled. STR 9.0.0, Legal Compliance, was unable to be verified but a process for registering with the FAA was laid out and the rules to follow when the drone is being flown were defined and confirmed for use with our system, showing sufficient legal considerations were made even though the drone could not be registered with the FAA.

Chapter 12: Costs and Funding

STR 5.0.0, System Costs, required the drone price did not exceed $10,000 and was less than $6,000. The current production price of the drone was $2,400 However, this STR cannot be verified because the system was not completed and additional costs may occur as the project is continued, in addition to the increase in retail price for the product. Most of the cost is due to materials ordered, but additional labor costs were added to estimates due to the labor intensive process of sewing the envelope, soldering the PCB board, and 3D printing. This was added to simulate outsourcing the manufacturing work. Additionally, efforts were made to reach out to a number of UCSC colleges for funding and to compete in pitch competitions for monetary rewards.

12.1 System Costs

Total costs of the system are broken down into the following categories and summarized from the system's Bill of Materials, which can be seen in the appendix in Chapter 14:

- Total Component/Material Cost is : $1493.56
  - Battery Charger and Battery: $369.98
  - Passive components (resistors, capacitors, and voltage regulators): $30.95
  - Controllers, Processors, and Oscillators: $69.28
  - Sensors: $85.22
  - Actuators, Transmitters, and Receivers: $322.08
  - Screws, fabrics and materials, PCB fab: $609.37
  - Plug accessories (U.FL, pin headers, battery holder): $6.68
- Estimated Fabrication Labor Costs: $906.03
  - Outsourcing sewing of envelope (40 hours) - $800
  - PCB soldering could range from $1.34-$6.03 per board depending on bulk of order (PCBWay)
  - 3D Printing Services (5 hours) - $100
- Estimated Product Costs ~ $2400

The highest costs of our systems are components and materials, which included helium costs, ordering the PCB layouts, as well as ordering the fabric and materials needed to fabricate the envelope and 3D parts of the drone. Other high costs included ordering the battery and charger that was decided for the system.

The labor costs were estimated using an approximation of the hours needed to complete the fabrication and a typical pay per hour.These approximations were based on how long it took our team members to do these tasks. Although the approximations were scaled down to be more in line with the hours a professional would take, the approximations are likely still overestimated. For fabricating the envelope, about 40 hours were needed to complete the fabrication. By approximating $20 per hour cost[65] of fabrication for 40 hours we get about $800 dollars. The 3D printing was also approximated to cost about $20 per hour[65] and since it took about 5 hours total to 3D print all the needed parts, the estimated cost was $100. Lastly, for professional soldering of the PCB board, which would be done in a future PCB iteration, it was stated to cost between $1.34-$6.03 per board to solder based on bulk order quotes at various quantities from PCBWay[66].

The total cost of our drone was about $2400 dollars total. This was less than half of our stretch goal of less than $6,000 and well below our requirement of less than $10,000. However, since the project was not completed, this is not the final cost of our drone. Additional costs may incur before the project is completed due to replacements needed or additional design considerations needed to address a problem later in the design of the drone when physical testing is able to be completed. Due to the very low cost of the drone currently, it can safely be assumed that the total cost of the drone will be less than $10,000 when the drone is completed. Therefore, STR 5.0.0, System Costs, is expected to be fulfilled.

12.2 Project Funding

Due to the nature of our project, much of the initial costs were out of the pockets from the team members. Some of the team member's colleges were then reached out for project funding. Porter College and Crown College approved funding for the project, while additional funding was received from winning Baskin Pitch Day 2021 and getting into the semi-finals of IDEA Hub 2021. A result of the funding can be seen in Table 12.1.

Table 12.1
Barone2 Project Funding

Funding Source	Crown College	Porter College	Baskin Pitch Day	IDEA Hub	Total
Funding Amount ($ Cash)	300	500	0	2,000	2,800
Funding Amount(Etc)	N/A	N/A	$200 Amazon Gift Card	$5,000 AWS Credits	$200 Amazon $5,000 AWS

In total the team will receive $2,800 in cash, a $200 Amazon gift card, and $5,000 in Amazon Web Service(AWS) Credits. Only $300 in cash and the $200 Amazon gift card have been received so far, but the cash to be received is $200 above the estimated costs of the drone. However, this will help cover additional costs related to replacement parts bought for the project. The funding will be used to reimburse team members who have paid for materials out of pocket based on the receipts they have for project expenses. Excess funding will be invested in the further development of the project.

12.3 Conclusion

While the estimated drone cost is $2,400, the total funding received was a value of $8,000. The current cost is well below the requirement of $10,000, STR 5.0.0, System Costs. Additionally the cost is expected to go down since purchase prices will decrease with larger orders. However STR 5.0.0, System Costs cannot be verified because the project is not yet completed. Due to incomplete variable business costs and our limited knowledge, this has a possibility of changing the cost dramatically. The large amount of funding will allow the team members to be reimbursed for their personal costs with the project and will allow future work on the project to be paid for as well.

Chapter 13: Conclusions and Next Steps

Although considerations and theoretical solutions were thought of for most technical requirements, the end result of the project was unable to be tested experimentally to verify most of the system technical requirements. Lessons learned by each team member and by the team as a whole are detailed in this chapter. Overall, we failed to meet the requirements given by USGS, although power draw did indicate the extended flight time is possible. The simulation confirmed controllability of the system, although adjustments should be made in servo support and in the manufacturing process. Finally, the next steps are explored if the project is continued beyond the end of this course, including what to keep, what to change, and recommendations to incorporate.

13.1 Review of Technical Requirements

Below is a summary of each of our 11 system technical requirements and a high level overview of their progress and verification status:

1. The drone shall fly for at least 30 minutes with magnetometer payload during normal autopilot flight. May fly for one hour as a stretch goal. **Power Test Verified**
    a. This was verified through power tests of each individual part that 30 minutes shall be reached and 1 hour could be reached, but could not be demonstrated experimentally as a whole through a flight test for either the 30 minute or the 1 hour goal.
2. The drone shall be able to fly 5mph in 15mph winds. **Simulation Verified**
    a. This was verified within the simulation environment described in Chapter 8. It was shown the drone could reach 5mph when facing a 15mph wind. Verifying this requirement for our drones ideal shape, however when simulated with our fabricated shape the requirement was not met.
3. The drone should have RC control implementation to allow for direct control of the drone. The drone can start in this state, or be switched to from autonomous control. **Matlab Verified**
    a. Closed Loop Control was tested independently of other features of the full requirements under 4.0.0, and auto landing, auto-take-off, large angle error, and hover functions were all verified in MATLAB, but not VREP
4. The drone should be able to fly on its own to collect data. **Not Verified**
    a. No significant work done
5. The drone shall cost less than $10,000. Reach-The drone may cost less than $6,000 **Expected to be Met**

   a. This requirement was verified with the current estimated costs of the drone to be $2,600, below both goals of the requirement. Even though the cost may change, it is very unlikely that the total cost will end up being above $10,000, fulfilling this requirement.

