Senior Design 2 Final Project Document

December 1, 2021

D.E.A.R Drone

Department of Electrical Engineering and Computer Engineering
University of Central Florida

Dr. Samuel Richie

Group B

Austin Perkins	Electrical Engineering
Raymond Chenoweth	Electrical Engineering
Ellie Lane	Computer Engineering
Dan Biller	Computer Engineering

Table of **Contents**

Table of Figures

Table of Tables

1. Executive Summary

This project was designed around the idea that each member would incorporate his or her own skill set and past into this project. The team members consist of two electrical engineers and two computer engineers, so it needed to incorporate aspects of electrical analog systems and also software design and the mix of these two skill sets was a computer-based system in a mechanical machine format. We then dived further to incorporate our past projects and ideas into the “flow” and gave us new ideas and problems to worked to fix. We took in Ellie’s past in machine learning, Dan’s past in drone manufacturing, Austin’s background in mechanical systems, and Raymond’s background in electrical analog systems to came to the idea of drones using machine-based learning. We incorporated the idea of accident avoidance, and the current environment in the world in response to Covid and how that limits trade and thought of a faster response to delivery in using drones and accident avoidance software to fly safely and quickly through urban environments that trucks found a hard time navigating though. This combined together to form the problem and solution of our drone project.

Now that we had the base of our idea, we now needed to went through the difficulties and needed of doing something like this drone would needed. We immediately started the discussion of how our drone would achieve the main component of this project, accident avoidance. We needed a system that could thought fast, incorporate multiple data points and parameters, and act in its best interest to the circumstance. This was where the different aspects of our disciplines would came into played. We needed to had a CPU that could handle these different parameters as well as reliably fly the drone as this was an automated system instead of direct control like most commercial drones were. We would then needed different sensors to incorporate different data points so the drone would knew when it needed to avoid an obstacle or when it was safe to fly. We needed to incorporate computer vision with a camera for even more accuracy as well for different environments that some sensors won't worked well in. Then machine learning was directed to the project through the used of simulations, thus “teaching” our drone before any needed for physical testing, which could cause hundreds of dollars in damages to the product depending on what happens. And finally, we needed to adjust our drone to our specifications of weight, lift, and space to incorporate all our different sensors and package space.

We had certain aspects of these designs that we needed to attain for the drone were viable for operation, and our objectives followed this route. Starting backward from the drone frame design, we needed to limit the weight of the drone frame, sensors, brain, and all other aspects of the weight because of diminishing returns on lift compared to the motors used for lift. For safety and ease of operation, we’ve designated half of our total weight that our motors could lift to the actual drone, thus giving the second half to flight operation and the ability to avoid obstacles when needed. Next, this drone itself was the prototype so there was an obvious limit to the size and weight of the package it would carry, but it would still needed were physically adjusted to reliably carry our package. Our drone needed to had safety incorporated into every feature of its design as well because this product had the potential to directly interact with customers when delivering or picking up packages. We needed to incorporate such things to guard against accidents. That was half taken care of by our accident avoidance program, as that would avoid directly coming into contact with any humans or animals during operation. The other half needed were taken care of through our physical modifications to our drone itself or safety manuals we supply for operation. We also needed to had a long battery life so as to had enough energy for our drones to reach destinations that could be on other sides of cities, which when taking into account the growth of cities, could be long distances. The drone would operate completely independently of any control networks and was capable of working even if it loses all networking signals. It was fully autonomous and all of the decision-making happens on the flight control board. The computer would used GPS for coarse path planning of the overall trip. The sensors and computer vision was used for fine path adjustment. This means that no human intervention was required throughout the flight process. While the drone does had a network connection to the ground station, updates and alterations could be issued to the drone during flight. Also, the flight could be canceled and the drone could return back to its launch site. This means that the operator had the flexibility to control the drone if they wanted to, but it was not a requirement for a human to took any actions during a delivery flight.

With proper budgeting and realizing where and when needed to acquire higher quality products versus bulk, we would came closer to a realistic budget and what we actually needed to spent. Currently, we were in the agreement of a budget of $1000 with a deviance of plus or minus $200 depending on design changes down the line. Sticking to this budget would require compromises and decisions were made, which were similar to what companies making a profit would had to made also. Our time frames were realistic and achievable and incorporate design and writing. We would followed our own timeframes so as to stay ahead of deadlines imposed by outside influences. Time management was an excellent skill to had in any workforce environment and we were improving upon ours in order to maintain the quality of output for the project.

Research had shown us that while our project was being incorporated into many companies, it was the edge of research into its respective field, and any advances into the aspect of accident avoidance would provided leaps and bounds into logistics and advancement into the field of drones and the future of the supply lines in the world. This was already steadily looked inwerecause of events in our world such as Covid, the blockage of the Suez Canal, and many others that had affected delivery times and cost the global market millions of dollars. We already saw this in such things as Matternet, Zipline, and the top competitor Amazon. This was only taking into account drones as well. Accident avoidance could be incorporated into hundreds of technologies, such as autonomous driving, helping in medical fields for the disabled, and many more. The progress being made by drone projects would continued to perfect those efforts and saw where it could be used more in day-to-day life while also providing extreme convenience for people needing deliveries. Imagine being able to order a project, medicine, food, and much more and had it delivered within the hour because the city in your area was able to package it up and send it on a drone were delivered to your door.

2. Project Description and Motivation

With the rise of artificial intelligence in recent years, coupled with cheaply available multirotor platforms, autonomous drones were being used in more places. Furthermore, the jobs they perform were becoming increasingly complex. One of these roles was package delivery.

Figure 1: Obstacle Avoidance

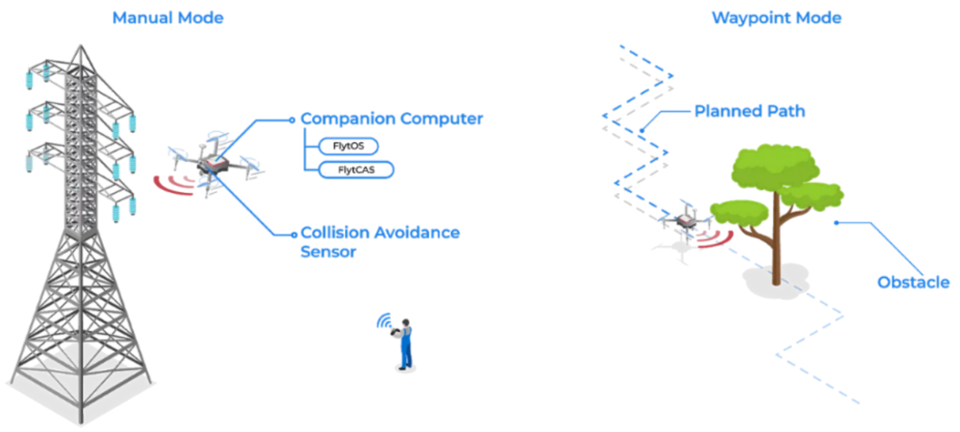
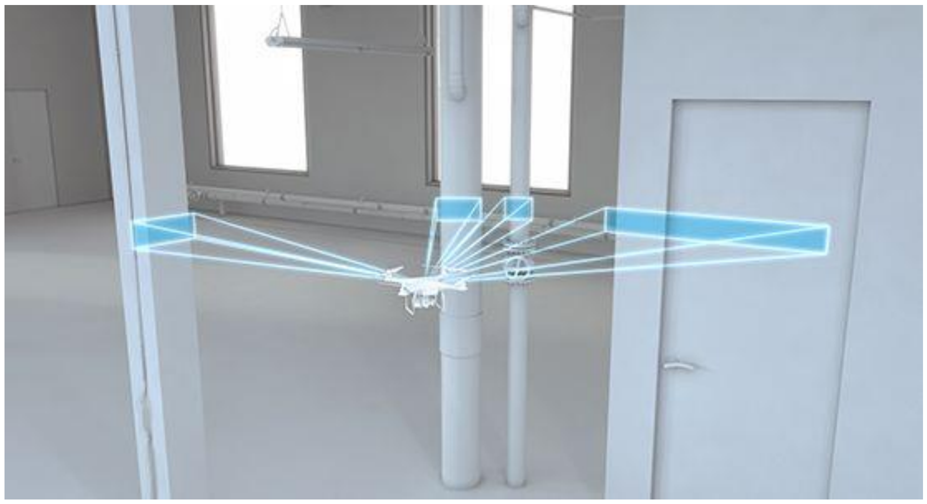


Figure 2: Proximity Detection



Online shopping had been trending upwards for years, and the recent pandemic had only accelerated this. This and other package shipping services require a cheap and scalable method for last-mile delivery: the delivery from local distributors to customer’s homes. Additionally, it could be problematic to deliver medical supplies and food to remote areas that were not accessible by vehicles.

Drones were the perfect platform to fill this role. They were cheap and scalable, as well as being able to travel quickly and navigate to non-accessible places. However, one problem with current drones was that they lack the ability to detect and avoid local obstacles. A general course could be planned and followed using GPS and terrain mapping data, but there was still a needed to avoid things like buildings, trees, and other drones **Figure 1**.

The project we were proposing was a payload delivery drone with the ability to detect and avoid local obstacles **Figure 2**, enabling fully autonomous package delivery.

Our motivation comes from the last huge changed in the distribution processing would have been shipping containers. What shipping containers did was made a standard unit that could be transported easily on ships, trains, and almost every form of transportation that we currently used. The impact that had was astronomical on the global economy. Can be easily stated that shipping containers were the cornerstone for the world trading market currently. We believed our drones could potentially had the same level of impact but on a more local scale. When improving the distribution of packages and mail. You were effectively making the world a more closely net Community.

Regardless of if you were delivering postcards or Christmas gifts you were essentially bringing people who cannot be in direct contact closer together. That means family who lived out of state, our brave soldiers who were serving this great country were more connected to their loved ones when you improve package delivery and distribution. That was our Motivation for building a drone that could deliver packages, ultimately, we believed we were bringing people closer together than ever before.

2.1 Goals

Our first goal was to used collision avoidance in combination with GPS. The sum total of all of our ideas was to make a drone that was capable of maneuvering itself and delivering a package. In hopes to revolutionize the distribution of our mailing systems of today. The goal was to make distributing mail quicker, cleaner, safer, and more effective than it had ever been before. In doing so we believed the best way did that was with a drone that had an external camera, IR sensors, and GPS, that could function practically anywhere and deliver packages safely and effectively.

Another goal was to make sure our drone delivery systems were as diverse and capable to match the surroundings and environment. With the used of GPS and an external camera, we hope that our drone was flexible enough to safely maneuver the number of objects and avoid them. While continuing its path to deliver the package.

To be able to create a drone that was capable of maneuvering in accomplishing such a goal, we needed to apply knowledge of kinematics and robotics with a touch of aeronautics. We would create a test bed that would allow us to test the drone’s ability of object avoidance using machine learning before applying it to her actual drone. Once applied to the actual drone we would have the computations be ran on the flight controller that would monitor the position and orientation of the drone autonomously.

Our final goal was to inspire the next generation of future minds were interested in technology and how the world functions and what could still be improved upon and used to make the world a more effective and better place. These drones were used and highly populated cities in towns and we hope that the youthful spectators were inspired to looked up our DIY instructions of how they could build their own drone and hopefully became drone hobbyists.

2.2 Specifications and Objectives

To guide us with what we were accomplishing in our drone project then specific objectives and specifications needed were made. These needed to included clear direction and numerical values of what we wanted to achieve. If these items were not made clear or numerical then there could be miscommunication and not a very guided direction to head in when it comes to making decisions about how to build.

2.2.1 Objectives

Our methods to achieve our goals for this project, was more of a divide and conquer approach to hopefully and successfully complete this building project.

The first objective of our drone build project was to acquire the necessary hardware such as the frame, the motors, propellers, ECS, and flight control. It was important that these materials had the proper specifications for our drone objective. The frame needed were large enough to support carrying a package as well as giving us enough room to build on top of it. The motors and propellers needed to provide enough to lift tonight only carry the frame in the hardware but as well as a package and be efficient enough to last the duration of the trip. The flight controller needed were robust enough to handle the inputs from all of our sensors to accurately maneuver the drone effectively to its respective coordinates.

The next Objective was to make sure we had the proper batteries incorporated into our drone build. The importance of having the proper batteries was critical for multiple reasons. First being you don't want too heavy of batteries for your needed because ultimately it would way down your drone and made it less efficient. Ultimately you would want a Goldilocks middle region of a drone that supplies the perfect amount of power for our flight time but it's not too heavy to weigh down the frame.

Successful object detection, tracking, and collision avoidance was our most important objective as well as our top priority. This feature was the cornerstone of our project, and its success would determine the outcome of this project. In order to Achieve this Objective, we plan on having an onboard flight controller that was in charge of the orientation and stability of the drone while also doing the computations of the input from the GPS and the external camera. Rather than detecting individual objects, our camera system would create a three-dimensional map of its environment by using computer vision as well as the inertial measurement unit on the flight controller. Using this map, the navigation system would detect if the current planned path with colliding with any physical objects on the map. If so, evasive maneuvers were taken to navigate around the object or structure. Once the obstacle had been avoided and there was a clear path for the drone to return to its planned course, it would do so and carry on as if there was no obstacle. This system was implemented using a Simultaneous Localization and Mapping (SLAM) algorithm to create a point cloud. This point cloud was fed to the navigation system to worked out the most efficient path around the obstacle.

Our final Objective was to have a fully functioning package delivery drone that was user-friendly to where you could plug in the correct coordinates, and it would find the best path from the lift-off point to the package's destination.

2.2.2 Specs

When deciding our specifications, we wanted to keep it reasonable and affordable. In **Table 1**, the physical specifications were majorly influenced by costs because the larger the quadcopter the more expensive materials were, but with that, we needed a sweet spot of size in order to fit the modifications. **Table 2** had our goals in terms of electrical performance by trying to maximize energy efficiency and increasing performance. **Table 3** goes into the type of sensor specifications we needed to fly the quadcopter/drone to the destination and be able to perform robot vision for the object detection system.

Table 1: Physical Specifications

Physical Size of Drone	2 to 4 feet long
Number of Motors	4
Number of propellers	4
Number of Circuit Boards	3
Number of Controllers	1
Weight of Drone (Without Batteries)	5 - 8 pounds
Weight of Batteries	1 pound
Payload Weight	0.3 - 0.5 pound

Table 2: Electrical Specifications

RF of Controller and Drone	2 - 2.4 GHz
Voltage of Drone	13 - 15V
Battery Capacity	5000 - 6000 mAH

Time of Battery to Run	30 Minutes
Amount of Voltage in 1 cell of LiPo	3.6V
Controller Voltage and Current	7V and 1A

Table 3: Sensing Specifications

Field of View	60 degrees horizontal (+- 10 Vertical)
Range of View	0.5 - 10 Feet
Measuring Frequency	10Hz
Autopilot	GPS
Number of Sensors	4 (IR, Camera, GPS, Altitude/Pressure)

2.3 House of Quality

Here we had our House of Quality in **Table 4**, Where we showed a correlation between the customer requirements and the engineering requirements. We saw this using the legend given in **Table 5** to correspond what we were seeing to what the data means, which shows a strong correlation of certain subjects like Accident Avoidance in the engineering requirements to the customer requirement of actual Package Deliverability and Safety of Package. We saw in the “hat” or triangle at the top of the HOQ the correlation between different engineering requirements and how they relate to one another. Such as Timing Accuracy, Turnover Time, and Accident Avoidance having strong “+” relationships.

House of quality was an important evaluation of what was a priority to the customer and project and provided that bridge of understanding for the mission. This ensures proper communication between the engineers and the people interested in the product. To create this from the perspective of engineers it was highly brainstorming what a customer might wanted and be interested in for the product. We had to used empathy and put ourselves in the customer's shoes to saw what we would want from the product without extremely considering the detailed design. From there we created the customer priorities on the left column. Then we thought about the technical parts of drones and the product we were building from an engineer and design point of view, which was where the titles in the top row were created. The middle consisted of thinking about the relationship between to two and visualizing the priorities ahead. These requirements were not always the same, but when the thought of together do have some correlation sometimes when considering making them happened together or how they interact.

In order to made this more accurate from a company standpoint, it was in their best interest to research and survey customers to saw their unique perspectives. There were teams typically placed on design projects like this to interact with people from outside the project went that feedback and not created an echo chamber of ideas. These strategies helped made the product more applicable and the House of Quality was a great tool to summarize that information for a team working with designers and engineers. The benefits would trickle down into the project by cutting down costs. If you could saw that one design factor was greatly prioritized and connected then the team knows to put a lot of time, though, and money into it. But for items not as greatly prioritized, it could be used to cut costs and not push the team so hard for a feature that was not as cared about as others.

Table 4: House of Quality

					+					
				+	+	+				
			-	-	-	+	-			
		+	-	-	-	+	-	-		
	Functional Requirements									
Direction of Improvement	▲	□	▲	□	▲	▲	□	□	□	□
Customer Requirements	Sensor Range	Payload Lift	Power Efficiency	Flight Speed	Accident Avoidance	Timing Accuracy	Turnover Time	Frame Durability	Functional Requirement 9	Functional Requirement 10
Package Deliverabilty	●	○	●	○	●	●	●	▽		
Payload Weight	▽	●	●	○	●	●	▽	▽		
Delivery Time	▽	○	●	●	●	●	○	▽		
Cost	●	●	●	▽	▽	●	●	●		
Durability	▽	▽	▽	▽	▽	▽	▽	●		
Usability	▽	●	○	●	●	●	●	○		
Aesthetics	▽	▽	▽	▽	○	▽	▽	○		
Ease of Maintenance	▽	▽	▽	▽	●	▽	●	●		
Safety of Package	●	▽	▽	▽	●	●	▽	●		
Customer Requirement 10										
Importance Rating Sum (Importanc	306	398	470	186	514	498	442	422	0	0
Relative Weight	9%	12%	15%	6%	16%	15%	14%	13%	0%	0%

Table 5: House of Quality Legend

Relationships		**Weight**
Strong	●	9
Medium	○	3
Weak	▽	1

Direction of Improvement		**Correlations**	**Correlations**
Maximize	▲	Positive	+
		Negative	-
Target	□	No Correlation	
Minimize	▼		

2.4 Block Diagram

Diagrams needed to represent this project was a general flight diagram and then the software component. **Figure 3** represents the basic form of the path that the drone would fly through to deliver a package safely and accurately to a GPS point. Having this visual was a great start for what was needed to go into accomplishing it. A huge project like this needed to be broken down into smaller missions or tasks that made up one product. That idea was Divide and Conquer. Completing one part of the diagram leads a path to the next. **Figure 4** shows a diagram of how the software was performing.

The software part of a drone contained its own set of instructions and pathways to accomplish its goal. **Figure 4** goes into detail about how it would detect an object and process it in order to properly avoid it. This gets accomplished by working with the hardware of the drone, the sensors, the camera, and the program written. From there it took in all the data and pass it through algorithms that process it to determine what needed were done and sends those instructions of avoidance to the drone. We were using specifically computer vision, which had the optical flow, filters the image, and gets smarter with each used. The ultrasonic sensor worked to give data of where the object specifically was or how far away it was from the drone, which helps determine how fast a decision needed were made or how fast the program was able to make that decision.

Figure 3: Flight Path Diagram

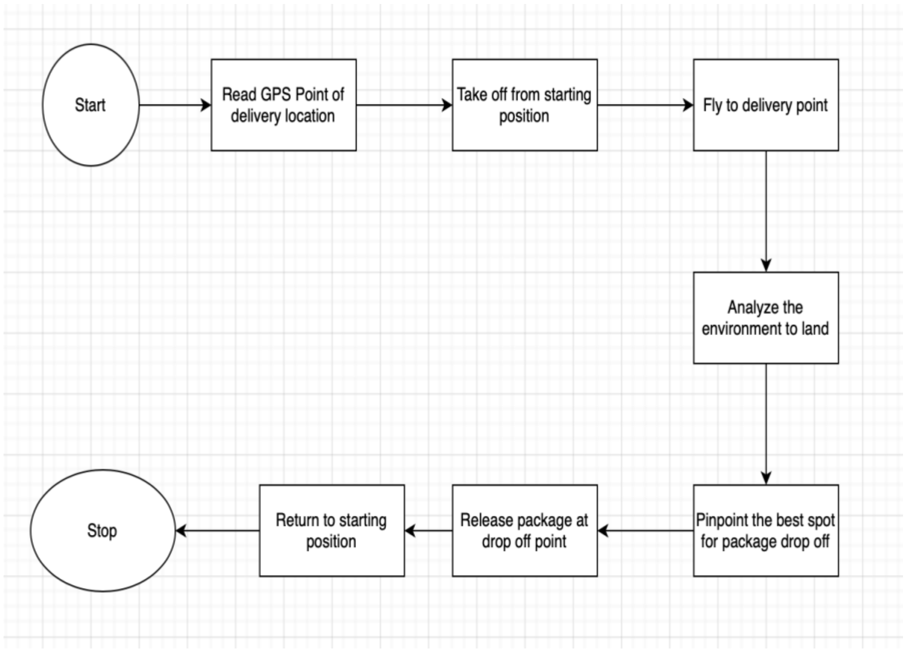
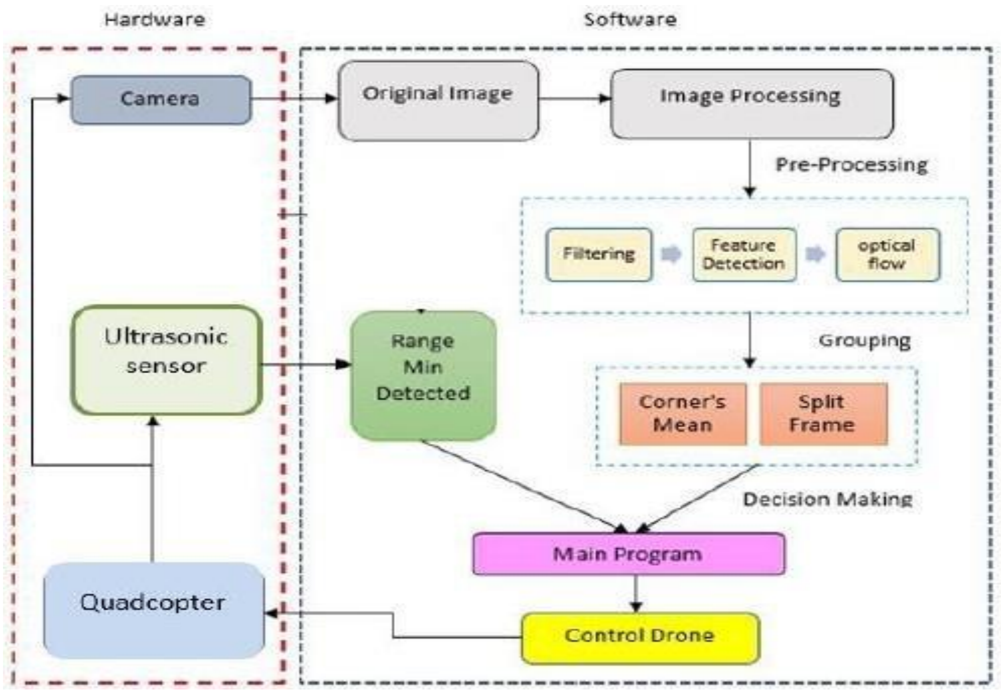

Start
Read GPS Point of delivery location
Take off from starting position
Fly to delivery point
Analyze the environment to land
Pinpoint the best spot for package drop off
Release package at drop off point
Return to starting position
Stop

Figure 4: Software and Hardware Flow



These diagrams were a great guide into what needed went accomplished and a visual of what was happening. There were many factors to a team besides just engineering and having things laid out in a visual guide those conversations of features and what was happening. An expanded team for a drone might had a sensor team that only works on sensors and might not needed to know about the computer vision but just needed to know how it was interacting or the importance of where the data was going. Overall, very helpful for keeping the communication in line with the teams.

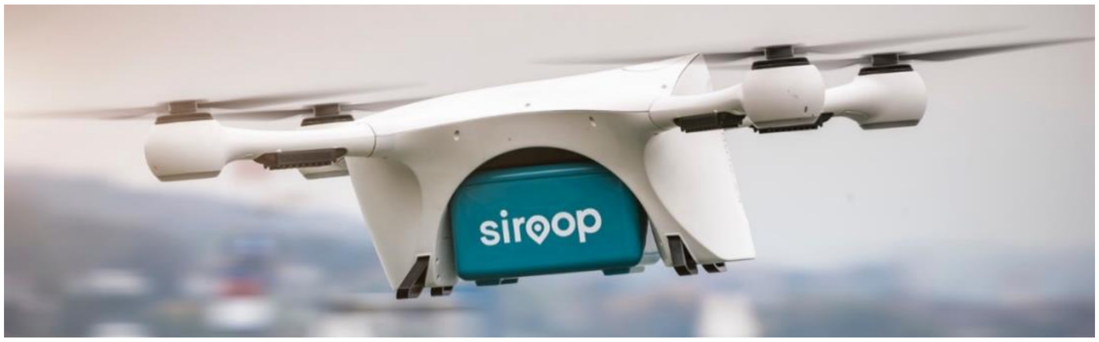
3. Technical Content

3.1 Existing Projects

When approaching an idea, it was important to saw what had already been done and currently out there on the market. If research was not done, then repetitive creation could have happened and after a bunch of hard worked and hours put it then it could end up already being around. This section goes over existing projects that helped provided us with direction and got the ideas flowing for what was out there and what was still left to discover and created.

3.1.1 MatterNet: Drone Companies in the Medical Field

Figure 5:MatterNet's Drone Product



MatterNet was one of the few companies in the recent future that had stepped into the field of logistics and medicine with the helped of drones. Their company started the climb to the top of their field by getting permission from the country of Sweden to provide their services of moving blood samples and other needed supplies between hospitals using drones, as shown in **Figure 5**, and increased their reputation through their innovative housing unit for the drone, which made the interaction between the drone and user simple and easy to understand. They then partnered with the US company UPS to supply logistics in the NC-based hospitals and to included themselves in US airspace. Their drone was intuitive and easy to use, with a quadcopter style-based drone with a center space designated for packages. It was continuously monitored by in-company drone pilots but was ran on the autopilot and accident-avoidance system so as to remove human error and time frames. MatterNet's mission statement regularly takes their drone in city limits and flies in urban environments, showing the similarity in situations we envision our drone's mission statement were.

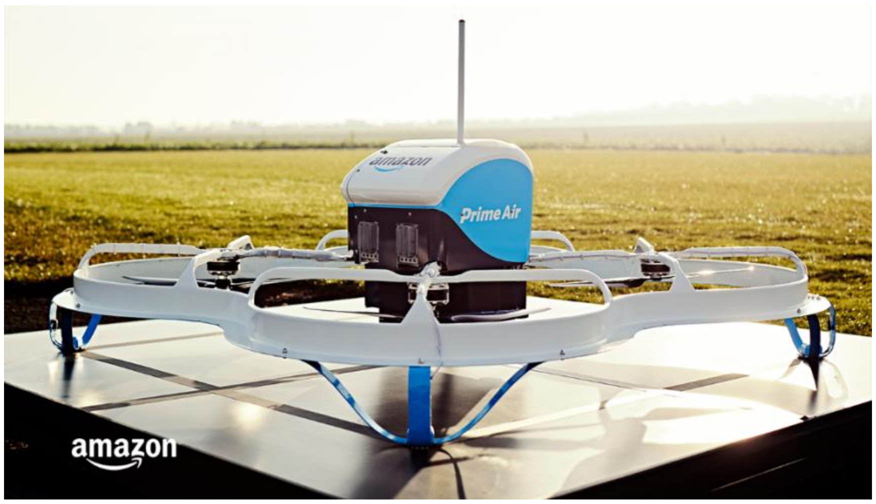
3.1.2 Amazon Delivery Drone

Prime Air, a division of Amazon, was focusing on making 30-minute delivery a reality. While this requires many infrastructure changes, the key development they were using to make this possible was an autonomous delivery drone. The drone was loaded with a package and then sent off to deliver it. The autonomous drone was capable of taking off, navigating to the destination, selecting a safe drop-off zone, navigating back to the warehouse, and landing. It does this all without the supervision of a human.

The specifications of Amazon’s delivery drones were very similar to our own specifications. The drone could fly for up to 15 minutes and deliver up to 10 miles away, giving it a round trip range of 20 miles. These specifications were maintained while carrying a payload of up to 5 pounds. Additionally, the propellors were fully shrouded for safety. This shroud also acts as a fixed wing, increasing the efficiency and speed of the drone. This was a key factor in achieving the above specification requirements. It was also an area where our drone differs from Amazon’s implementation. Their drone was controlled with six degrees of freedom as opposed to the standard four degrees of freedom. This helps the drone to remain stable in windy conditions. **Figure 6** shows the actual drone.

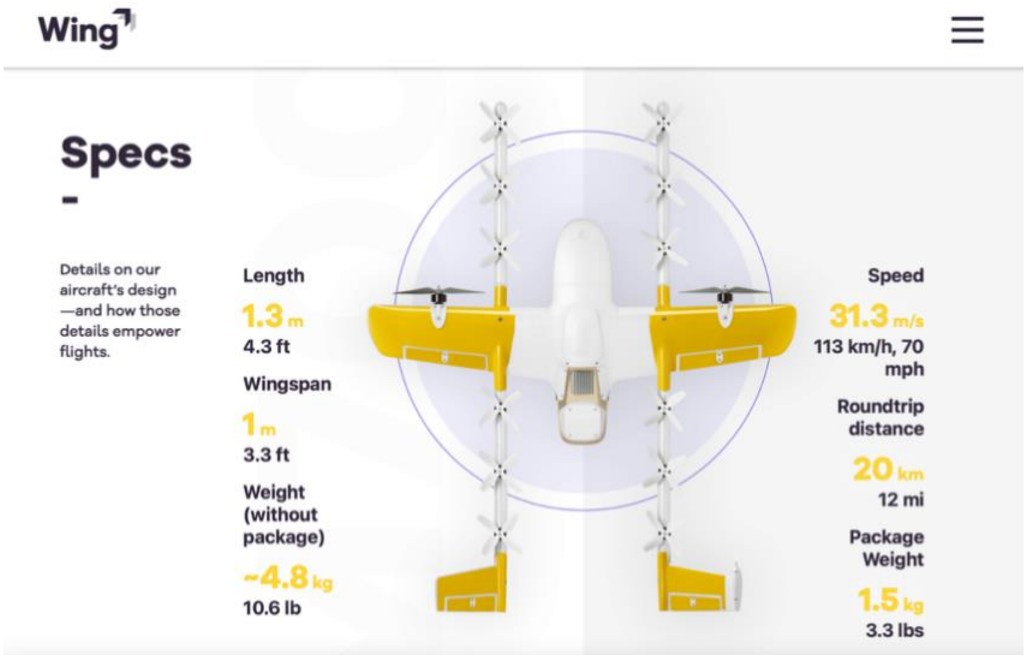
Prime Air’s drone uses similar, but more advanced sensors to our own. They utilize sonar, thermal cameras, and depth cameras to sense in every direction around the drone. These sensors were utilized in the two most safety-critical stages of flight: transit and drop-off. During transit, the drone uses these sensors, combined with machine learning algorithms, to detect static and moving objects from any direction. During drop-off, the drone scans the area for hazards such as powerlines, wires, people, and animals, as well as uneven terrain. This allows the drones to made decisions in real-time to kept themselves and the people around them safe.

Figure 6: Amazon Delivery Drone



3.1.3 Google/Alphabet Wing X

Figure 7:Wing Prototype 1



Specs

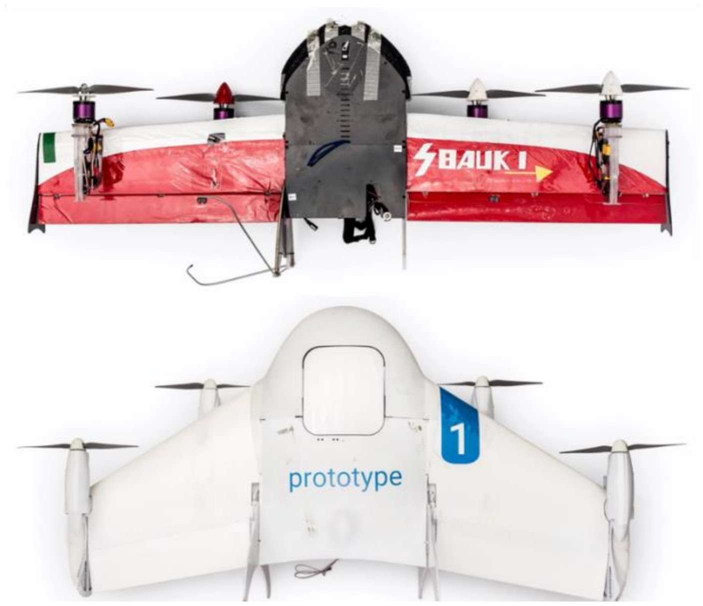
Details on our aircraft's design —and how those details empower flights.