6. The magnetometer interference shall be less than 10 nT **Not Verified**
   a. This was not tested or verified because we were unable to get access to a magnetometer, but the test that would have been done was detailed. No analysis was able to be done to estimate the expected result as well.
7. The drone, its usage, and build should be safe to all individuals involved **Not Met**
   a. Safety precautions were taken for all considerations except for the safety of the propellers during a collision, causing this requirement to not be met. The safety of the electrical components and helium were secured through procedures in chapter 11.
8. The lift bag shall maintain 90% of its buoyancy over a one week period **Not Met**
   a. This was verified to not be fulfilled, as the drone lost 4.2% of helium in just half an hour, much more than what was allowed to escape.
9. The drone and team shall abide by all applicable laws for drone flight **Expected to be Met**
   a. FAA drone laws and registration process was researched, where it was seen that our drone would meet the requirements if registration was finished. Since the drone was not finished, it could not be registered with the FAA and this requirement could not be experimentally verified.
10. The drone should be quieter than 65dB **Not Met**
    a. This was only verified to not be fulfilled in an indoor environment at 72dB, but when the drone is completed and an outdoor test could be run, this can be tested again.
11. The drone should be able to be manufactured with equipment within our access **Not Met**
    a. Since manufacturing of the drone used 3d printing, a sewing machine, and soldering equipment, which were all available to the team, this requirement was verified.

Overall, only the flight time, drone speed and RC control of the project were verified, although the verification was not experimental. Rather, the verification was the result of simulation for the RC control and individual power tests for the flight time. There were 5 requirements that were verified to not be met, either due to fabrication errors or oversights in the project. The requirements that were expected to be met include the legal and cost requirements, which were expected to be met due to the confirmation that our drone would obey drone laws and the current costs of the system, respectively. The two requirements that were not verified include the autonomous design and the magnetometer interference. The autonomous design could not be verified because the effort was instead transferred to complete the RC control requirement and the magnetometer interference could not be verified due to the lack of equipment to test magnetic fields at the scale required.

13.2 Project Conclusions

The project faced a lot of challenges and obstacles, resulting in the failure of most requirements and the inability to experimentally test others. The physical prototype of the drone was only partly completed with only the servos and motors operational. Furthermore, the only physical flight test failed due to mishandling of the equipment, causing both lift bags of the drone to pop. Further testing was prevented by shipping delays and failure to complete certain systems for testing.

The simulation of the drone was also met with delays, causing the autonomous functionality of our drone to not be completed to ensure remote control simulation with controls implemented first. The PCB of our system had errors in the design that prevented its use, which also was unable to be fixed due to long shipping times beyond the constraints of the project.

The project was successful in a few areas. First, although the physical prototype failed to complete a flight, the motors were verified to supply sufficient thrust to lift the system when filled with helium by confirming both the thrust provided by the motors and the weight of the inflated system. Additionally, the flight time we hoped to achieve was proven possible through power draw testing of all of our systems components. Based on our power draw testing with our intended motors our minimum flight time would have not only been met but would have exceeded the flight time requirement by 33% and the hovering flight time goal would have exceeded the requirement by 58%. These power draw tests prove the plausibility of our primary requirement and could be considered the biggest success of this project.

Another success of our project was the creation of a simulation and control system. Although the closed loop remote control system was not able to be added to the simulation, the simulation was successfully able to verify that if our drone had the ideal shape it was designed for it would be able to meet our minimum speed requirement. The closed-loop control system for a unique and bespoke system was verified in MATLAB, although detailed simulations are needed to see if the design was correct. These are both large accomplishments that our team is proud of.

Despite our failures in many areas of the project several large accomplishments were made proving certain aspects of the project are viable and worth being persuaded by researchers and engineers in the future.

13.3 Lessons Learned

Throughout the three quarters of this project it is safe to say that every member of the team has learned more practical, technical, engineering skills and methods than any other single year.

The biggest lesson learned by the team was to check each other's work, which was gradually improved throughout the project through the increased work in sub-teams towards the end of the project. The environment created in smaller subteam meetings was better suited towards presenting and reviewing work to team members. Peer review is essential to any engineering project and needs to be done more by the team. We learned that the peer review process is done not because team members are not capable of doing work but because everyone makes mistakes and oversights, and by having their work questioned by other team members the vast majority of these oversights can be caught. This is especially true for larger projects where an oversight caused by one team member not considering others work will cause rippling effects throughout the project. Feedback can also help team members work more efficiently together as the project goes on through the improved teamwork and diverse thoughts considered.

Another lesson learned by the team was the tasks should never be exclusive to just one person. This is similar to the peer review lesson learned. If the team members are working together on issues more design considerations are made, especially if each member is working on multiple parts of the project. Tasks should not be assigned to a sole member as it will isolate them from the rest of the project. This is likely one of the largest mistakes we made as a team early on in the project. Each member was responsible for a chunk of work with little overlap. This meant that when it came time for the integration of the various systems. It was like trying to make 6 different projects fit together rather than 6 pieces of one project. To prevent this integration needs to be planned from the beginning of the project. This must be accomplished by the team being proficient in abstracting tasks that will be done later months ahead of time.

Something else learned is that working on a team assignment does not necessarily mean all at once. Early on in the project we would all try to create documentation at the same time, this was slow, ineffective, and even aggravating at times. When there are six competing ideas, communication can allow for a solution to be agreed on, but when communication fails it becomes impossible to find a compromise that makes everyone happy. Instead the compromises that were made to get the work done usually left everyone feeling unhappy and resulted in lower quality work. Even though every member's option should be considered on team assignments, by delegating individuals or smaller groups to work on assignments, foundations are laid much more efficiently. These foundations can then be peer reviewed by the other members of the team. This way there are less clashing ideas all at once but everyone's options are still heard and considered. Conversely, delegating someone to a task does not mean they alone should do it. When the team first started to not do team assignments all at the same time we sometimes ran into the opposite problem, where one person would essentially do the work on their own and the team would quickly give it a look over. This puts undue stress onto the individual assigned to this work and also results in work that is lower quality, completed slower, and has less design considerations made. Ultimately our team hit its stride in the latter half of the project when we started working more pairs and subteams, this is when we were able to get the most done as well as produce the highest quality work. For example, the physical testing of the drone was able to be attempted within a few days of meeting together to consolidate the envelope, gondola, and electronics needed.