Length	1.3 m, 4.3 ft
Wingspan	1 m, 3.3 ft
Weight (without package)	~4.8 kg, 10.6 lb
Speed	31.3 m/s, 113 km/h, 70 mph
Roundtrip distance	20 km, 12 mi
Package Weight	1.5 kg, 3.3 lbs

Starting in 2019, Wing, a drone carrier project under Google and Google's parent company Alphabet, received Air Carrier Certification from the FAA. This drone had been testing and going through prototype models for a few years prior and doing test flights in Australia, Finland, and the U.S. **Figure 7** below summarizes a lot of the performance specs that came with the approved Wing carrier drone. Some of the design choices seen that made for interesting consideration were the multiple smaller propellers in parallel of each other that helped it had more hover control. The drone also uses tether technology in order to held packages [1]. Researching more of the technology that was included with the success of this drone, showed similarities to our design choice. Both our drone and the Wing included the used of LiPo batteries, Wi-Fi, GPS, and used of a camera. The LiPo batteries were the best choice for maximum energy and power output in the industry right now for projects of this caliber. Wi-Fi and GPS were used to communicate with the drone and pilot system to kept track of where the drone was, where it needed to go and alternative routes in the case of avoiding objects or things in the way. The camera was used for computer vision to give the drone the most accurate readings of the surroundings and its delivery spot and that was exactly what we would try to accomplish by using a camera on ours for its computer vision capabilities [2].

The drone did not always have this structure, if anything, it almost looked like the complete opposite. In **Figure 7**, the length was longer than the wingspan, but in earlier prototypes, it had a longer wingspan than the length size, this could be seen in **Figure 8**. This was an interesting observation and brings up curiosities on how this affected the power output, consumption, and accuracy of the drone. The drone also capitalizes on modern technology and investments to helped made it a zero-emissions drone. In comparison, our drone structure would have more of a square shape overall and the body of the drone was built in the Z-axis of sorts. We were hoping this would give it more balance once components were being added to the drone and kept the center of gravity in one compacted area [1].

Figure 8: Wing Prototype 6 and 7



The technology included was not completely public but speculated were a Python and C++-based back-end system for the traffic management system and then the front end included JavaScript. This being a Google drone was able to implement their advanced GPS and mapping systems that were being used in much other software. Seeing this fully built and operational drone using the same software's that we would utilize was reassuring that we were headed in a great direction. Our drone would also be using Python and C++ in the firmware and software for controlling. These were dependable and widely available resources for computing drone projects with those languages. Python provides a wide variety of computer vision and machine learning libraries and resources that were applicable to drones. C++ was an embedded systems software to easily communicate with hardware, firmware, and software at once for the controllers and such [2].

3.1.4 Tesla's Autopilot

Tesla had one of the best and most competitive auto pilot cars on the market as of current within the automobile industry. They were currently capable of object avoidance as well as self-driving to the owner of the car through a parking lot to the most convenient location for them. These features were similar to what was implemented on our drones such as collision avoidance and auto pilot to a destination.

The Tesla automobiles currently worked through a Central control computer. This computer controls the acceleration of the car depending on the objects it detects. The object avoidance that Tesla implements were far more sophisticated than what we had to implement in our project but how Tesla implement and anticipate what objects would do was very useful to how we wish our drone has when confronted with the obstacles.

Tesla automobiles were equipped with numerous cameras that were capable of processing lived images and interpreting hundreds of objects simultaneously. With this implemented in Tesla cars were capable of detecting traffic lights, road debris, and other hazards. We anticipate our drone were capable of detecting a fewer number of objects but hopefully had the same outcome of obstacle avoidance the Tesla had accomplished [45].

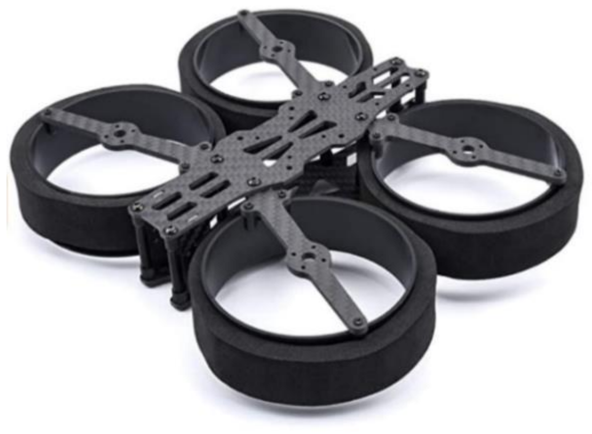
We also wish our drone were capable of finding its way to a certain location. Tesla had implemented this in their cars through their key fob. Drivers were able to click on the key fob and the car was able to go into auto pilot and drive to the most convenient location for the driver. With package delivery in mind for our drone, we hope to have a similar outcome [45].

Due to budget, time, and processing limitations, we were not expecting our drone were able to detect all of the objects that Tesla cars were able to however we do expect were able to detect a few objects simultaneously and accelerate and navigate accordingly. We saw Tesla automobiles as inspiration on how it was currently revolutionizing how cars function in our society and how well designed its autopilot currently was.

3.2 Details Related to Project and Part Selection

3.2.1 Frame

Figure 9: The Readytosky 149HD

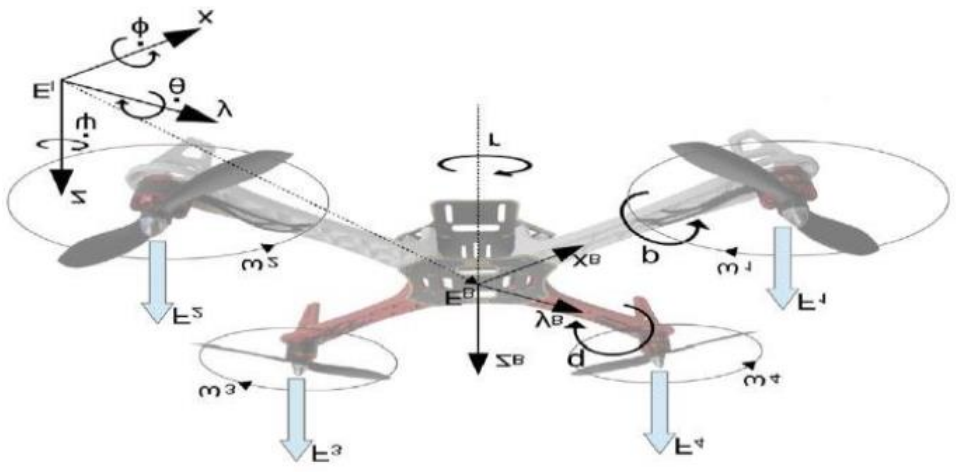


The frame was the foundation of our drone project. So, making sure to pick a drone Frame that was able to support all our needed was critical. We then narrowed it down to 4 drones to pick from. The first was going were The Readytosky 149HD. Some pros and cons about The Readytosky 149HD as seen in **Figure 9**. The Readytosky 149HD was a great drone from its reviews and we liked the idea of having guards for the blades because safety was always a major concern. The cons being that we might not had enough building area on the drone to accomplish our goal because the drone was only 149 mm large. Also, with being that small we were also unsure if we were able get the correct amount of lift needed to deliver and carry our package.

The Readytosky S500 was the next drone we took at as seen in **Figure 11**. The pro of this drone was that it was 450mm and we believed it was large enough for any design or configuration of sensors we needed to put on it. Also, it had landing gear which we believed were important for 2 reasons, the first being that it gives us area under the drone to build from and had room for our package. The second reason was that with landing and taking off the thing our drone was going were hitting the most was the ground so having landing should improve the longevity of the drone. The con to this drone was that we don't knew how stable the landing gear was on uneven terrain, as well as not having guards for the blades.

The FPVKing 500-X4 was the 3rd drone we looked at as seen in **Figure 12**. The FPVKing 500-X4 was the largest drone we looked at being 500mm in size. The pros to this drone were not only the size that gave us the building freedom we needed but also the arms of it being able to fold down or came off which was handy in transport as well as any un foresee maintenance that might needed were done to the drone during testing. The cons for this drone were similar to the ones found in The Readytosky S500 with the same concerns about the landing gear being unstable on uneven ground. If we looked into **Figure 10**, we saw the physics that we needed to apply and looked for when looking into frames.

Figure 10: Physics Looking into Drone Frame



Frames was the foundation of our project because when trying to divide and conquer the decision of parts it was quickly seen that making a decision on one part led to another one. This put a stopped to trying to make a lot of decisions at once and stopping to evaluate more the decisions being made. All of these frames were leading to what type of motor we would eventually pick and other components because a heavier frame means having a more powerful motor that could power it. It was important for our frames to have a landing gear attached to it whether we were going to had to attach our own landing gear or look for frames with it already attached. We ended up doing both in this case as our initial drone had no landing gear while our second had legs. Some pros to having the landing gear attached means that we would not have to buy others or made the landing gear, then there was the issue of securely attaching the landing gear. This was a con when approaching **Figure 8**, but then the cons with having landing gear was that if it were to break, we might have had to replace the whole frame.

Replacing the whole frame would not be very cost efficient to the project and means more testing would have had to happened in order to made sure the drone would not crash and break the landing gear. A delivery drone specifically also needed were able to handle the weight of the components and the potential packages. Smaller frames might not be able to handle all of that and led to looking at frames that match a certain size versus a certain shape.

Figure 11: The Readytosky S500

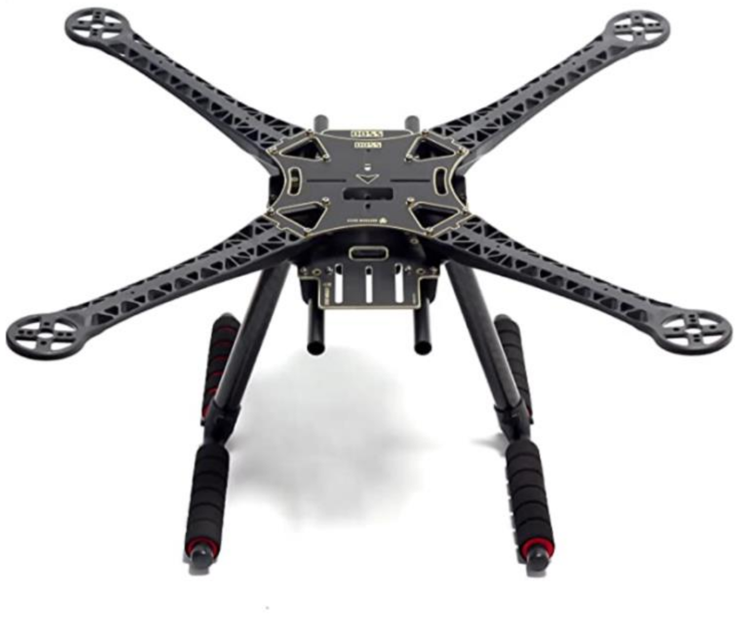
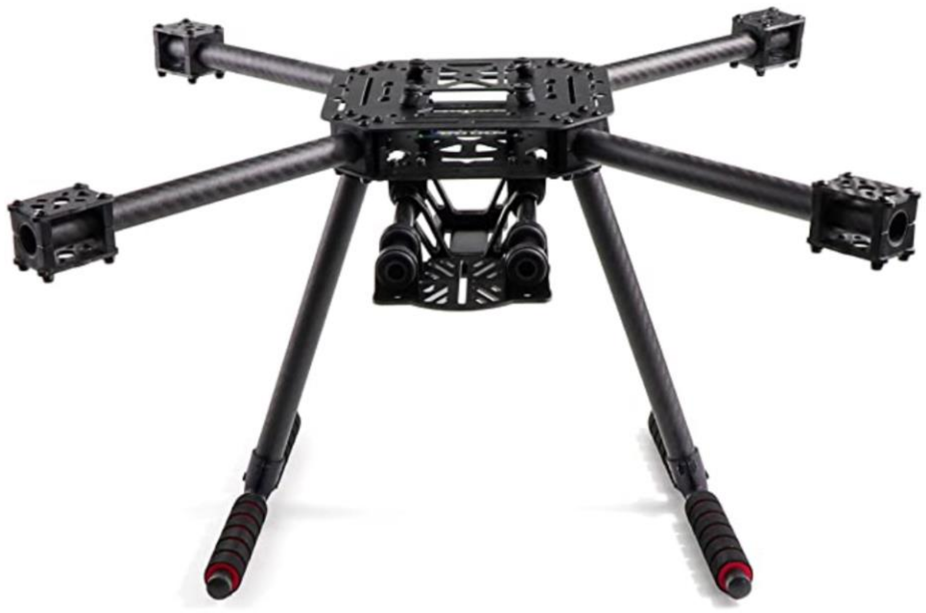


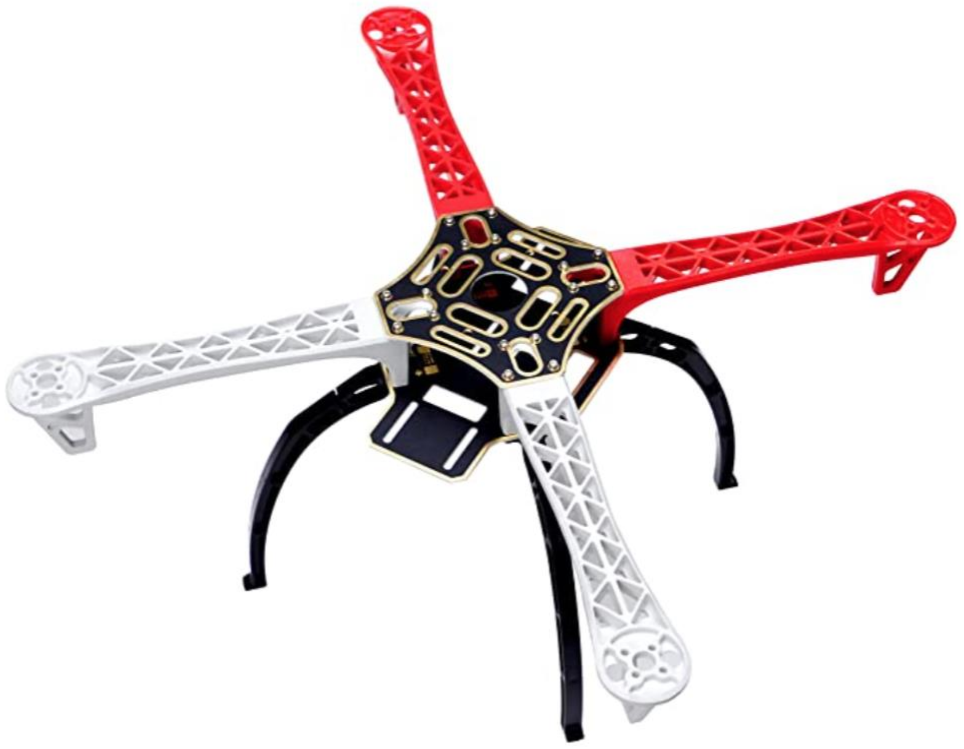
Figure 12: The FPVKing 500-X4



The YoungRC F450 was the last drone we looked at and was as seen in **Figure 12**. The YoungRC F450 was 450mm in size and so we believed it gave us the building freedom we needed. We also thought the landing gear on the YoungRC F450 was better suited for multiple terrains while still giving the space underneath for creative freedom. The only cons for this drone were that it didn't look to have the largest amount of space in the center as much as the other drones offered as well as not having blade guards.

In **Table 6** below, we compare the drones in a side-to-side manner and looked at the price differences as well materials used in the frame. As we compared the drone's side by side in the table above, we quickly found that The YoungRC F450 was the most cost-effective drone. We then also came to the conclusion that we would like our drone were made out of plastic rather than carbon fiber for two reasons, the first being that carbon fiber was heavier than plastic which would have held us back from the maximum amount of lift and weight of the package. The second reason being that was a part where to break or chip in testing there was a very limited number of repairs, we were able did because when carbon fiber breaks it does break cleanly, it splinters out making it impossible to repair. So as to reiterate, this drone had the highest potential not only because of its size and performance in certain sectors that could affect our testing and eventual completion of our project, but also the direct cost to the project budget.

Figure 13: The YoungRC F450



From all this research we were able to pick The YoungRC F450 in **Figure 13** as our drone frame for the following reasons. First, it was 450mm which we believed was large enough to carry a package while giving us the freedom to place the sensors as we saw them. Seconded was that it was made out of plastic and was lighter than the other drone frames and was easier to repair if damages arise. The third was the landing gear being able to withstand a wider range of terrain while also giving us the needed space underneath to build as we needed for our sensors, battery, and package placement. The fifth reason was the cost being more than half the cost of the other drone frames. This series of reasons were why we believed The YoungRC F450 was the best drone frame and makes up for not having guards for the blades and would still be the foundation on which we build upon.