The last thing the team learned was that organization tools like a Gantt chart or Trello boards are critical tools to use when completing a project. Completing a project without organization tools is like setting a nail without a hammer. Sure it can technically be done, but it would be so much faster and easier if the proper tools were used. Early on in the project the gantt chart was hardly used. We created it and then didn't look at it for weeks. This was the primary cause of our falling behind early on in the project. Once the gantt chart began to be used and checked on a regular basis more progress was made as people could better keep track of the due dates of their tasks as well as how delays in these tasks would affect the project as a whole. Additionally if a place in the project did fall behind schedule, it was able to be easily identified. The identification of these cracks allowed us to delegate extra assistance to these areas and help keep progress on the critical path moving forward. For example, When sensor programming started to fall behind team members who were familiar with the protocols that needed to be implemented were assigned hours to assist with this which allowed this work to be completed. Without the use of the gantt chart this crack would not have been detected and a fix could not have been applied.

These lessons learned have helped all of us grow as engineers. These lessons can be applied in any team based work environment, to see significant improvements in both the quality and efficiency of the team's work.

13.4 Personal Reflections

13.4.1 Dylan

This project has pushed me to and past limits I didn't even know I had. As an engineer, a teammate, and a leader.

As an engineer I learned a huge variety of skills more so then in any one class. By working in a project with so many different aspects I had to learn and give feedback on many different things. This varied from things I had some experience in and enjoyed such as coding or working with solidworks, to things I had some experience with but did not enjoy like analysis of torque or circuits, or to things I was clueless about like controls systems or V-REP simulation, as a member of the team I had to try my best to understand and help things things that I was unfamiliar or uncomfortable with.

As an engineer, I also learned that my biggest weakness is often my overconfidence. Sometimes this overconfidence is caused by laziness i.e. “I can put off that research because I can get it done in an hour”. Sometimes it is caused by prior experience i.e. “Oh I don't have to think about 3D printing stuff until the envelope is finished because I've printed a few things before and I do not need to learn anything new”. This destroyed me at every turn, from overlooking milestones at the second design review, to frantily spending night trying to fix 3D printer issues that I had no clue how to solve. These issues put an unneeded amount of stress on me that I was barely able to deal with. This stress made work even harder to complete as I now had to deal with my mental state as well as the work I had to do simultaneously. These stresses could have been avoided if I had been proactive in my acceptance that I may struggle with something. When I finally sucked it up and asked for help, whether it be from a teammate who knows more about 3D printing or from a peer or mentor outside the team who knows more about a subject than me, my stress was often greatly reduced. In order to become a better engineer I have to learn to not try and do everything on my own and ask for help before problems occur.

As a teammate I learned not to take things personally. I often become very defensive over my work, especially things I am struggling with. This is because I put a lot of effort into doing it and when it is questioned I often feel attacked. However this was clearly not the case. All members of the team just wanted to see the project go well and would often ask why simply to make sure I had a reason, or sometimes just because they were curious themselves. Even when the feedback was circular, I had to learn to swallow my pride and fix my mistakes not just for myself but for the good of the team. Once I got used to this I not only felt like I was better at maintaining a dialog about the work I had done, but I was better at asking questions to other team members about the work they were doing. This skill of being able to both take and give sometimes critical feedback to and from peers was without a doubt one of the most important skills I learned throughout this project.

Finally as a leader I feel like I have been pushed further than I thought possible. In the past I have had leadership experiences but In those experiences things often went well and when problems arose I was usually able to find a solution. This was not the case for this project. As a leader I felt I needed to come to every meeting with a can do and energetic attitude. In my past leadership experiences such as boy scouts where I needed to do this once a week this was easily doable. For a project that meets almost every day for close to nine months this was not feasible for me. Some days I was just tired and when I was I often felt like the meetings were not productive and this was on me. As a leader I needed to find a better way to have meetings run smoothly even when I am not at my 100%. This could have possibly been solved with better communication to the team of what needs to be done, or what with better delegation of tasks so that the team could still fully function without me. Additionally the mental toal being a leader of a project that has real, and difficult problems to overcome is much higher. The hardest parts of this project for me were not when I was getting roasted in a design review but when I had to sit and watch teammates be roasted. As the leader I often felt like any failure in any part of the project was on me. In a project that had so many failures this often took a heavy toll on me, this oftentimes made it even harder to bring the needed energy of a leader to team meetings, especially daily. I often found that when I was feeling out of it the meetings would often have long pauses where nothing got done. To remedy this I tried a new method. I stopped trying to put on an energetic mask at every meeting when it was not possible, instead I learned that sometimes being down to earth and honest with the team about how you are feeling often boosts team morale even if the feelings you are expressing are not positive ones. I believe this was effective because it reminded the team that we are all just people trying to do our best, and that we all need to have each other's backs if we want to succeed.

13.4.2 Leon

Looking back at the project, I realize just how little I knew starting out in the project. When I thought about Capstone in the past, I thought of it only as a year-long project with friends and nothing more. I didn't realize just how much work and planning went into it, and how working with a team for a full school year would look like. I learned this the hard way the first two quarters of Capstone, where I took difficult classes along with Capstone and was not able to put in as much effort into Capstone as I would have liked. I changed this in the final quarter of Capstone, and saw the most progress and satisfaction from the project. In terms of technical skills, I also realized how unprepared I was to start a project from scratch. I had always thought that it was easy to think of some project idea, and to make design choices. This course taught me how punishing it can be to make bad design choices, as we saw in buying parts and only afterwards realizing they were not suitable for our needs, or when parts broke and we had to wait weeks for new ones to arrive. Overall, this Capstone project was grueling, but taught me to be a better engineer.

13.4.3 George

The first lecture of the year, we were asked what engineering was. I remember answers were all over the place and there wasn't much consistency, but, to be fair, it is a hard question. I think I have my answer now, since personally I did not care for the definition of engineering as given in the class. Engineering, in my understanding, is the identification, analysis, and optimization of TRADEOFFS. At every design decision, alternate solutions should be considered, the benefits should be identified, then those tradeoffs should be analyzed with the math and science skills we have worked to develop. The ideal solution also needs to be defined, or the order of importance of different traits so the optimal solution can be chosen among the infinite possibilities of tradeoffs. Also, my understanding of engineering is more general, because the business, management, public relations, documentation, financial and other aspects are inextricably linked to engineering projects. If the best engineering solution in history is created, but we lack the skill to communicate our solution well enough to get the information out, we fail as engineers, since, in the end, nothing changed. Engineering is tradeoffs, and understanding that is the most important thing I could have learned in school.

13.4.4 Jeremy

Overall, I wish I could have learned more from this project in person, as I did in the last few weeks. I feel I was restricted a lot from the rest of the team geographically and it was hard to keep the motivation at first to push myself with this project because of the lack of tasks I could do virtually at the beginning of the project. At first, I could really only work with datasheets for parts we found online, while other team members had some software they were experienced with that helped them plan their work. I spent most of the beginning of the project helping other team members with their design considerations and taking on managerial tasks because there were only a small amount of design tasks I could do by myself. After the parts were ordered, and were then worked on by Leon, who lived closer to me than the rest of the team but there honestly was no thought of me driving to his house so I could test the parts that had already arrived. Another thing I struggled with was the engineering design considerations and the documentation that could have been written that described the different choices I could have made. I found a lot of my decisions could have been done better at the beginning of the project, even though they ended up working out in the end, such as the battery choice and the justifications for it.

The last month of the project, the idea came up that I should drive to his house to test the power draw of all the parts. In the last month, I was more interested and learned more than the rest of the time because I was physically there and working with the hardware. The physical verification of the parts that we had and the comparison of their expectations were able to be done, but they could have been done sooner if I had gotten the idea to go to another team member's house for physical testing of the parts they already had. The last week of the project, I wanted to help a lot because I definitely felt like I was the one on the team who had been doing the least amount of work. With my help, the physical testing of the drone was attempted but still failed due to some oversights. Overall, I really wish I would have been able to physically be with other members of the team earlier on in the project, because I struggled a lot with the motivation for a virtual project until it became physical.

13.4.5 Isaac

This design project challenged me to be responsible for work that my team is held accountable for. Whether I ended up completing a task or not meant less if the team were not able to achieve the same goal we had set for ourselves. Since the team's work relied on all sections to be completed in order for us to move forward, I learned the importance of labor distribution and the necessity to work on all parts of the project equally. Throughout this year, I also learned more of why having a large team did not necessarily make the project any easier; in fact, when everyone is required to perform and contribute equally well towards an end product, our project became the perfect example of how much more difficult making a product innovative and simple can be, given the task at hand. The main issue we ran into, trying to create the perfect product, was that we intentionally convoluted parts of our problem to generate more work to satisfy project requirements. Overall, I learned a lot more about simulation than I thought possible in an engineering project and learned to value the work that I did over the course of the year.

13.4.6 Ryan

Being the sole board designer of the team has been an immense challenge and I feel responsible for letting the team down on not being able to deliver my design to the team. I have made some critical mistakes in my design and resulted in a major change in the electrical side of the design. I learned to use textbooks and Appnotes for standardized board design in the wrong chronological order. But those mistakes have been realized and I know how to recover from my mistakes now.

During the fall quarter of the SDP, I felt confident in PCB design since I had prior hobbyist knowledge of the design process and that is why I chose the role. A lot of documentation and brainstorming was involved, and in retrospect, I wish we could finish brainstorming our idea earlier so we could put more time into System Technical Requirement. I was pretty proactive in weekly meetings and gave out a lot of ideas as to what components we could choose, but now I think I should have done more pugh charts for all the components we've chosen for our electrical specifications.

Winter quarter was very intense as I was also taking my design class along with SDP. The design class took up more time than I expected and I did not put enough time into researching as I needed to. If I was to design the PCB again I would have gathered my sources first on good PCB practices for our specific system technical requirements and gone in that direction.

Spring quarter was difficult in a different way, I was not a part of the fabrication process due to my location out of the U.S. Instead I worked on troubleshooting shipment delays due to concerns and geo restriction of certain components that were ordered on online electronic retailers. If I knew there were restrictions on exporting electronics outside of the U.S. I would have asked someone else to order it to not delay the shipping times. I am also taking 3 major ECE classes which took more time off SPD, similar to the situation in winter quarter and seeing my team working together on location placed some mental strain in my mind because I wish I was there to work with them and prove my engineering skills to the team. Instead I am stuck in a different location without the hindsight of knowing my design had some critical errors, which meant my team members could not implement my design. I think I would have shipped the components to myself and soldered it and found out sooner about my problems and I would have shipped it off after to the team when I completed verifying the board. Giving more work to my teammates was unfair to them and put them in more stress than they needed.

Going into this project, I knew I wanted to be responsible for board circuitry because I had novice experience with it personally and I wanted to apply my engineering design skills to this field.

However I did not work closely enough with Leonid and Jeremy on the wiring schematic. This resulted in a disconnect between my design with PCB and the implemented design with development boards. Looking back at it I would have prepared and done my research more before submitting my work without thorough verification and should have more frequent subteam meetings with Leonid and Jeremy to keep.

13.5 Team Reflection

The project as a whole provided further experiences to the team beyond the technical field, in areas that we had not considered before. For example, our ability to present material while receiving immediate technical feedback, from both peer and professor, and having to expand our perspectives to ask questions such as why we chose a design over another or where responsibility lies within the team and where it is being ignored. Simply working as a team brings many of its own challenges. In order to move forward as a team we have learned to negotiate around some of these challenges or address the problems within our team. Communication being the key element we have continually found to be worth improving. This is especially true as engineers who are in charge of design decisions; it is imperative that we take the necessary amount of time to explain and discuss our thought processes we have about certain topics with a level of professionalism and practicality, so that we might catch any errors before they are seen on the product.

In the same vein, accountability with our peers and most importantly ourselves is something we have learned to foster. After all, every sprint meeting is a reminder to keep ourselves accountable and productive so that we might have something to show for it at the end of a sprint or occasional design review.

Taking time to review information before meeting would also have been greatly beneficial and should be incorporated in the future. In addition to saving meeting time, it also would help everyone have ideas and thoughts going into the meetings and prepare us for more informed and thoughtful discussions.

13.6 Next Steps

13.6.1 What to Keep

One of the successes of the project was the creation of a simulation environment for testing the dimensions and movement of the drone. This simulation can take a variable wind speed, motor throttle, and motor angle. RC control was implemented into the simulation, but work needs to be done with integrating autonomous control.

The power budget of the system consists of estimated and verified power requirements of each part used, the voltage each part requires, and estimated heat losses. The power required is also variable depending on a performance factor for the motors and servos across a pre-set flight time on the power budget, allowing fast simulation of flight time.

The remote control system is verified in MATLAB and performs well, staying within all the technical requirements, even without fine tuning the system. The decoupling of the x and z components of forces and converting those forces back to the servo angle and thrust commands is, at least with current tests, a valid way of implementing a linearized control system. The tuning specifically will need to be adjusted with the aid of more detailed nonlinear simulations, but the integral path of the regulator will help maintain controllability. Also, a faster servo may want to be considered to help the control system react faster to the environment and help system stability.