Table 6: Drones Frames

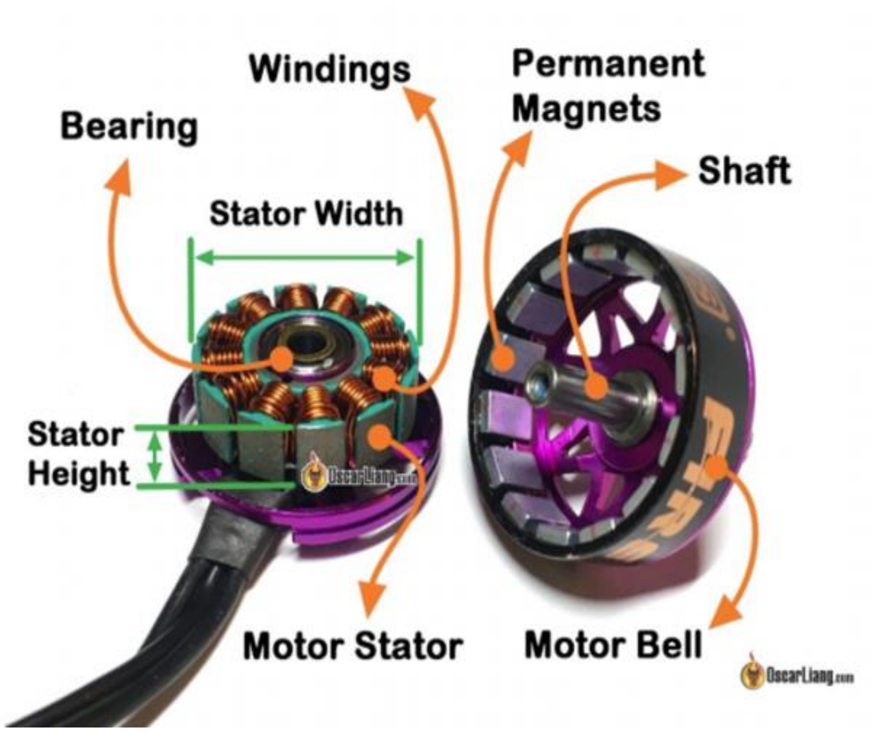
Names	Cost	Size	Materials	Landing gear
Readytosky 149HD	$46.00	149mm	Carbon Fiber	No
Readytosky S500	$47.00	450mm	Plastic/Carbon Fiber	Yes
FPVKing 500-X4	$75.00	500mm	Carbon Fiber	Yes
YoungRC F450	$21.00	450mm	Plastic	Yes

3.2.2 Motors and Propellers: Stators and their relation to Lift

Our motors were vital aspects of our drone and needed were calibrated to our specific needed. However, when researching motors, we found that it very much was a chicken and egg scenario where we needed to also discover the size of our propellers and base that on stator size.

We knew our frame size was going to be at least 450mm wide with wingspan included, so when we took that into account, we could begin to estimate what other properties of our drone would come to. With Guidance from **Table 7** from DIY makers of drones, and the physical picture we had of the inner working of a drone motor from **Figure 14** we could correspond to what we needed for our motor needed and began the tiresome process of achieving the right motor for the right kind of lift and mission we would need to acquire the motor for. Motors needed to account for the numeric and specifications that came from the frame, which shows how much motor power was needed.

Figure 14: Inner Components of Motors



Our motors led to more decisions being made because the ESCs correspond to the motor power also. The table goes into how when the frame size changes the motor power, and such also increases or decreases according to size. So, picking the right one was important to help the drone to takeoff with enough burst power and stay steady. We actually came to a problem in our motors where they weren't durable enough for sustained testing because of our prototype status, crashes happened a lot. To showed this we could looked below at the many different instances and kinds of motors we would need to consider for our future drone and possibly any future adjustments that would needed were done to our motor's specifications because of future changes to actual environmental factors that we cannot perfectly saw now and would not be able to account or until such a time that we had all other components that our motors would had to lift so as to perfectly knew environmental factors.

Table 7: Propellor and Motor Size Comparison

Frame Size	Prop Size	Motor Size	KV
150mm or smaller	3" or smaller	1306 or smaller	3000KV or higher
180mm	4"	1806	23600KV
210mm	5"	2204-2206	2300KV-2600KV

250mm	6”	2204-2208	2000KV-2300KV
350mm	7”	2208	1600KV
450mm	8”,9”,10”	2212 or bigger	1000KV or lower

We saw from this that since our frame size was at the very edge of the Graph (450mm) we would have to supply propellers of 8” or higher on to our motors, with a stator size of 2212 or higher, which is one of the reasons we agreed on the A2212 from QWinOut that we decided on. The number 2212 was the technical language used by most drone manufactures when discussing motor and stator size. The number 2212 was broken into two sections that were described with the first two digits being the stator width, while the last two digits were the stator height. The higher these two numbers were meaning the more torque one motor could supply. We decided after comparing the cost to the amount of material we were buying what motor went, shown in comparison in **Table 8**:

We eventually picked the Qwinout motors as the proper motors to used, not just because of the cost for only the motors, but because of the package, we had where we could buy both the ECS, propellors and the frame, thus keeping cost down, even when the weight of the Qwinout motors was more than all other motors of the comparison.

Table 8: Comparison of Different Motors

Name	Current	Weight	Price
Powerday	4-10 Amps	47g	$32.69
Goolsky	4-10 Amps	48g	$49.99
Readystosky	4-10 Amps	190g	$36.99
Qwinout	4-10 Amps	249g	$26.88

3.2.3 Batteries and Power Distribution Boards:

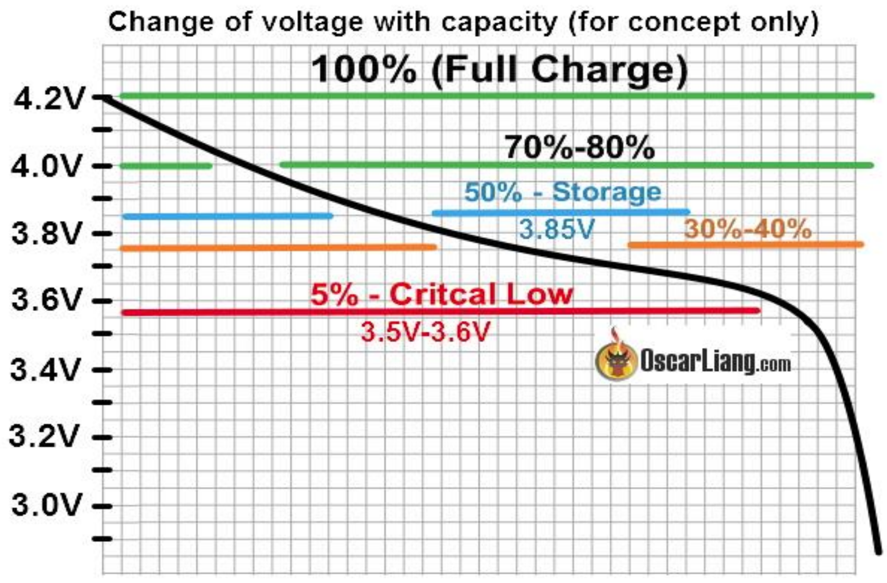
We knew from our research into drones that almost all drones fly using the Lithium Polymer battery pack. Because it had a higher energy density and lighter weight than other battery types. However, we also needed to decide on what type of battery level we would use as the battery was again dependent on what types of components it would need to be able to power. In the future, we would expect with future iterations that we would need more power or possibly two in series of the same kind. Namely how much current the motor and ESC was using during maximum operation. Each cell of a Lithium Polymer battery included 3.7 Volts and was said were 1S, with 2S being 7.4 Volts, etc... However, if we saw in **Figure 15**, this was actually the average usually given for that specific cell with the top of the fully charged battery was around 4.2V. Now, if we chose an ECS that drew 15 Amps for operation, and knew the amount of time we wanted our drone to ran per battery pack, then we took the C rate and battery capacity and used this equation in **Table 9** to find the numbers needed:

Table 9: C-Rating Equation

Max Current Draw = Capacity * C-Rating

So, taking the current draw of 15 amps and times it by the amount of time required we got 15 times (10 divided by 60 to account for minutes in an hour) and that would give us 2.5 Amps per hour. This was the amount the battery needed to supply. Using this number, we could then find the C-Rating and Discover the battery that would fit our needed. Now to also went over the storage of this battery type, it needed were more well maintained for the battery were usable, as shown in **Figure 14**, the storage charge that most LiPo batteries needed were at was around 50% to safely be stored for further used in the field or for travel. We also knew that from our mission of avoiding obstacles and delivering a package to a destination, we needed enough battery life for this initial model of the drone to last at a minimum of 10 minutes. Now if we took the max current draw of our ESC that we had decided upon, namely the 2-4S 30A Brushless ESC, and took our known timeline we needed, ten minutes, and divide those ten minutes by 60 mins, we got 0.167 hours, which was the time we needed. We're going to convert this was mAh, however, so we needed to take the max current draw, which was 40 amps when under “burst” conditions, which was when the drone was doing many different things and needed to move the motors fast, and then times that by our 0.167 Hours. We, of course, convert this to milliamps but afterward, we achieve our mAh we needed to know. We saw we achieve a mAh’s of 6680 mAh. Further, we could now find our needed C rating so we may find the battery we needed for our timeline and drone type. We ended up not achieving the exact power output we wanted as we could not test under complete conditions that may occur because of our crashes during testing not giving us accurate timing of system requirements.

Figure 15: Concept of Charge Levels of Cells and Storage Levels



C rating could also be said were the charge rate for the batteries, or how long it would take for them to discharge or charge, with 1C being equal to 1 hour or 60 mins. 2C being equal to 0.5 Hours, or 30 mins and so on and so forth. If we wanted a more efficient testing period, in other words, if we don't want to have to buy numerous batteries to also had them available for testing, we had at least a C rating of 1C, with a more robust route was to have 2C for a 30 min charge rate as shown in **Table 10:**

Table 10: Difference in C-Rating

C Rate	Time
2C	30 mins
1C	1 hour
0.5C	2 hours
0.1C	10 hours

Now when comparing real-world batteries and discovering what we needed and what could be affordable, we would use **Table 11** to compare different batteries and their prices.

Table 11: Different LiPo Batteries on Amazon

Brand Name	Voltage level	C-Rating	Capacity	Case Style	Weight	Price
SIGP x1	11.1V	25C	2250 mAh	Hard Case	180g	$12.60
Turnigy x1	11.1V	20C	5000 mAh	Hard Case	360g	$24.72
Zeee x1	11.1V	50C	5200 mAh	Soft Case	334.7g	$33.99
Zeee x2	7.4V	80C	6000 mAh	Hard Case	309g	$49.99
HRB x2	11.1V	55C	7000 mAh	Hard Case	449g	$103.99
Zeee x1	11.1V	100C	8000 mAh	Hard Case	466g	$68.99

We saw from comparing our different categories that in between these 5 different batteries, the ones with our needed were only the 7000 mAh HRB and the 8000 mAh Zeee. We knew that we do not need above 8000 mAh for our operation time however if the price range was acceptable, it was still within our market as this would just give us more operation time overall. However, we saw that for the price given for the HRB we got 2 battery packs, which cannot be beaten by Zeee even in our price range, showing that we were giving very much consideration to the pros and cons of higher mAh or cheaper price.

Now for PDB or Power Distribution Boards, they could range from the most basic where you had what amounts to copper plating that integrates the power from the battery to the components like the ESC or the flight controller. This would not work for us our sensor package would require a power module converter as shown in the basic PDB in **Figure 16**:

This shows how the basics of the PDB could be simplified immensely. To continue to our needed for this project, however, we were including voltage regulators in the sections flowing to different outputs in the PDB so we may control how much voltage goes where, as the sensor package would need a certain voltage with much "noise" and the ECS might needed the voltage regulators to quickly increase power intake when sudden accelerations or decelerations happened. During the final implementation of the Power module we designed, our parts were not available, but we did show correct current flow in the circuit lines.

For when it came to designing or PDB otherwise known as power distribution board we had a couple of options in terms of software to design on. The software being Eagle or Kicad. We chose Eagle over kiCad for a few reasons. We believed Eagle were more widely supported if we had to asked questions when finding complications in our design. We had also already taken a class with Eagle and had some level of familiarity. Eagle had an iOS version as well windows version, so the program was accessible to everyone in our group.

While kiCad did seemed easy and intuitive the fact that it was not assessable where IOS (Apple devices) made it unusable. The software we decided to design our power distribution board on was Eagle as you could saw this information more clearly in the graph below. We needed to verify and be very insistent with the specifications of the PCB software we would need to use and implement in our design as this could affect all systems within our design, as power needed to run into every component and without efficient flow, we had catastrophic effects such as crashing or missing data points that could led to our drone missing instances if objects coming towards it. Which did lead to catastrophic crashes and the destruction of parts of the drone.

Figure 16: Basic PDB Module with Converter between Battery and PDB

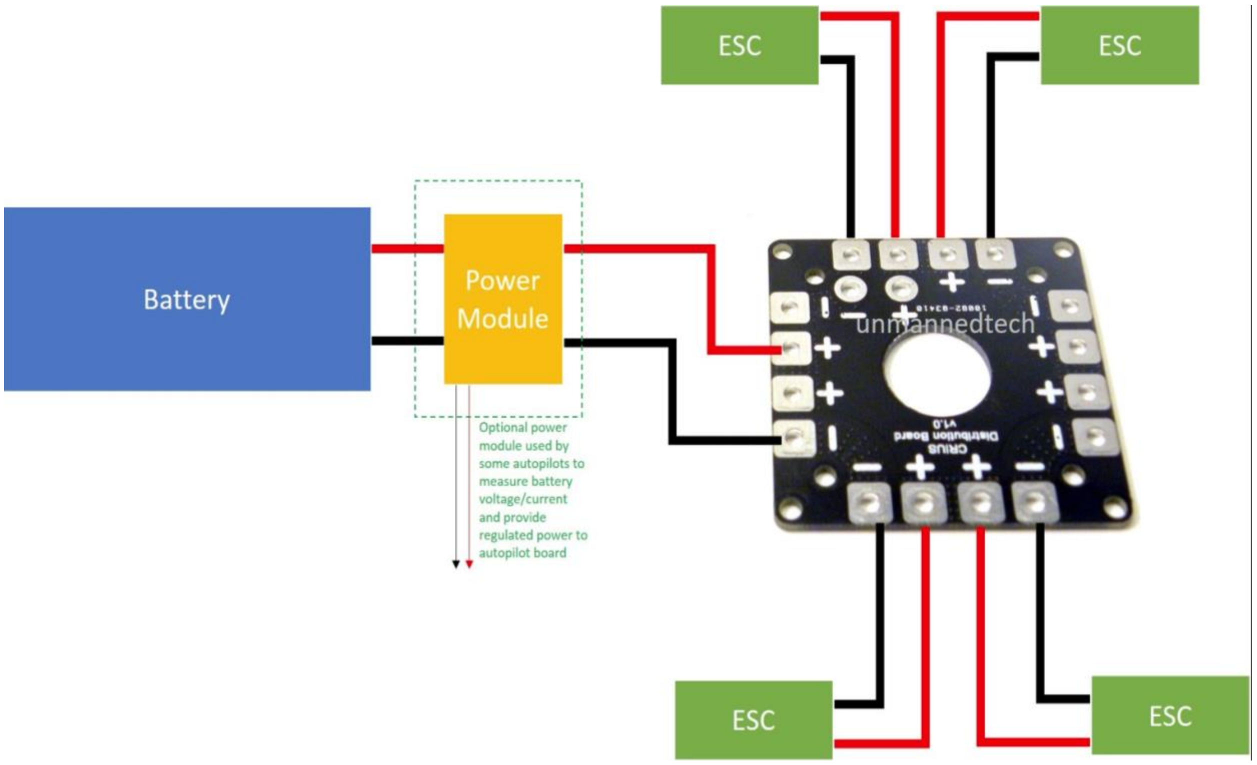

ESC
ESC
Battery
Power Module
unmannedtech
Optional power module used by some autopilots to measure battery voltage/current and provide regulated power to autopilot board
ESC
ESC

Table 12: PCB Software Design Comparison

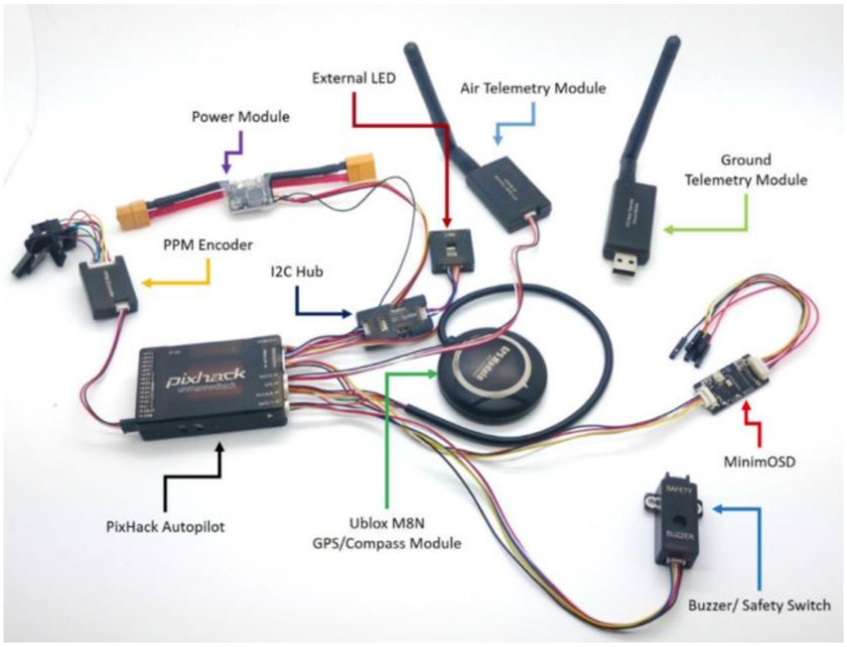
PCB Software			
	IOS	Window	User-friendly
KiCad	FALSE	TRUE	TRUE
Eagle	TRUE	TRUE	TRUE

Now the mission of the PDB was to distribute the power from the battery safely and effectively. The loads the battery would need to power directly was our ECS and our flight controller and not to forget our data processor, the Raspberry Pi. Our ECS being a 2-4S Brushless ESC and being rated toward 3 Amps. The flight controller was the other important Device the PDB would have to deliver power to safety and effectiveness. The flight controller was the brain of the drone and was the most expensive part. So, making sure we deliver the correct number of voltages needed was critical. The voltage needed to power the flight control was 5 volts. To make sure we do this, we first compare the two in **Table 12** we would use the TI web bench and upload the correct Library into Eagle that would allow the correct voltage output. While dealing with our limited space on the drone for placement and had the correct milled out hole to fasten our power distribution board to the drone frame. We actually ended up using models designed by Webench for this like we predicted but for our Power Distribution Board specifically we actually used CAD to design the correct dimensions needed for the odd angles and power routes we had to deliver too.