The system's programming code was unfortunately not completed, but the layout is given in the flow chart and state machine in Fig 6.1 and 6.2-3 respectively. The layout provides a good base for developing the system behavior and can be adopted or further elaborated on in the continuation of the project.

13.6.2 Points of Failure to be Improved

Extra considerations need to be taken into account when fabricating the next iteration of the envelope. The envelope shape was not only fabricated to the incorrect shape, but the lift bag was punctured repeatedly. Future iterations of the envelope could experiment with an internal frame to provide protection for the lift bag as well as better support the weight of the propulsion system. Although this method would likely see benefits in structural stability it would likely increase the the effectiveness of the system, this would mean that either the system would have to increase

Future PCB design needs to be iteratively verified to ensure signal bus as far away from power traces as possible. Increasing PCB footprint from 4 by 4 inches to a larger size can help improve signal and power trace separation and a ground plane bridge needs to be implemented for the GPS module and the antenna.

Soldering components to the PCB should be done with reflow solder because the resistors and some of the IC's had only have solder pads on the bottom of them, using reflow solder would make it much easier.

13.6.3 Considerations of New Technologies

ZeRONE's blade-free propulsion drone uses ultrasonic vibrations of piezoelectric elements as propulsion. Each propulsion system is called a microblower, and each of the microblower's piezoelectric elements operate at ultrasonic frequency ranges which generate less noise than conventional quadcopter drones. The microblower flaps a diaphragm at ultrasonic speeds instead of using conventional spinning propellers, removing the risk of injury by propeller blades[4]. The

ZeRONE drone uses a 24- inch aluminum-metallized film balloon filled with helium gas making it a neutrally buoyant drone. The total weight without helium uplift of the ZeRONE drone is 106.4g including the balloon, microblower, carbon rods, drive circuit, receiver, battery, joint, screws, etc. The drone can be used as advertisement billboards in indoor crowds or halls. The drone also has been tested with a camera for crowd monitoring, human flow analysis and security. However, the microblowers cannot provide enough thrust beyond 1 meter per 7.5 second upwards and downwards. ZeRONE is prone to drift caused by slight winds, either by people walking past it or areas with air conditioning. [3]

Although the drone uses low noise microblowers to provide lift, it cannot counteract external forces such as slight breeze which makes it not ideal for precise sensor data collection. ZeRONE cannot add IMU and GPS sensors for autonomous flight control without increasing the balloon diameter and adding more microblowers. The technology is not advanced enough yet to be used to solve researchers problems with data collection, but can be useful with further development and should be watched.

13.6.4 Recommendations for Proceeding with Project

If someone was to continue working on this project there are several recommendations that can be made. First, future iterations of the physical prototype could experiment with an internal frame to provide protection for the lift bag as well as better support the weight of the propulsion system and maintain system shape, especially if the lift bag deflates. Although this method would likely see benefits in structural stability it would likely increase the effective weight of the system without adding additional helium, but this will also increase drag. An optimization problem should be written and solved containing the information developed throughout this report to weigh the effects of all these factors and help with a more optimized drone. For the physical implementation of the system, finding the balance between the weight and size of the system is the primary problem that needs to be solved. Our main recommendation is that support be added to the mechanical system for better structural integrity and the balance of weight vs size of the system be reconsidered.

The best part of the system that future work could continue on would be the simulation. We were able to create an environment that is able to support real time simulation of aerodynamics on our system. We were also able to design some control systems. By finishing the work of implementing the controls system into the V-Rep simulation using Remote API these control systems could be verified and improved upon through better guided fine tuning. The simulation phase should be the focus of incremental testing due to the difficulty and costs of building the drone, and working more with simulation may have helped us identify certain problems earlier. The VREP simulation used should not be the only one, since simulation of the system's structural rigidity as the lift bag deflates would have helped us identify the problem that caused the second flight test to fail.

Finally through power draw testing we were able to successfully verify a longer flight time of our system was theoretically possible for up to 40 minutes, however, since our PCB design was unsuccessful, the circuitry layout was not optimized for efficiency. By finishing the implementation of an efficient PCB that takes advantage of the switching regulators the heat loss of our system can be minimized allowing for the possibility of even longer flight times.

Although this project failed in many respects, the parts that were verified prove a valuable foundation for future continuations of an extended flight time buoyant system for data collection. Since the idea for this project was conceived around a buoyant system this likely limited our considerations for other methods of improving flight time. Methods of extending flight time other than buoyancy could be explored as technology improves and as helium supplies run low, but we feel we showed, even with drone technology that changed how researchers collect data, there is still room to improve. These methods could still be explored using the aerodynamics simulation as well as our unique propulsion system that could allow more payload stability in flight of non buoyant drones, since the drone would no longer depend on having a tilt angle to maneuver.

We hope that future work is persuaded on the creation of a long flight time system as it would greatly benefit many researchers, and could pave the way for future breakthroughs in the fields of aviation.

Chapter 14: Appendix

The following is a link to the project Github: [lgshuster/Barone2 (github.com)]

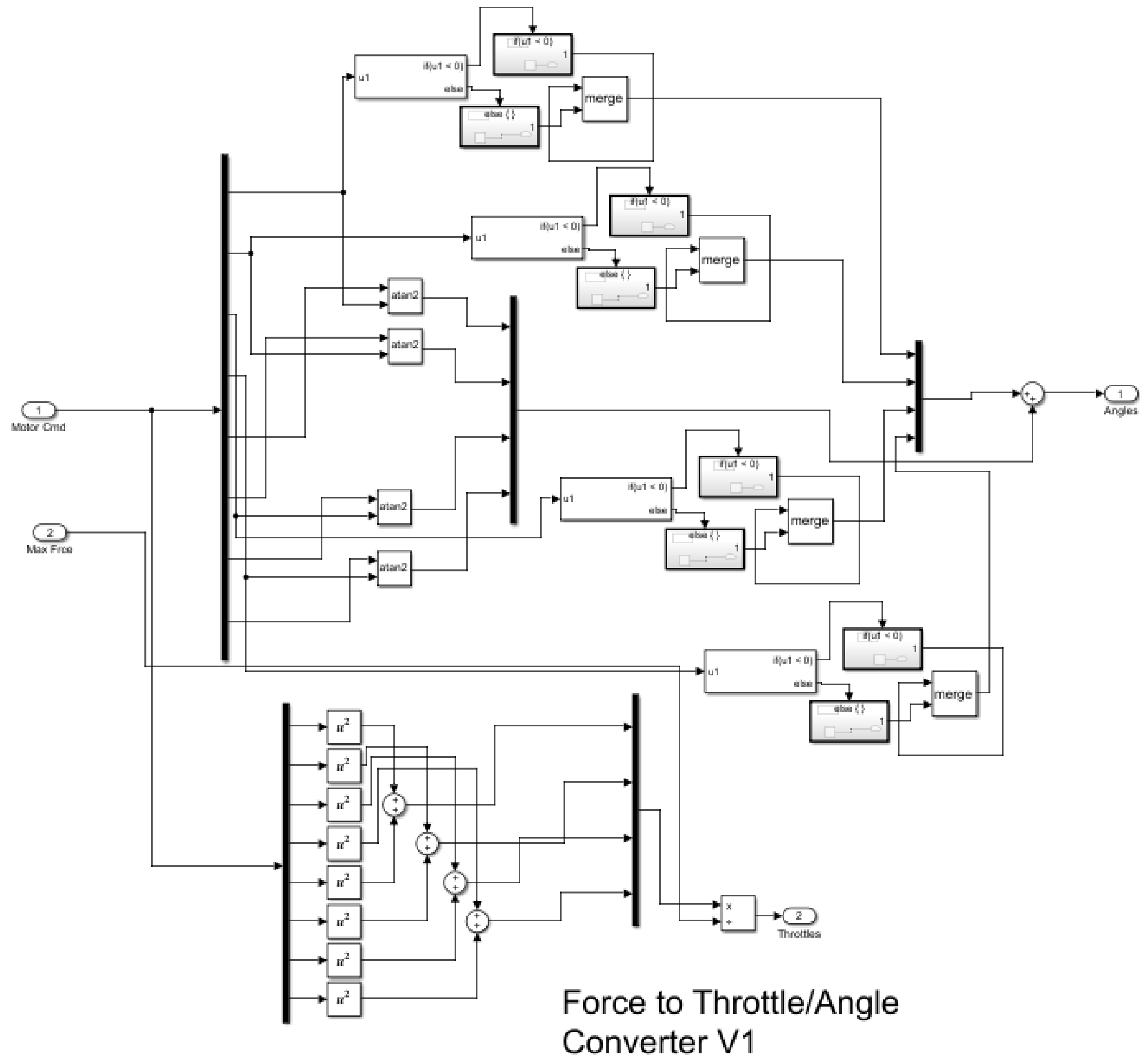
This Github includes the following:

- Systems programming code
- Matlab Controls
- Simulation Code
- CAD Files
- Matlab Analyze files
- Eagle Cad files for PCB design
- Wiring schematics

The following items are also presented as the appendix after the bibliography on the paper:

- Force to Throttle/Angle Converter used in Closed-Loop Controls
- System Technical Requirements
- Bill of Materials(BoM)
- Weight Allocation
- Power Budget V5.2 with Original Motors
- Power Budget V5.3 With Replacement Motors

Force to Throttle/Angle Converter used in Closed-Loop Controls



Bibliography

[1] Bouligand, Claire. “Distribution of Buried Hydrothermal Alteration Deduced from High-Resolution Magnetic Surveys in Yellowstone National Park.” Journal Of Geophysical Research-Solid Earth, vol. 119, no. 4, AMER GEOPHYSICAL UNION, 2014, pp. 2595–630, doi:10.1002/2013JB010802.

[2] J. M. G. Glen, A. E. Egger, C. Ippolito, and N. D. Athens, “Correlation of Geothermal Springs with Sub-Surface Fault Terminations Revealed by High-Resolution, UAV-acquired Magnetic Data,” p. 8.

[3] W. Yamada, H, Manabe, and D. Ikeda “ZeRONE: Safety Drone with Blade-Free Propulsion,” thesis, Dept. Research Labs, NTT DOCOMO Yokosuka, Yokosuka, Kanagawa, Japan, 2019.

[4] Yamada, Wataru, Hiroyuki Manabe, and Daizo Ikeda. ”Zerone: Safety drone with blade-free propulsion.” Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. 2019. Y

[5] Yao, Jingjing, and Nirwan Ansari. ”QoS-aware power control in internet of drones for data collection service.” IEEE Transactions on Vehicular Technology 68.7 (2019): 6649- 6656.

[6] Shoaxmedova, Nozima. ”IMPROVING DATA COLLECTION TECHNOLOGY IN RURAL AREAS USING DRONES.” 16 (2020).

[7] “Matrice 600 Pro - Product Information - DJI.” DJI Official, www.dji.com/matrice600-pro/infospecs.

[8] “Census of Fatal Occupational Injuries (CFOI) - Current and Revised Data.” U.S. Bureau of Labor Statistics, U.S. Bureau of Labor Statistics, 22 Dec. 2020, www.bls.gov/iif/oshcfoi1.htm2019.

[9] Aug 12, 2019. “Companies Use Drones to Limit Dangerous, Potentially Fatal Tasks for Workers.” Occupational Health amp; Safety, ohsonline.com/articles/2019/08/12/companies-usedrones-to-limit-dangerous-potentially-fatal-tasks-forworkers.aspx?m=1&fbclid=IwAR1eD8a42VdW1WpK961 Q2xblDOFpSvY0p2C-QjOl7yS9bPC1009CVNR5ro.

[10] Tarigan, A. P. M., et al. ”Mapping a volcano hazard area of Mount Sinabung using drone: preliminary results.” IOP Conference Series: Materials Science and Engineering. Vol. 180. No. 1. IOP Publishing, 2017.

[11] “Data Capture with Drones – Digital Engineers’ Eyes in the Sky.” Aurecon, www.aurecongroup.com/expertise/digital-engineeringand-advisory/data-capture-drones.

[12] Keller, Gordon. “Re: Long Flight Time Buoyant Drone.” Message to Dylan Arius Harootunian. 24 Nov 2021. E-mail.

[13] 36.39 -- Helium-filled balloon. (n.d.). http://web.physics.ucsb.edu/~lecturedemonstrations/Composer/Pages/36.39.html#:~:text=Helium%2C%20which%20has%20a%20mass,%2D0.164)%20%3D%201.02%20g.

[14] *Equations from Blevins, Robert D. Applied Fluid Dynamics Handbook. Repr. ed. with corrections. Malabar, Fla.: Krieger Pub. Co., 1992.*

[15] P. Virtic, P. Pisek, T. Marcic, M. Hadziselimovic and B. Stumberger, "Analytical Analysis of Magnetic Field and Back Electromotive Force Calculation of an Axial-Flux Permanent Magnet Synchronous Generator With Coreless Stator," in IEEE Transactions on Magnetics, vol. 44, no. 11, pp. 4333-4336, Nov. 2008, doi: 10.1109/TMAG.2008.2001528.