The next part of our PDB design was to have it manufactured and ship to the United States. There were many companies that would manufacture such a board and we don't expect this to have a strong impact on our budget. However, the limitations on global trade in this current climate might after shipping times and prices which could possibly affect where we purchase our manufactured board. We decided on JLCPCB as our manufacturer but even then, we didn't get our required components.

3.2.4 Autopilot systems and their relation to GPS systems:

Figure 17: Components of a Basic Autopilot System



For Autopilots, they included many components that we were going over such as shown in **Figure 17**, to include the basic components like the telemetry system, the power module that regulates power for the autopilot system, and the compass module. This system was made were in parallel with a manual system that all drones came with and the autopilot system itself was controlled from a “ground station” that was usually a computer with the ground telemetry system attached to it so the Drone and the Ground station could talk to each other. Now for our instance of used in the term for ground control, we would most likely be relying entirely upon a program that would act faster than anything we could possibly do with human error and reflexes. The manual aspect of the drone, or the manual control in other words is needed only when all else has failed and a crash is imminent because of weather conditions or other situations.

The Autopilot runs on a processor that runs separate front the flight controller and was usually a 32-bit processor although there were 8-bit ones, we used a Raspberry Pi and Navio for our purposes which we knew would handle the processing requirements of our specific style of Autopilot. We used many different sensor systems that we included in our drone such as the barometer to measure pressure, while drones usually came with accelerometers and gyroscopes, and the autopilot system itself would need a compass module and a GPS module that needed were far enough away from the drone so as to not interfere with the electronics of the boards. This was accomplished through a mount added to the GPS Module. Now, these sensors were known in drone manufacturing as DOF or “Degrees of Freedom”. With more DOF you got more sensors included in your drone. While we were buying drones that do not came with many sensors, we did add many sensors to artificially raise our drone's DOF. To use GPS modules as we will talk about later in the section, a drone needed at least 10 DOF. To continue onto our telemetry modules, these spoke to each other through RF frequencies, and both the drone and ground modules needed were the same MHz to worked properly. From there we again talked about our power module converter, which was a module that was in between the lines from the battery to the Power distribution board so there was less “noise” in the power, as we would need that for working with GPS signal and sensor modules. Most power module converters put out a stable 5V from the battery. This also lets the GPS took over for flight control when the battery gets low as with the ability to acquire more power with “clean” signal strength, the GPS could tell the ground station when it was low and would return to base. This later had to be adjusted to make the landing protocol for this less hazardous for the drone. Finally, there was instances of loss of signal for the GPS system as this was inevitable when going through the thick bush or other obstacles. This was where the optical flow mechanism would come in, where we used machine learning and computer vision that was talked about later in this paper to “drive” the drone when needed.

3.2.5 Flight Control Boards

The flight control board was the single component that integrates all other individual components into a coherent system. Its overall purpose was to take in inputs from the user, a predefined waypoint mission, and various sensors in order to translate these signals into the proper motor speeds to achieve stable flight. This control board was heavily coupled with the flight control firmware. Together, these two elements determine the capabilities of the drone, which could vary wildly depending on the feature set of the board-firmware pair. On the low end, the control system only tries to hold the drone level and varies the motor speeds according to user input for steering. On the high end, the control system could autonomously fly the drone using GPS data and waypoint missions. Some could even perform computer vision processing. Because our application requires autonomous flight, the high-end flight control boards were more appropriate. Given that this was the keystone component of our system, it was crucial that we select the optimal control system for our application. **Table 13** compares flight control boards and displays the features we considered when deciding on one. We saw the weight difference was extreme between different boards, but the firmware was incredibly similar systems to each other. Other things that were similar were Bus width, with many being a 32-bit variety because of the influence of cost that of course comes with boards that were trying to achieve the highest processing power with such small sizes.

Table 13: Flight Control Board General Features

Board	Price ($)	Weight (g)	Firmware	OS	Popularity	Bus Width
Pixhawk	80	60	ArduPilot, PX4	No	High	32 Bit
Pixhawk Cube	250	150	ArduPilot, PX4	No	High	32 Bit
Navio2 + Raspberry Pi 4	168 + 35/55/75	23	ArduPilot, PX4	Linux	Medium	64 Bit
BeagleBone Blue	80		ArduPilot, PX4	Linux	Low	32 Bit

Table 14: Flight Control Board Hardware Features

Board	Processor	RAM	Ports	Sensors	Networking
Pixhawk 1	180 MHz ARM® Cortex® M4 with single-precision FPU	256 KB	I2C, CAN, ADC, UART, CONSOLE, PWM, PWM/GPIO/PWM input, S.BUS/PPM/SPEKTRUM, S.BUS out, microUSB	gyroscope, accelerometer/ magnetometer, accelerometer/gyroscope, barometer	Wi-Fi, DSM-X

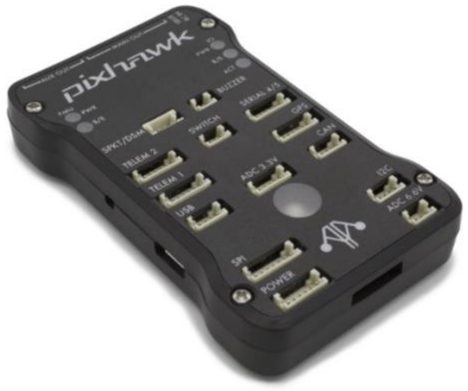
Pixhawk Cube	180 MHz ARM® Cortex® M4 with single-precision FPU	256 KB	Same as Pixhawk 1 but more UART ports + UART HW flow control	gyroscope, accelerometer/ magnetometer, accelerometer/gyroscope, barometer	Wi-Fi, DSM-X
Navio2 + Raspberry Pi 4	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	2GB, 4GB or 8GB	UART, I2C, ADC, 12 PWM servo outputs, PPM/S. Bus input, HDMI, MIPI display port, MIPI camera port, microSD, USB3.0, USB 2.0	Accelerometers, gyroscopes, and magnetometers, Barometer	2.4GHz Wi-Fi, Bluetooth, BLE
BeagleBone Blue	Octavo Systems OSD3358 1GHz ARM Cortex-A8	512 MB	4x UART, 2-cell LiPo, 2x SPI, I2C, 4x A/D converter, CAN bus (w/ PHY), 8x 6V servo motor, 4x DC motor, 4x quadrature encoder, USB 2.0 480Mbps Host/Client Port, USB 2.0 Host Port	9 axis IMU (access, gyros, magnetometer), barometer, thermometer	2.4GHz Wi-Fi, Bluetooth, BLE

Our first key consideration was the available flight control software supported by a particular board. Because the control firmware and control board were so interconnected, this feature would have the most significant impact on the flight control board, so we decided to invest in it firmly. This firmware was discussed in a later section, but for now, it was sufficient to knew that we narrowed our options down between PX4 and Ardupilot. The weight of these boards was very similar and only accounts for a small percentage of the overall drone, so that was not a big concern. All of the boards discussed in **Table 14** had autonomous waypoint mission capabilities, which was a requirement for us. Our next biggest consideration was the supported operating systems. Many of the options do not even ran operating systems; the flight control firmware runs on bare metal. However, the BeagleBone Blue and Navio2 both support Linux. This could be extremely helpful because it has allowed us to run external applications in addition to the control firmware such as the computer vision software. This was beneficial but not strictly necessary. Another consideration was the popularity and community around each of these boards. Having a large, active community developing solutions on a particular board not only validates its merit but also provides resources for us to learned from and asked questions during the development process. Additionally, it was important to appraise the processor on the board. Given that the options we considered were all supported by the firmware we chose, we knew they was powerful enough to ran that. However, having extra power would allow us to run other software like the computer vision algorithm or sensor fusion programs but these weren't realized at the end of the project. A final consideration for us was the price. Keeping the overall cost of the system low was a key functional requirement for our project. This board was one of the single most expensive components and as such, price was important to consider.

3.2.5.1 Pixhawk

The Pixhawk was a very capable flight control board. It supports both PX4 and Ardupilot. This gave us options when deciding the flight control software. It does not run an OS which forced us to use an external processing board to run the computer vision algorithm and translate the information it gathers into inputs the Pixhawk could understood. With that being said, its popularity was very high, and it had an active community which could be very helpful if we ran into problems with hardware integration. The Pixhawk had a relatively powerful ARM Cortex M4 processor and 256 KB. This computing power was useful in processing sensor data such as ultrasonic sensors and lidar. While we needed an external computer vision board as stated previously, splitting the sensor processing between them was beneficial to balance the workload. As seen in **Figure 18,** the variety of ports on this board gave us options for connecting to various external sensors as well as other processing boards. One key benefit of this board was in the networking capabilities. While all of the Flight Control Boards do had Wi-Fi capabilities, this particular one supports DSM-X. This was a protocol that allows communication from a radio control transmitter on the ground to the drone itself. This was not strictly necessary because our drone was flying autonomously, but it was a good safety precaution if the drone starts behaving unsafely and looks like it would crash. Having the ability for a human operator were able to take over control could not only save the drone from damage but could also save team members from injury. In terms of price, this was one of the cheapest options.

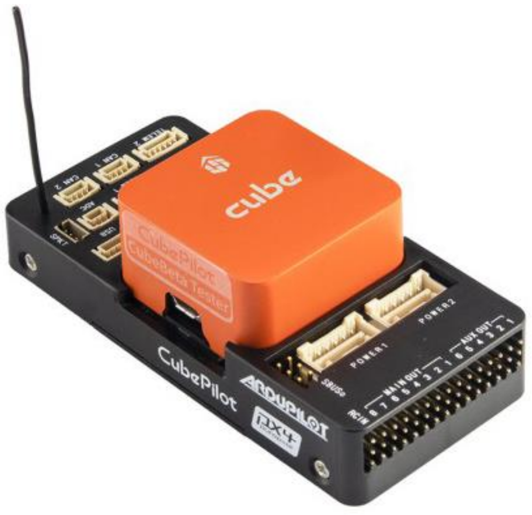
Figure 18: Pixhawk 2.0



3.2.5.2 Pixhawk Cube

The Pixhawk Cube was extremely similar to the original Pixhawk. It also supports ArduPilot and PX4 for flight control firmware with no underlying OS. It had the exact same processor, memory, and integrated sensor combination. Again, the benefit of having DSM-X capabilities applies. One of the only differences in hardware was that it had more UART ports, each with higher bandwidth, as seen in **Figure 19**. The biggest difference between the prices. The Cube was $250 as opposed to $80 for the Pixhawk. The extra ports do not seem worth the huge price increase, so for that reason, we had decided that the Cube was not the best board for our used case.

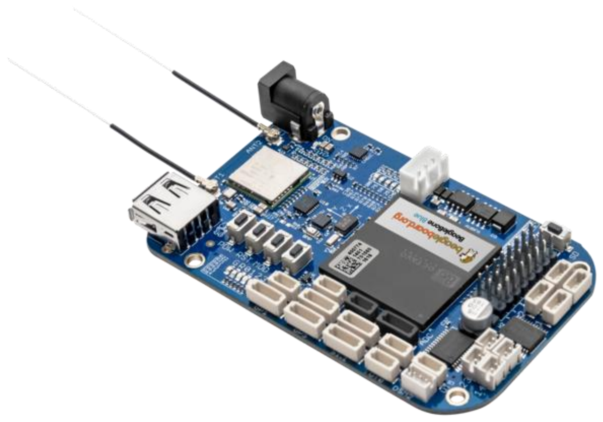
Figure 19: Pixhawk Cube



3.2.5.3 BeagleBone Blue

The BeagleBone Blue was different from the previous two boards in that it runs a Linux-based operating system. This allowed us to run other programs concurrently with the flight control firmware. Both PX4 and Ardupilot were capable of running on top of the Linux kernel. The processors on this board were significantly more powerful than both versions of the Pixhawk. In addition to the single-core ARM Cortex A8, there were two Programmable Real-time Units (PRUs). These PRUs handle the I/O signals for interfacing with peripherals. This takes the load off of the main processor, freeing it up for user applications. While this processor was probably fast enough for sensor processing and light computer vision applications. However, it may limit us in terms of the number of sensors we could process and the speed at which the computer vision runs. This board had 512 MB of memory which was useful when running heavier applications like the ones mentioned previously. This board had all of the sensors needed to achieve stable flight, including accelerometers and gyroscopes. Additionally, it had a magnetometer to maintain heading, a barometer to detect relative altitude, and a thermometer. The BeagleBone Blue supports Wi-Fi, Bluetooth, and BLE. Unfortunately, it does not support DSM-X which means we were unable to use a human-controlled transmitter to intervene in the case of an emergency. Bluetooth or Wi-Fi could be used to send signals to the drone during flight, but it was not a good recovery option in the event of an imminent crash. As seen in **Figure 20,** this board had an extensive set of I/O ports such as 4 UART, 2 SPI, I2C, and many PWM outputs to control servos and motors. It also had USB 2.0 for connecting to the ground station or an external board. This became useful when we determined that we needed more processing power. At $80, this board was a very powerful option. It was the same price as the Pixhawk, but it runs Linux and had a significantly more powerful processor. The only thing that was missing was the DSM-X receiver, but this could be easily purchased and integrated separately from the control board.

Figure 20: BeagleBone Blue



3.2.5.4 Navio2

As mentioned previously, more demanding programs such as sensor fusion and certain applications of computer vision may require extra processing power in the form of an external board. The processing board was required to communicate with the flight control board in order went sensor data as well as to send control inputs. The Navio2 was probably the most unique board in this respect because it was not a fully functioning standalone board. It was a daughterboard intended were attached to a Raspberry Pi. This option drastically simplifies the connection between the control board and processing board. The board had connections that fit directly onto the GPIO pins of the Raspberry Pi, as seen on the top left side of **Figure 21.** This would also increase the capabilities of our drone. The Navio2 itself contained all of the features you would expect from a flight control board that you don't get from the Raspberry Pi itself. This included the onboard sensors (accelerometers, gyroscopes, a magnetometer, and a barometer). It also had an M3 coprocessor used to process PWM signals from the sensors. As with the BeagleBone Blue, this frees up the main processor, in this case, the Raspberry Pi, for more computationally intensive tasks. The difference was that the Raspberry Pi's processor was significantly more powerful than the BeagleBone's. It was a quad-core rather than a single core, 64-bit versus 32-bit, and runs at a higher clock rate. Additionally, the Raspberry Pi had between 2 GB and 8GB of memory which was particularly necessary for imaging processing applications like robot vision. In terms of I/O, this was one of the better options. Six of the Raspberry Pi's GPIO pins were left unused by the Navio2. This gives us an additional 6 UART, SPI, or I2C input on top of the 3 ports included on the Navio2. The Raspberry Pi also had four USB ports that could be adapted to any of the other protocols. This gives us tons of options in terms of connectivity which allowed us to add in more sensors or different sensors during the development phase. However, we had the problem of still not enough connections. The Pi had just enough pins for everything we had to connect except the Ultrasonic, and no other options for other sensors with the pins we had left. Just like the BeagleBone, this board combination supports Wi-Fi, Bluetooth, and BLE. It does not implement DSM-X, but as mentioned previously, that could be added with an external receiver. Another benefit of this solution was the low power requirements of the Raspberry Pi 4. It typically only draws 3W versus the 10W which the Jetson Nano board consumes. Power consumption was something to consider when deciding if we got an external processing board as this drone runs on a battery. As far as I saw, the only downside of this flight control board solution was the price. It seems like a much better version of the BeagleBone, and the price reflects that. It was between $200 and $250 depending on the model of Raspberry Pi we choose. This was around $60 more than the BeagleBone Blue paired with an Nvidia Jetson Nano configured similarly. Because the flight control board and computation board were interdependent, we needed to consider them as a pair when making decisions. With that being said, the extra reliability and ease of communication between boards makes the Navio2 seemed like the best option for a flight controller board.

Figure 21: Navio2

3.2.6 Data Processing Board

The purpose of the data processing board was to take in all sensor data, process it, and then output drone commands. As mentioned previously, flight control boards only had very weak coprocessors which were used to process onboard sensor data and ran flight control algorithms such as stabilization. In order to integrate additional sensors such as ultrasonic or infrared rangefinders and process the data in order to achieve autonomous navigation, additional computational power was required. Since we plan to implement a robot vision-based navigation algorithm, our system would require a relatively high-powered data processing board.

Recently, powerful single-board computers had risen in popularity for robotics applications. Being powerful but small and light makes them perfect for our application. Just like the flight control board, this was one of the key components of our system, so it was important were diligent in choosing the very best options for our application. One of the main considerations for us was computing power. The CPU and GPU processing capability was what determines the speed of our obstacle detection algorithm. Running slower could be the difference between avoiding an obstacle and crashing. For that reason, we only looked at units with high computing power. On a related note, our particular application of computer vision was a memory-constrained task. This means that more memory was a significant benefit that would realize serious performance benefits. We only considered data processing boards with a minimum of 2GB of memory, but more was better. Connectivity was another big concern for our board. We needed were able to connect every sensor onto the board, as well as connect it to the flight controller. This means that we needed a certain number of GPIO pins in addition to the other I/O methods on the board. Certain solutions integrate with the flight control board much easier than others. This was something to consider as spending less time on the flight controller-data processor interface allowed us to spend more time on algorithm optimization and additional feature development. Being part of a flight platform that runs on batteries, this board needed were lightweight as well as efficient. A data processing board that uses too much power would decrease the flight time which was a trade-off we had to consider. From a programming perspective, software support was something we should consider when choosing a board. The ability to program with high-level languages or even ran an operating system was something that could drastically increase the ease and speed of the software development process. The ability to support opened-source libraries was even better. On a related note, having a community around the processing board allowed us to adapt solutions from other contributors as well as ask questions to experts when we had trouble. The final consideration for our data processing board was cost. With our overall goal of keeping the drone inexpensive, this was important to consider.

As mentioned in the flight controllers' section, the data processing board and flight control board were very interdependent and thus should be considered together. The two options we had for integrating them into the system were a separate flight controller-data processor pair and an all-in-one solution. The option that we choose could determine the data processing board that we went with.

3.2.6.1 Nvidia Jetson Nano

The Nvidia Jetson Nano was a very small but very powerful single-board computer. As seen in **Table 15,** it had a 64-bit Quad-core ARM CPU, an NVIDIA Maxwell GPU, and either 2GB or 4GB of memory. These components made it a very powerful computer, especially for image processing and AI-related processing. As far as connectivity, the Jetson Nano had 40 GPIO pins, 4 USB 3.0 ports, and a MIPI CSI port for a camera. This was important because the Jetson Nano does not have the capability to be part of an all-in-one solution. We were required to connect it to all of the sensors and to the flight control board on our own. Since all of the GPIO pins was opened, this was plenty of connections for our used case.

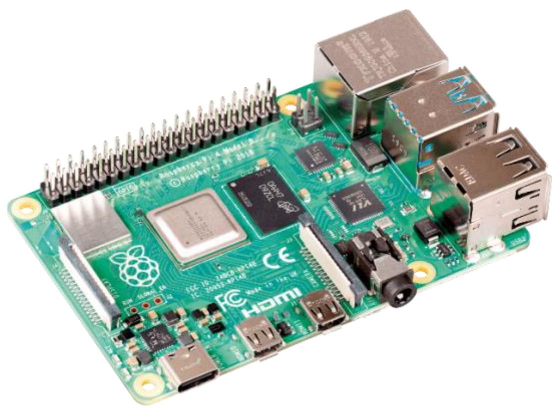
Figure 22: Nvidia Jetson Nano

At 250 grams, the Jetson Nano was relatively lightweight and would not made a significant impact on the final system. One of its biggest flaws, however, was its power consumption. This board could use between 7.5W and 15W under load. This had the potential to impact the efficiency of the drone and decrease the flight time. Additionally, this much power requires a relatively large and heavy heatsink, as seen in **Figure 22.** Being a full-fledged computer, supported software was not much of a concern. It could run Linux and any software on top of that. The Jetson Nano does have an active community, though it was smaller than the Raspberry Pi 4's. Thus, the topics were less diverse and may not be as applicable to our application. The price was around $60 for the 2GB version and $90 for the 4GB option. This was more expensive than the Raspberry Pi 4 on its own, but it was important to consider the price in conjunction with the corresponding flight controller. After doing this the system with the Jetson Nano could be between $140 and $170.

3.2.6.2 Raspberry Pi 4B

The Raspberry Pi was very popular for robotics projects, and for good reason. The latest version, the Raspberry Pi 4B, was extremely powerful and very efficient. It had a 64-bit Quad-core ARM processor, integrated Broadcom GPU, and either 2GB, 4GB, or 8GB of memory, seen in **Table 16.** These specifications allowed it to tackle many of the computational tasks we needed it to. The board had 40 GPIO pins, 2 USB 3.0 ports, 2 USB 2.0 ports, and a MIPI CSI port for a camera. At this point, it was important to consider how we would connect the board to the flight controller. Like the Jetson Nano, the Raspberry Pi could run independently and communicate with the flight controller via communication standards like UART or I2C. However, we also had the option to connect it directly by using the Navio2. This would drastically simplify the connection between the flight controller and data processor, as well as made communication between them easier. The software for both units was running on the Raspberry Pi 4B. While this would made integration easier, it would slow down computation as well as took up some of the GPIO pins. In fact, the Navio2 took up 34 GPIO pins, leaving only 6 for the sensors. Many of these pins was utilized by connecting the board to an external flight controller anyway, so this was not much of a drawback. This became a problem later when trying to tie in other sensor packages, however.

Figure 23: Raspberry Pi 4B



Also, it was important to note that with the 6 remaining pins and 3 additional I/O ports on the Navio2, it was possible to connect multiple sensors to each. This means that I/O availability was not too much of a concern. Where the Raspberry Pi 4B really shines was in the airframe-specific specifications. As seen in **Figure 23,** the Raspberry Pi was significantly smaller than the Jetson Nano. It weighs only 45g, which was significantly lighter than the Jetson Nano. It also only consumes between 2.5W and 5W of power. Together these two attributes would contribute to a very efficient system which could improve our power efficiency and flight time. As a single board computer, the Raspberry Pi ran Linux as well as any software that was compatible with Linux, which was very important during our implementation phase. This Board had a very active community, especially for related robotics projects. This could be very helpful in learning from other’s experiences and making connections and helped when needed. Additionally, since the Raspberry Pi 4 was backward compatible with software for all previous versions of the Raspberry Pi, there was a huge backlog of projects that we could learned from. Even though it was a very capable board, it was still inexpensive. The 2GB version was only $35 and the 4GB version was $55. Being able to go all the way up to the 8GB version for an even lower cost than the 4GB Jetson Nano gives us even more flexibility when it comes to choosing our configuration. However, that was not the whole picture. Going with the discrete data processing board, this unit was cheaper than the Jetson Nano, but not when implementing the all-in-one flight controller-data processor system. Combined with the Navio2, the price ranges between $200 and $250 which was more than any of the configurations with the Jetson Nano.

Table 15: Data Processing Board Specifications

	Raspberry Pi 4	**NVIDIA Jetson Nano**
CPU	Quad-core ARM Cortex-A72 64-bit @ 1.5 Ghz	Quad-Core ARM Cortex-A57 64-bit @ 1.42 Ghz
GPU	Broadcom VideoCore VI (32-bit) (Integrated)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 Mhz
Memory	4 GB LPDDR4**	4 GB LPDDR4 (2 GB version also available) ***
Networking	Gigabit Ethernet / Wifi 802.11ac	Gigabit Ethernet / M.2 Key E (for Wi-Fi support) (2 GB version included USB Wireless adapter)
Display	2x micro-HDMI (up to 4Kp60)	HDMI 2.0 and eDP 1.4
USB	2x USB 3.0, 2x USB 2.0	4x USB 3.0, USB 2.0 Micro-B
Other	40-pin GPIO	40-pin GPIO
Video Encode	H264(1080p30)	H.264/H.265 (4Kp30)
Video Decode	H.265(4Kp60), H.264(1080p60)	H.264/H.265 (4Kp60, 2x 4Kp30)
Camera	MIPI CSI port	MIPI CSI port
Storage	Micro-SD	Micro-SD

Price	$55 USD	$99 USD

Table 16: Data Processing Board Benchmarks

Test	**Raspberry Pi 4 B 8GB**	**Jetson Nano 4GB**	**Notes**
Tinymembenc Standard Memcpy	2526.8 MB/s	3501 MB/s	MB/s > Higher Is Better
TTSIOD 3D Renderer Phong Rendering with Soft-Shadow Mapping	32.3075 FPS	41.00 FPS	FPS > Higher Is Better
7-Zip Compression Compress Speed Test	3701 MIPS	3501 MIPS	MIPS > Higher Is Better
C-Ray Total Time – 4K, 16 Rays Per Pixel	609.431 Seconds	932 Seconds	Seconds < Lower Is Better
Primesieve 1e12 Prime Number Generation	519.238 Seconds	468 Seconds	Seconds < Lower Is Better
AOBench Size: 2048 x 2048	125.643 Seconds	187 Seconds	Seconds < Lower Is Better
FLAC Audio Encoding WAV To FLAC	85.805 Seconds	104.01 Seconds	Seconds < Lower Is Better
LAME MP3 Encoding WAV To MP3	124.952 Seconds	144.21 Seconds	Seconds < Lower Is Better
Perl Pod2html	0.58082059 Seconds	0.7114 Seconds	Seconds < Lower Is Better

Redis GET	491168.39 Requests Per Second	568431 Requests Per Second	Requests Per Second > Higher Is Better
PyBench Total for Average Test Times	5673 Milliseconds	7080 Milliseconds	Milliseconds < Lower Is Better

3.2.6.3 Comparison

In terms of performance, the Raspberry Pi 4B and NVIDIA Jetson Nano were very close. As seen in **Table 16,** the slightly faster CPU of the RBP4 gives it the edge in serial tasks like compression, while the discrete GPU of the Nano makes it noticeably faster in parallel tasks like rendering. It was also important to consider AI image processing applications, as our drone was using computer vision to navigate. **Table 17** shows that the Jetson Nano was between three and ten times faster than the RBP4 when running Convolutional Neural Network algorithms.

However, this extra GPU performance of the Nano comes at the cost of nearly three times the power consumption and five times the weight compared to the RBP4. In embedded applications, having adequate compute power was important but having extra was not necessarily a good thing. It means that you sacrificed in some other aspect of the system such as cost, efficiency, size, or weight. Based on our research, we were confident that the Raspberry Pi 4B had the adequate performance to ran all of the flight control and data processing algorithms. The Jetson Nano would allow us to run our algorithms faster, but we had determined that the tradeoff on power consumption, weight, and complexity was not worth it.

Additionally, if the RBP4 turns out were underpowered for our application, we had a backup plan or adding a “Compute Stick” which was basically an external GPU. As seen in **Table 17**, the Intel Neural Stick 2 and the Google Coral USB both improved the RBP4’s performance to near the Jetson Nano’s, sometimes even beating it. This was an additional \$60 to \$80 on top of the already more expensive RBP4 + Navio2 combination, but it was important to note that this was only a backup plan. Based on our research, we would not need to use these. We had determined that the performance of the Raspberry Pi 4B combined with the Navio2 was sufficient for our purposes, and the extra cost was worth the increased simplicity and additional capabilities it would allow. This was still our design at the end of the project, and we would continue to use these but if we could do something, it would be increasing the GIPO pins.

Table 17: Data Processing Board CNN Benchmarks

Model	Framework	Raspberry TF-Lite	Raspberry Pi NCNN	Raspberry Pi Intel Neural Stick 2	Raspberry Pi Google Coral USB	Jetson Nano

Efficient Net-B0 224x224	TensorFlow	14.6 FPS (3) 25.8 FPS (4)	-	95 FPS (3) 180 FPS (4)	105 FPS (3) 200 FPS (4)	216 FPS
ResNet-50 (244x244)	TensorFlow	2.4 FPS (3) 4.3 FPS (4)	1.7 FPS (3) 3 FPS (4)	16 FPS (3) 60 FPS (4)	10 FPS (3) 18.8 FPS (4)	36 FPS
MobileNet-v2 (300x300)	TensorFlow	8.5 FPS (3) 15.3 FPS (4)	8 FPS (3) 8.9 FPS (4)	30 FPS (3)	46 FPS (3)	64 FPS
SSD Mobilenet-V2 (300-300)	TensorFlow	7.3 FPS (3) 13 FPS (4)	3.7 FPS (3) 5.8 FPS (4)	11 FPS (3) 41 FPS (4)	17 FPS (3) 55 FPS (4)	39 FPS
Binary model (300x300)	XNOR	6.8 FPS (3) 12.5 FPS (4)	-	-	-	-
Inception V4 (299x299)	PyTorch	-	-	-	3 FPS (3)	11 FPS
Tiny YOLO V3 (416x416)	Darknet	0.5 FPS (3) 1 FPS (4)	1.1 FPS (3) 1.9 FPS (4)	-	-	25 FPS
Open_Pose (256x256)	Caffe	4.3 FPS (3) 10.3 FPS (4)	-	5 FPS (3)	-	14 FPS
Super Resolution (481x321)	PyTorch	-	-	0.6 FPS (3)	-	15 FPS
VGG-19 (224x224)	MXNet	0.5 FPS (3) 1 FPS (4)	-	5 FPS	-	10 FPS
Unet (1x512x512)	Caffe	-	-	5 FPS	-	18 FPS
Unet (3x257x257)	TensorFlow	2.0 FPS (3) 3.6 FPS (4)	-	-	-	-

3.2.7 Mapping Algorithms

The mapping algorithm was the piece of software that makes our drone fully autonomous. It allows the drone to understood where it was within its environment and understood the obstacles it had to overcome, rather than simply telling it the distance to the closest obstacle. The rangefinders included in our drone was a key part of the collision avoidance system which stopped the drone from bumping into things. However, our goal was for the drone were able to navigate between obstacles and reach a destination. In order for the drone to plan a path for itself, it needed to know where it could go and where it cannot.

From there we could ran a simple pathfinding algorithm such as A* and the drone would navigate autonomously.

When selecting a mapping algorithm, we analyzed several characteristics. One of the key requirements was the computational expense. Being an embedded system, we only had limited resources to worked with. Keeping in mind that there were other programs being ran in addition to the mapping algorithm, such as the flight control firmware, the operating system, and sensor reading, it was important for our mapping algorithm were efficient with its computational resource usage. Additionally, it was important for our algorithm were fast. Being a drone, every second of flight time takes energy. Flying vehicles do not had the luxury that a ground rover had in terms of waiting idly while computations were performed. This also means that flying the drone slower to account for the slower mapping algorithm would reduce range. It was important for our mapping algorithm to kept up with our drones flying at a reasonably high speed. Another consideration was that our mapping algorithm should be based on monocular computer vision. One of the goals of our drone was where inexpensive so that it could be deployed on a mass scale for shipping companies. For this reason, we chose camera-based computer vision over LIDAR mapping since LIDAR sensors were orders of magnitude more expensive than cameras in the current market. Similarly, we chose a single camera over stereo cameras because two cameras would increase cost and complexity. Monocular computer vision algorithms rely on motion parallax to calculate depth information rather than comparing the information between images of different perspectives as with stereopsis.

A key consideration for our project was the level of robustness of our algorithm. For this, it was important to understand the difference between Visual Odometry and SLAM (Simultaneous Localization and Mapping). Visual Odometry aims to create a locally consistent record of camera motion, which was effectively the same as drone motion for our used case. SLAM, on the other hand, aims to create a globally consistent map of the environment. Both algorithms would give us useable 3D maps of our environment, which we could then use to calculate a path for the drone to took. The difference was how they do it, which could be seen by comparing **Figure 24** and **Figure 25.** SLAM uses much more sophisticated calculations to made it more accurate over long distances and long timeframes, as well as making it robust over periods of bad visual information. Of course, the extra calculations also slow down the SLAM algorithm significantly. A comparison of the different algorithms we looked at, and their different approaches to solving the problem could be seen in **Table 17.** Since Visual Odometry suffers from inaccuracies in regions far in space and long ago in time, but remains accurate in local time and space, it should be sufficient for obstacle avoidance. The goal for us was not to create an accurate 3D map of the world, but just get our drone past an obstacle. While the robustness features of SLAM were nice to have, they were not strictly necessary and not worth the additional computation time. With these considerations in mind, we selected four potential mapping algorithms to evaluate.