[16] Physics for Scientists and Engineers, 4th edition, by Douglas Giancoli.

[17] Anne Marie Helmenstine, P. D. (n.d.). *Why Helium Balloons Deflate So Quickly*. ThoughtCo. https://www.thoughtco.com/why-do-helium-balloons-deflate-4101553#:~:text=Helium%20balloons%20float%20because%20helium,the%20balloon%20stays%20inflated%20longer

[18] Peterson, Dave. “Dave Peterson.” *The Math Doctors*, 17 Feb. 2020, www.themathdoctors.org/making-a-sphere-from-flat-material/.

[19] Beard, Randal W., and Timothy W. McLain. *Small unmanned aircraft: Theory and practice*. Princeton university press, 2012

[20] *Aerodynamics for Students*, www.aerodynamics4students.com/.

[21] ELEC Freaks, “Ultrasonic Ranging Module HC - SR04”, NA, NAhttps://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[22] NXP Semiconductors, “MPL3311A2”, NA , April 12 2018
https://www.mouser.com/datasheet/2/302/MPL3115A2-1127148.pdf

[23] InvenSense, “ICM-20948”, DS-000189, June 2 2017
https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf?ref_disty=digikey

[24] GlobalTop Technology, “MT3339”, FGPMMOPA6H, Jan 31 2012
https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf

[25] Honeywell, “MPRLS0001PG00001C”, 32332628, NA
https://sensing.honeywell.com/honeywell-sensing-micropressure-board-mount-pressure-mpr-series-datasheet-32332628-en.pdf

[26] Brad Suppanz, PCB Trace Width Calculator, Advanced Circuits, 2018.
https://www.4pcb.com/trace-width-calculator.html

[27] UltraCAD Design Inc., “Using the IPC Temperature Charts”, NA, Jan 2001
https://www.ultracad.com/using_ipc_temp_charts.pdf

[28] Henry W. Ott, Electromagnetic Compatibility Engineering 1st ed. New Jersey: Wiley, 2009, pp. 624-625.

[29] Henry W. Ott, Electromagnetic Compatibility Engineering 1st ed. New Jersey: Wiley, 2009, pp. 638-639.

[30] Henry W. Ott, Electromagnetic Compatibility Engineering 1st ed. New Jersey: Wiley, 2009, pp. 106-107.

[31] "PIC32 Microcontroller Brochure Datasheet by Microchip Technology," *Digi*. [Online]. Available: https://www.digikey.com/htmldatasheets/production/1226947/0/0/1/pic32-microcontroller-brochure.html?utm_adgroup=Integrated+Circuits&utm_source=google&utm_medium=cpc&utm_campaign=Dynamic+Search_EN_Product&utm_term=&utm_content=Integrated+Circuits&gclid=Cj0KCQjw5PGFBhC2ARIsAIFIMNclkbZgYsZL4TNqLElx6Y66iKwxbTL8PUYJ-CD_EM-rkiEYb5YGFjwaAvgxEALw_wcB. [Accessed: 06-Jun-2021].

[32] "Power Supply," *Power Supply - Raspberry Pi Documentation*. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md. [Accessed: 06-Jun-2021].

[33] "AKK KC03 2.8MM 120 Degree 800TVL NTSC Switchable Camera with 40CH 600mW FPV Transmitt," *RC Groups RSS*. [Online]. Available: https://www.rcgroups.com/forums/showthread.php?2892758-AKK-KC03-2-8MM-120-Degree-800TVL-NTSC-Switchable-Camera-with-40CH-600mW-FPV-Transmitt. [Accessed: 06-Jun-2021].

[34] "Flysky FS-i6 Transmitter / FS-iA6B Receiver Digital Proportional Radio System," *Dragon Sailing North America*. [Online]. Available: https://radiosailing.net/products/flysky-fs-i6-transmitter-fs-ia6b-receiver. [Accessed: 06-Jun-2021].

[35] "Serial Telemetry Radio Kit - 915MHz, 100mW," *WRL-15007 - SparkFun Electronics*. [Online]. Available: https://www.sparkfun.com/products/15007. [Accessed: 06-Jun-2021].

[36] A. Industries, "MPL3115A2 - I2C Barometric Pressure/Altitude/Temperature Sensor," *adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/1893. [Accessed: 06-Jun-2021].

[37] "Ultrasonic Distance Sensor - HC-SR04," *SEN-15569 - SparkFun Electronics*. [Online]. Available: https://www.sparkfun.com/products/15569. [Accessed: 06-Jun-2021].

[38] A. Industries, "Ultimate GPS Module - 66 channel w/10 Hz updates," *adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/790. [Accessed: 06-Jun-2021].

[39] "ICM-20948," *DigiKey*. [Online]. Available: https://www.digikey.com/en/products/detail/tdk-invensense/ICM-20948/6623535. [Accessed: 06-Jun-2021].

[40] "MPRLS0001," *MPRLS0001 Honeywell Sensing and Productivity Solutions | Pressure Sensors, Transducers*. [Online]. Available: https://www.digikey.com/en/products/base-product/honeywell-sensing-and-productivity-solutions/480/MPRLS0001/454895. [Accessed: 06-Jun-2021].

[41] "Turnigy Aerodrive SK3 2822-1275kv Brushless Outrunner Motor," *Hobbyking*. [Online]. Available: https://hobbyking.com/en_us/turnigy-aerodrive-sk3-2822-1275kv-brushless-outrunner-motor.html?queryID=&objectID=47269&indexName=hbk_live_magento_en_us_products. [Accessed: 06-Jun-2021].

[42] Banggood.com, "Flash Hobby D2822 1100KV 1450KV 1800KV 2600KV 2-3S Brushless Motor For RC Airplane," *www.banggood.com*. [Online]. Available: https://www.banggood.com/Flash-Hobby-D2822-1100KV-1450KV-1800KV-2600KV-2-3S-Brushless-Motor-For-RC-Airplane-p-1628289.html. [Accessed: 06-Jun-2021].

[43] "Sail Winch Servo 25T 10.63kg / 0.9sec (360deg) / 55g," *Hobbyking*. [Online]. Available: https://hobbyking.com/en_us/sail-winch-servo-13kg-0-7sec-360deg-55g.html. [Accessed: 06-Jun-2021].

[44] R. W. Beard and T. W. McLain, *Small unmanned aircraft theory and practice*. Princeton, NJ: Princeton University Press, 2012.

[45] "Pololu 5V, 5A Step-Down Voltage Regulator D24V50F5," *Pololu Robotics & Electronics*. [Online]. Available: https://www.pololu.com/product/2851. [Accessed: 06-Jun-2021].