Figure 24: Mapping Algorithm Flowchart

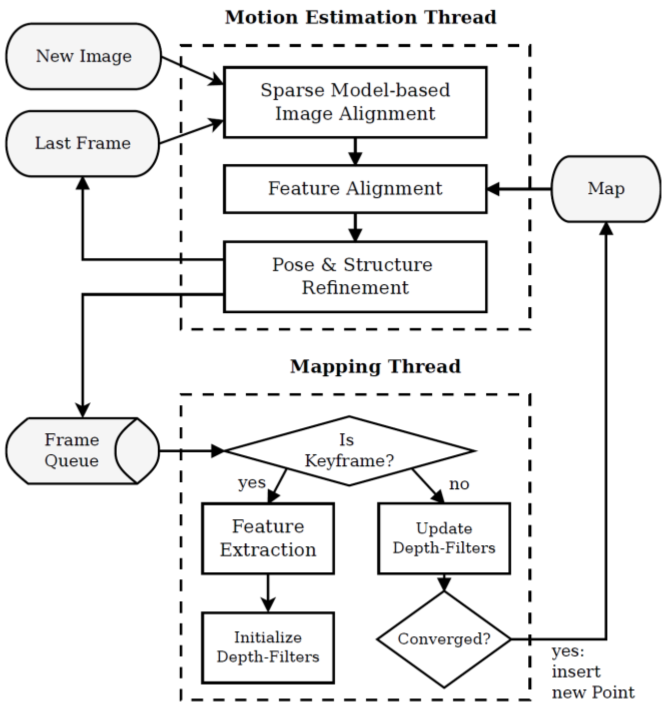
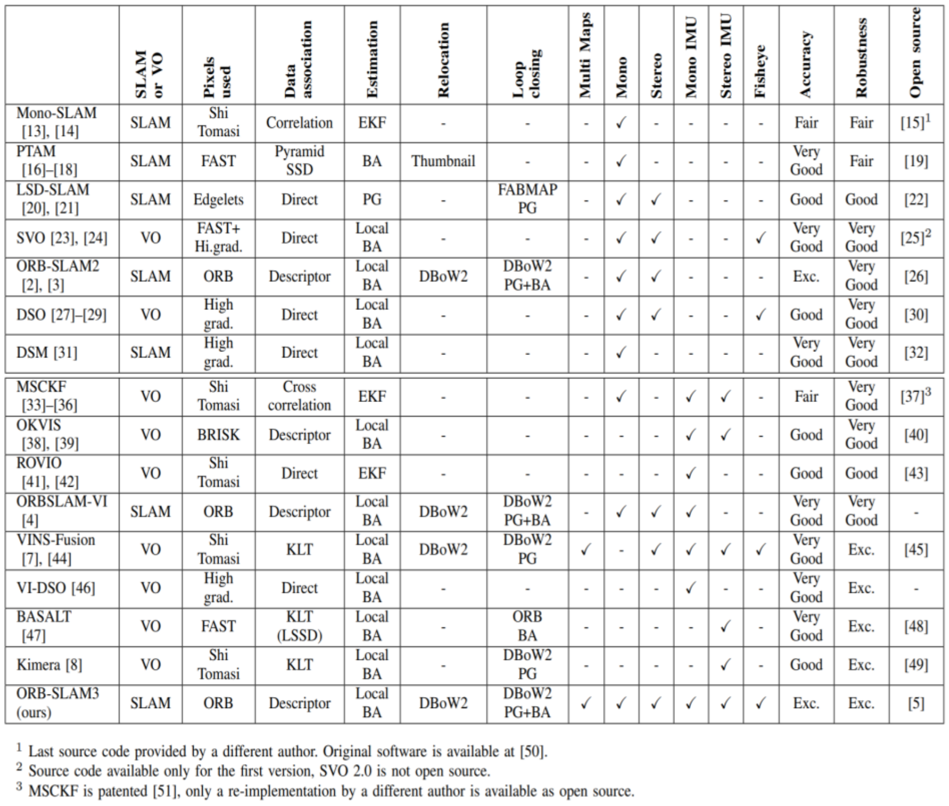


Figure 25: Mapping Algorithm Table



	SLAM or VO	Pixels used	Data association	Estimation	Relocation	Loop closing	Multi Maps	Mono	Stereo	Mono IMU	Stereo IMU	Fisheye	Accuracy	Robustness	Open source
Mono-SLAM [13], [14]	SLAM	Shi Tomasi	Correlation	EKF	-	-	-	✓	-	-	-	-	Fair	Fair	[15][1]
PTAM [16]–[18]	SLAM	FAST	Pyramid SSD	BA	Thumbnail	-	-	✓	-	-	-	-	Very Good	Fair	[19]
LSD-SLAM [20], [21]	SLAM	Edgelets	Direct	PG	-	FABMAP PG	-	✓	✓	-	-	-	Good	Good	[22]
SVO [23], [24]	VO	FAST+ Hi.grad.	Direct	Local BA	-	-	-	✓	✓	-	-	✓	Very Good	Very Good	[25][2]
ORB-SLAM2 [2], [3]	SLAM	ORB	Descriptor	Local BA	DBoW2	DBoW2 PG+BA	-	✓	✓	-	-	-	Exc.	Very Good	[26]
DSO [27]–[29]	VO	High grad.	Direct	Local BA	-	-	-	✓	✓	-	-	✓	Good	Very Good	[30]
DSM [31]	SLAM	High grad.	Direct	Local BA	-	-	-	✓	-	-	-	-	Very Good	Very Good	[32]
MSCKF [33]–[36]	VO	Shi Tomasi	Cross correlation	EKF	-	-	-	✓	-	✓	✓	-	Fair	Very Good	[37][3]
OKVIS [38], [39]	VO	BRISK	Descriptor	Local BA	-	-	-	-	-	✓	✓	-	Good	Very Good	[40]
ROVIO [41], [42]	VO	Shi Tomasi	Direct	EKF	-	-	-	-	-	✓	-	-	Good	Good	[43]
ORBSLAM-VI [4]	SLAM	ORB	Descriptor	Local BA	DBoW2	DBoW2 PG+BA	-	✓	✓	✓	-	-	Very Good	Very Good	-
VINS-Fusion [7], [44]	VO	Shi Tomasi	KLT	Local BA	DBoW2	DBoW2 PG	✓	-	✓	✓	✓	✓	Very Good	Exc.	[45]
VI-DSO [46]	VO	High grad.	Direct	Local BA	-	-	-	-	-	✓	-	-	Very Good	Exc.	-
BASALT [47]	VO	FAST	KLT (LSSD)	Local BA	-	ORB BA	-	-	-	-	✓	-	Very Good	Exc.	[48]
Kimera [8]	VO	Shi Tomasi	KLT	Local BA	-	DBoW2 PG	-	-	-	-	✓	-	Good	Exc.	[49]
ORB-SLAM3 (ours)	SLAM	ORB	Descriptor	Local BA	DBoW2	DBoW2 PG+BA	✓	✓	✓	✓	✓	✓	Exc.	Exc.	[5]

[1] Last source code provided by a different author. Original software is available at [50].
[2] Source code available only for the first version, SVO 2.0 is not open source.
[3] MSCKF is patented [51], only a re-implementation by a different author is available as open source.

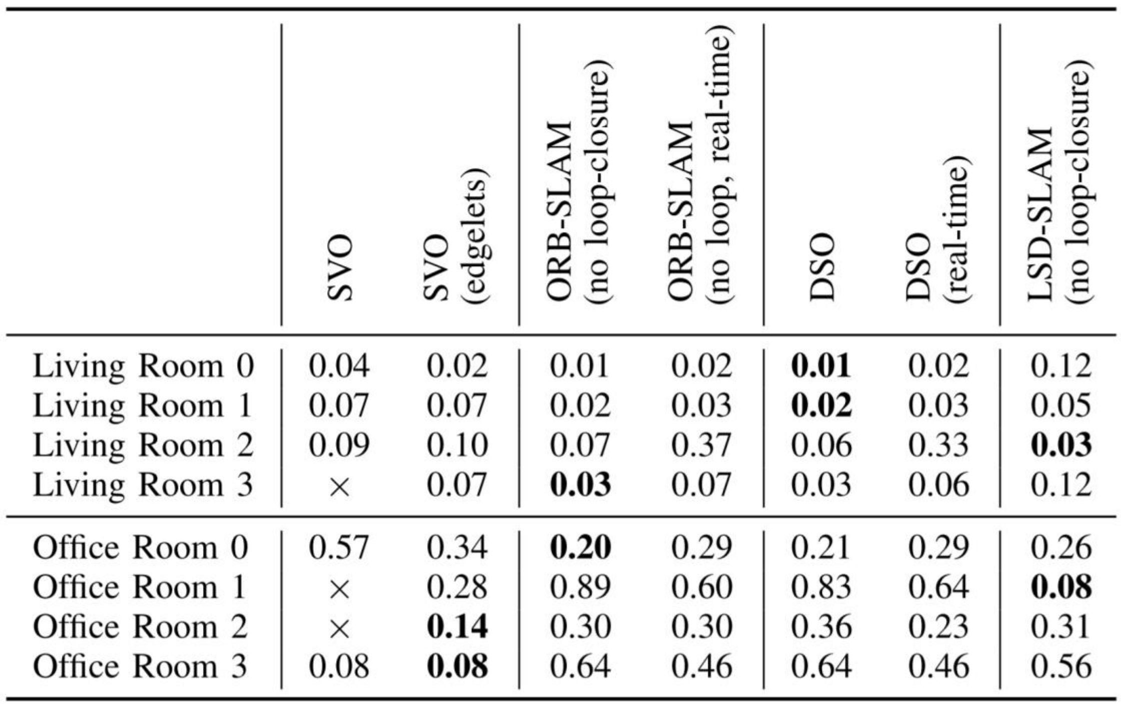
3.2.7.1 Semi-Direct Visual Odometry 2.0

Semi-Direct Visual Odometry was first implemented in 2014 and was later extended in 2017, making it one of the oldest algorithms being compared. This was not necessarily a bad thing, though, as it was still very effective. As the name suggests, SVO implements Visual Odometry and lacks the features of SLAM such as loop closing and key point reuse. However, it makes up for it by being around ten times faster than the SLAM algorithms, as seen in **Table 18.** Additionally, its CPU utilization was between three and five times lower than that of the SLAM algorithms. This would free up resources for other processes such as path planning.

Table 18: Mapping Algorithm Frame Time Comparison

	Frame Times (ms)		
Mapping Algorithm	Mean	St.D.	CPU@20fps
SVO Mono	2.53	0.42	55
ORB Mono SLAM	29.81	5.67	187
LSD Mono SLAM	23.23	5.87	236
DSO Mono	9	-	-

Table 19: Mapping Algorithm Accuracy

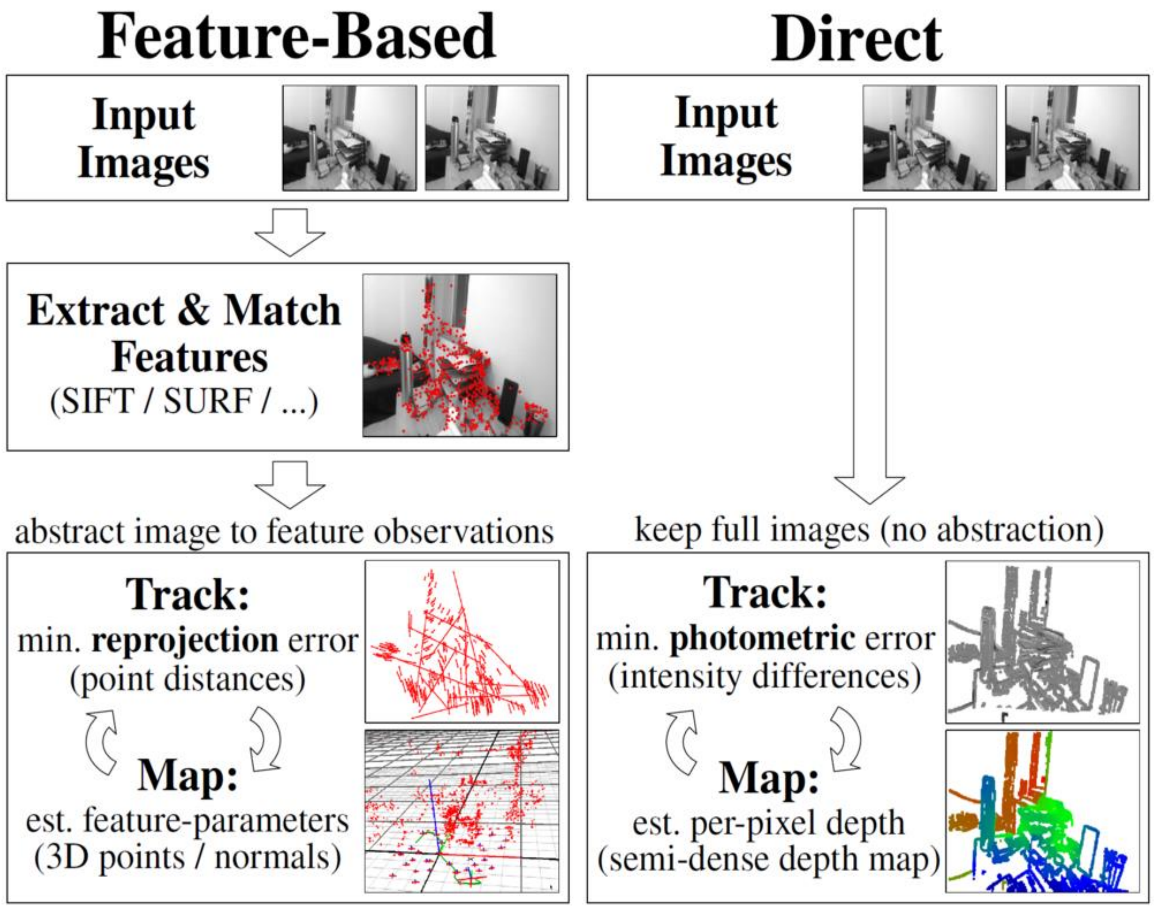


	SVO	SVO (edgelets)	ORB-SLAM (no loop-closure)	ORB-SLAM (no loop, real-time)	DSO	DSO (real-time)	LSD-SLAM (no loop-closure)
Living Room 0	0.04	0.02	0.01	0.02	**0.01**	0.02	0.12
Living Room 1	0.07	0.07	0.02	0.03	**0.02**	0.03	0.05
Living Room 2	0.09	0.10	0.07	0.37	0.06	0.33	**0.03**
Living Room 3	×	0.07	**0.03**	0.07	0.03	0.06	0.12
Office Room 0	0.57	0.34	**0.20**	0.29	0.21	0.29	0.26
Office Room 1	×	0.28	0.89	0.60	0.83	0.64	**0.08**
Office Room 2	×	**0.14**	0.30	0.30	0.36	0.23	0.31
Office Room 3	0.08	**0.08**	0.64	0.46	0.64	0.46	0.56

Another requirement we had was for our mapping algorithm to worked well with a monocular camera setup, which SVO 2.0 does very well. Additionally, it was seen that SVO does very well compared to the other algorithms we looked at, in terms of accuracy, as seen in **Table 19.** Each algorithm had a dataset on which it performs best, but the fact that SVO was keeping up with the others while being so much faster was impressive. It was important to note that in the comparison **Table 19,** ORB-SLAM and LSD-SLAM both had loop closing turned off, which was one of the features that makes it more accurate but also more computationally expensive. Overall, SVO was a very effective mapping algorithm and seems like a good fit for our application. It was nice to had some of the extra features of SLAM, but they were not strictly necessary. The extra computational power freed by using SVO 2.0 versus some of the other algorithms could be worth the slight decrease in inaccuracy.