[46] J. Galos, K. Pattarakunnan, A. S. Best, I. L. Kyratzis, C.-H. Wang, and A. P. Mouritz, "Energy Storage Structural Composites with Integrated Lithium-Ion Batteries: A Review," *Wiley Online Library*, 15-Apr-2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/admt.202001059?casa_token=4qYDt26PI0YAAAAA%3A9QCrXJ3Hc0xuuQ3ql-sT0bae0kJfknuRlFTqXRbpW_mq09iNIKVys1IIDBa7rcchB98vW-imUv32wDc. [Accessed: 06-Jun-2021].

[47] "Lithium Polymer Charging/Discharging & Safety Information," *MaxAmps.com*. [Online]. Available: https://www.maxamps.com/lipo-care.php. [Accessed: 06-Jun-2021].

[48] S. Barcellona, M. Brenna, F. Foiadelli, M. Longo, and L. Piegari, "Analysis of Ageing Effect on Li-Polymer Batteries," *TheScientificWorldJournal*, 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4506813/. [Accessed: 06-Jun-2021].

[49] "LiPoly Battery - When to stop draining?," *Electrical Engineering Stack Exchange*, 01-Jan-1961. [Online]. Available:

https://electronics.stackexchange.com/questions/32321/lipoly-battery-when-to-stop-draining. [Accessed: 06-Jun-2021].

[50] "LiPo 11,000 3S 11.1v Battery Pack," *MaxAmps.com*. [Online]. Available: https://www.maxamps.com/lipo-11000-3s-11-1v-battery-pack. [Accessed: 06-Jun-2021].

[51] Boeing, Adrian & Braunl, Thomas. (2007). Evaluation of real-time physics simulation systems. 281-288. 10.1145/1321261.1321312.

[52] "Enabling the Remote API - Client Side." User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm.

[53] "Remote API Functions (C/C++)." User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctions.htm#simxGetIntegerSignal.

[54] "Shape Dynamics Properties." User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/shapeDynamicsProperties.htm.

[55] "Remote API Modus Operandi." User Manual, CoppeliaSim, https://www.coppeliarobotics.com/helpFiles/en/remoteApiModusOperandi.htm.

[56] Hall, Zac, and Zac Hall, "How Accurate Is Apple Watch Noise Level Detection for Hearing Health?" *9to5Mac*, 25 Sept. 2019, 9to5mac.com/2019/09/25/apple-watch-noise-app-accuracy/.

[57] *Register Your Drone*, 02-Dec-2020. [Online]. Available: https://www.faa.gov/uas/getting_started/register_drone/. [Accessed: 04-Jun-2021].

[58] B. Guthrie, "Text - H.R.302 - 115th Congress (2017-2018): FAA Reauthorization Act of 2018," *Congress.gov*, 05-Oct-2018. [Online]. Available: https://www.congress.gov/bill/115th-congress/house-bill/302/text?q=%7B%22search%22%3A%5B%22PL%2B115-254%22%5D%7D&r=1. [Accessed: 04-Jun-2021].

[59] "Electronic Code of Federal Regulations," *Electronic Code of Federal Regulations (eCFR)*. [Online]. Available: https://www.ecfr.gov/cgi-bin/text-idx?SID=10f2c4539074face0af16e24e02809f8&node=pt14.1.47&rgn=div5. [Accessed: 04-Jun-2021].

[60] "Aircraft Registration," *Aircraft Registration – Unmanned Aircraft (UA)*, 25-Sep-2020. [Online]. Available: https://www.faa.gov/licenses_certificates/aircraft_certification/aircraft_registry/UA/. [Accessed: 04-Jun-2021].

[61] "Information to Aid in the Registration of Civil Aircraft" *Aircraft Registration – Unmanned Aircraft (UA)*, 25-Sep-2020. [Online]. Available: https://www.faa.gov/licenses_certificates/aircraft_certification/aircraft_registry/media/REG-AR-94.pdf. [Accessed: 04-Jun-2021].

[62] UC Center of Excellence on UAS Safety, “UC Drones Knowledge Portal,” *UAS Insurance*. [Online]. Available: https://ucdrones.github.io/ch-insurance.html. [Accessed: 04-Jun-2021].

[63] “Lithium Polymer Charging/Discharging & Safety Information,” *MaxAmps.com*. [Online]. Available: https://www.maxamps.com/lipo-care.php. [Accessed: 04-Jun-2021].

[64] “LiPoly Battery - When to stop draining?,” *Electrical Engineering Stack Exchange*, 01-Jan-1961. [Online]. Available: https://electronics.stackexchange.com/questions/32321/lipoly-battery-when-to-stop-draining. [Accessed: 04-Jun-2021].

[65] “Average Maintenance Technician Hourly Pay,” *PayScale*. [Online]. Available: https://www.payscale.com/research/US/Job=Maintenance_Technician/Hourly_Rate. [Accessed: 07-Jun-2021].

[66] “Prototype PCB - Online PCB Quote - Full feature custom PCB prototype service at low cost - PCBWay,” *Custom PCB Prototype Manufacturer*. [Online]. Available: https://www.pcbway.com/orderonline.aspx. [Accessed: 07-Jun-2021].

[67] *PIC32 Family Reference Manual Section 14. Timers*, http://ww1.microchip.com/downloads/en/devicedoc/61105f.pdf

[68] *Section 15. Input Capture*, https://ww1.microchip.com/downloads/en/DeviceDoc/60001122G.pdf

[69] *PIC32 FRM Section 16. Output Compare*, https://ww1.microchip.com/downloads/en/DeviceDoc/61111E.pdf

[70] *Ultrasonic Ranging Module HC - SR04*, https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[71] *Section 24. Inter-Integrated Circuit (I2C)*, http://ww1.microchip.com/downloads/en/devicedoc/61116f.pdf

[72] *Xtrinsic MPL3115A2 I2C Precision Altimeter - Data sheet*, https://cdn-shop.adafruit.com/datasheets/1893_datasheet.pdf

[73] *DS-000189-ICM-20948-v1.3.pdf*, https://cdn.sparkfun.com/assets/7/f/e/c/d/DS-000189-ICM-20948-v1.3.pdf

[74] *PMTK command packet-Complete-A11*, https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf

[75] *PIC32 Family Reference Manual – Section 21. UART*, http://ww1.microchip.com/downloads/en/DeviceDoc/61107G.pdf

[76] “What Noises Cause Hearing Loss?” *Centers for Disease Control and Prevention*, Centers for Disease Control and Prevention, 7 Oct. 2019, www.cdc.gov/nceh/hearing_loss/what_noises_cause_hearing_loss.html.