3.2.7.2 LSD-SLAM

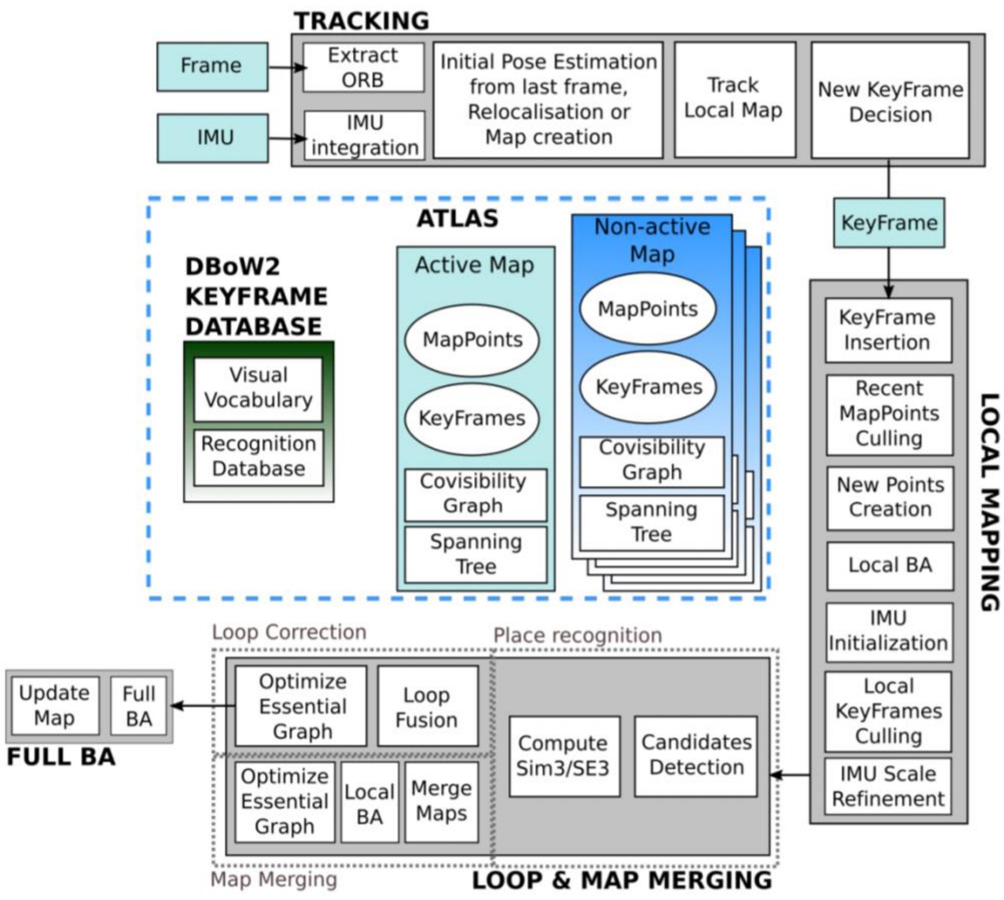
Figure 26: Feature-Based vs Direct Visual Information Mapping



LSD-SLAM or Large-Scale Direct SLAM was released in 2014 with no updates since then, making it the oldest algorithm being considered. It uses the direct method, shown in **Figure 26,** which increases accuracy and robustness in sparsely textured environments, as well as making the 3D map denser. Non-direct methods took the input images and extract features. This decreases the bandwidth of data being processed, but abstraction necessarily decreases information that could be processed. By contrast, the direct method skips feature extraction and uses the full images for tracking and mapping. This does not necessarily speed up the algorithm as processing happens on larger objects of data and could increase memory usage as these images needed were stored for longer. In terms of speed, LSD-SLAM was relatively slow. In **Table 18,** it was shown were slightly faster than ORB-SLAM, but it takes significantly more computational power. Additionally, ORB-SLAM had since been updated to increase both accuracy and speed, while LSD-SLAM had not. It does work well with a monocular camera setup, however. Being a SLAM algorithm, it was extremely accurate and robust. LSD-SLAM was a very effective method for mapping, but it was simply too slow and too expensive to run on an embedded system such as our drone. For that reason, this was not a very good option for us during implementation.

ORB-SLAM3

Figure 27: ORB-SLAM3 Mapping Algorithm



The second SLAM algorithm we evaluated was ORB-SLAM3 which uses Oriented FAST and Rotated BRIEF (ORB) techniques for feature extraction. As this implies, ORB-SLAM was a feature-extraction-based method as seen in **Figure 27.** The newest version, ORB-SLAM3 was released in mid-2020, making it a very new algorithm. As expected, it was one of the best we had looked at. It was relatively fast for what it does, definitely faster than LSD-SLAM. It was also one of the most accurate options. It supports monocular setups like the other options we had discussed. It was also extremely robust against periods of poor visual information. An example of poor information was pure rotation with no lateral motion. This was bad because it was possible for previously mapped regions to leave the visual frame even though the region entering the frame was unable were mapped. This was because, without lateral motion, the system was not receiving depth information. Effectively, pure rotation forces the system to restart its map. ORB-SLAM3 was robust against this, but even if it does happen, the algorithm supports re-localization, combining the new map with the old map, as seen in **Figure 25.** It also supports map reuse over multiple runs, which could decrease computation around high traffic areas like the warehouse. This algorithm supports all of the other features expected of SLAM such as loop closing too. Overall, ORB-SLAM3 was the most fully featured, robust, and accurate of all of our options. Our only concern was that it could be too slow or too computationally expensive.

3.2.7.3 Direct Sparse Odometry

Direct Sparse Odometry was the Successor to LSD-SLAM, made by the same research team. It was released in 2016 which leaves it average in terms of age. It was an implementation of Visual Odometry similar to SVO. However, it did have multiple extensions available, such as Visual-Inertial DSO which adds in measurements from accelerometers and gyroscopes to obtain more accurate ego-motion. It also had an extension with loop-closure and Sim (3) pose graph optimization. This gets the algorithm closer to full implementation of SLAM. As seen in **Table 18,** Direct Sparse Odometry was only slightly slower than SVO, and between two and three times faster than both SLAM implementations. Being Visual Odometry rather than SLAM, it was also less computationally expensive. It was designed to worked with a monocular camera setup, so that should not be a problem. It was relatively accurate as seen in **Table 19.** Furthermore, it was robust to short periods of poor visual information. The robustness could be increased further with the previously mentioned extensions. Overall, Direct Sparse Odometry was a very good option for our system. It was fast enough while also not utilizing too much of the computation power. It also gives us the flexibility to pick the features we really needed without including the ones we don’t which helped us optimize our computation usage.

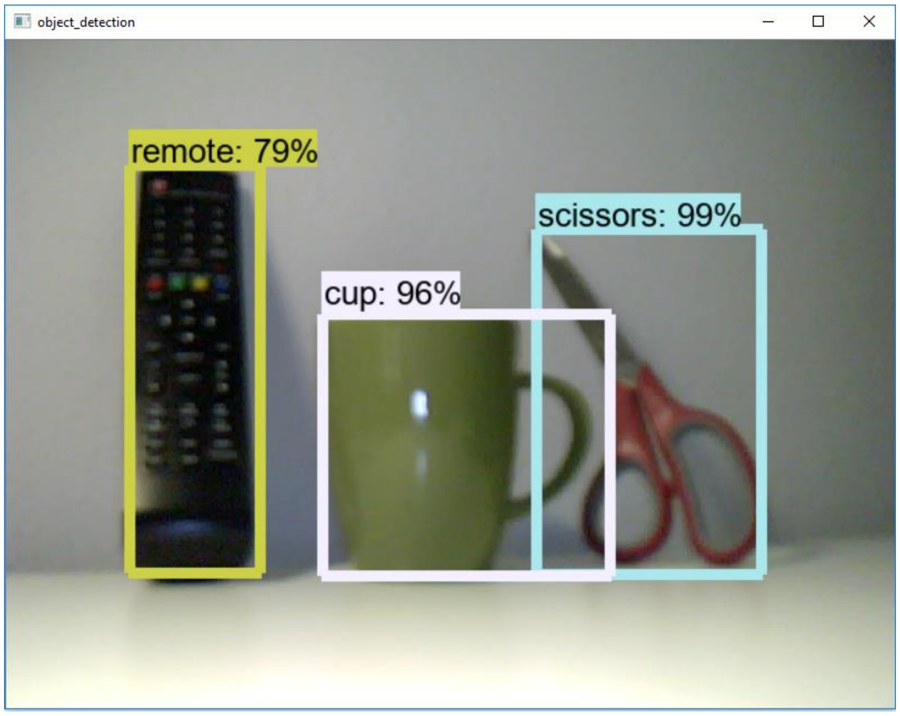
3.2.7.4 Comparison

It was impossible to determine the mapping algorithm for our application without performing our own testing. However, from our research, ORB-SLAM3 appears were the best option for us. Being a full SLAM implementation, it was the most fully featured, the most accurate, and the most robust. For this reason, ORB-SLAM3 was our first choice. It was possible that it does not ran fast enough on our processing board, or that it utilizes too much of the available compute resources. In that case, we tried to use Direct Sparse Odometry. This was our second choice because it was still very accurate and robust, while still being much faster than ORB-SLAM3. It also gave us the option to add in features such as loop closing if we had leftover compute power. Our final backup option if the previous two do not worked was Semi-Direct Sparse Odometry. It was the fastest and least computationally expensive algorithm we looked at. It was slightly less accurate and robust than the others, but it was almost guaranteed were able to run on even the lowest powered compute board. We were not further considering LSD-SLAM as an option. It was not a bad algorithm, but there were better options for what we were using it for. ORB-SLAM3 was just as full-featured while being faster and less computationally expensive. If ORB-SLAM3 does not worked for us after testing, it could be assumed that LSD-SLAM would fail for the same reasons. For that reason, we would attempt to used ORB-SLAM3, with DSO as a backup and VSO as a worst-case scenario.

Unfortunately, during implementation we found multiple problems with our decision to use a 3D mapping algorithm for collision avoidance. The first problem is that it is very complicated to set up. Initially, we had enough time to work through problems and troubleshoot. Unfortunately, the initial setup of the flight platform took much longer than expected. This did not give us enough time to correctly implement the mapping software. The next problem was with our development environment. As a team, we had access to Windows and Mac for development environments, but we intended to deploy the software on a headless version of Linux. This made it hard to transfer the software over to be actually utilized on the drone. We were unable to get any of the mapping software working on the drone; not just ORBSLAM, but all of the backup algorithms as well. Another problem for us was the mapping algorithms required much more custom development than we previously thought. From all the documentation and our research, it seemed that these algorithms would be able to be installed and run. However, we discovered that they were much less reliable than this. They required custom code to be written for us to get them running on each platform. For all of these reasons, we realized that it was infeasible for us to use 3D mapping for our project. Given a longer development timeframe and more resources, we stand by our decision to use 3D mapping for obstacle avoidance in our drone. However, we needed to stay within our budget and complete the project before the deadline, so we decided to pivot. We realized that using object detection for our drone would be a good alternative without any of the problems we had when implementing 3D mapping. For this reason, we decided to use OpenCV for the object detection.

3.2.8 Object Detection

The responsibility of the computer vision software is to identify obstacles in the drone’s environment. It does this by using shape, size, and color of objects to estimate the object type, as seen in Figure X. The position of these objects is then passed to the flight control software to navigate around any obstacles and avoid collisions. Additionally, the computer vision software will be responsible for identifying the package drop zone, marked by a red rectangle, so that the drone knows where to deliver the package.



Our drone used OpenCV as our computer vision software. This is a widely used and supported computer vision library which has many different models such as object detection, depth mapping, and visual odometry. All of these features mean that our drone will not get lost or confused during the two most critical times of the flight: takeoff and landing. However, we did consider other options besides OpenCV. The most realistic competitor was TensorFlow.

TensorFlow was a very capable platform for object detection. In fact, it is used very frequently for applications like this. There were a couple of reasons why we steered away from this though. For one, we found out that TensorFlow requires much more tuning and tweaking than OpenCV. This can be beneficial for custom use cases, as it provides more freedom. However, our use case is not super out of the ordinary. This meant that the ease of use of OpenCV was a huge benefit. For this reason, we chose to use OpenCV for our object detection library. We used YOLOv2 with python to program. This algorithm was used during flight for object detection to avoid collisions, but it was also used in the delivery phase of flight to identify the drop zone. For this use case, we used a red square on the ground to denote the location that the package should be dropped. We could not rely on GPS because it is only accurate to a couple meters, and it drifts from day to day. For a more precise delivery, we denoted the spot and had the computer vision locate it as the drop zone. From there it was much simpler to ensure that our package was delivered to the right place.

Overall, the object detection algorithm was very critical in every aspect of flight, as it is used to avoid collisions as well as find the perfect location to deliver the package to. For this reason, we deliberated very carefully about which algorithm would be best for our specific application. After doing this, we finally arrived at using OpenCV to control our drone.

3.2.9 Computer Vision Camera

The camera is the heart of our obstacle avoidance system. The industry standard for collision avoidance in drones is to use expensive sensors such as LIDAR or depth-sensing cameras. By contrast, our system uses a single-point camera and calculates the 3D point cloud by performing calculations with motion data from the drone. Using the cheaper camera reduces the overall unit cost by around 40%. The tradeoff is that our system uses extra processing power to perform the 3D point cloud calculations, reducing the system’s power efficiency and flight time. However, this is not nearly as impactful as it sounds, since the computer vision only needs to run during takeoff and landing. As there are no obstacles at the flying altitude, it is safe to turn off the collision avoidance system to save power and rely on GPS for coarse navigation.

The camera we chose for our project has a 5 Megapixels (2592 × 1944 pixels) OV5647 sensor. It is able to capture still images at full resolution, 30fps video at 1080p, and 60fps video at 720p. This particular sensor provides a good balance between cost, resolution, low light sensitivity, and framerate for fast-moving images. This is what allows us to decrease the cost of our drone by so much compared to current offerings.

3.2.10 Drop Mechanism

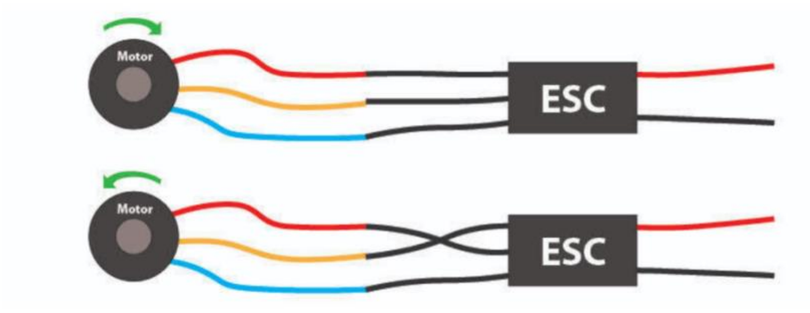
A key feature of our aircraft is being able to drop our package at its destination. Having the drone carry the package once it is attached is not an issue. However, it was a challenge to ensure that our drone is able to be equipped with a variety of packages sizes and drop them with minimal modifications to the packaging. Rather than a claw mechanism or a hook, we felt the best option for this was a sliding pin release mechanism, as seen in Figure 5. Any shape of packaging can be secured to the drone using a harness that attaches to the drone and fits through the sliding pin. Additionally, in the event of a power failure or other error, the mechanism stays closed rather than releasing the package, which is another safety feature. When the package is ready to be released, the pin slides out, releasing one side of the harness, while the other side stays attached to the drone. The means that the harness can be reused and is not wasted, which would be a problem for a system with extremely high usage volume.

Our particular drone used an MD752 electric servo, which can provide 6.3 kilograms of torque at the arm radius. This was more than enough torque to even have the servo hold the half pound package. It had an even easier time pulling the pin out to open the mechanism and drop the package. We could actuate it by manually pressing a button on the transmitter for testing, or have the drone automatically move the servo either at a certain waypoint or when it identified the drop zone with computer vision. This mechanism was a critical piece of our project as it provides the main feature: delivery. It needed to be reliable enough to be safe and not drop the package during flight, which could be dangerous. It also needed to be reliable enough to drop the package at the correct place, when necessary, not getting stuck or becoming unresponsive.

3.2.11 Electronic Speed Controllers

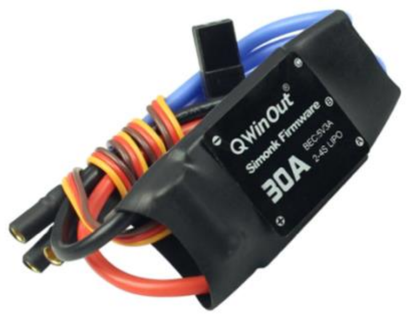
ESC stands for Electronic Speed Controller and from the name comes the goal of the device. The device's goal was to control the speed of the drone, and this was being given by telling the motor to rotate and essentially acts as a translator from the commands being given through the pilot and what voltage/current needed to go through the motor to accomplish it. The ESC adds a lot of control to the motor starting with how the wires were connected. Depending on the user's choice for wire connection would determine which direction the motor rotates. As could be seen in **Figure 28** below, shows that connecting the wires very specifically turned the motor in one direction and the opposite direction, which was important to set up a 4-propeller system with 2 of them going in the same direction and 2 going in the opposite [3].

Figure 28: ESC Wiring Options



From here there were a few factors that went into choosing an ECS. The factors included weight, size, voltage, and current output, BECs, and 1 ESC vs 4in1. For all of these factors, ultimately affect the performance of the ESC on the drone. Weight and size were a consideration for all components on a drone because the heavier and larger the size the more power was needed to fly and control the drone. With ESCs they had small and light sizes, but there was a balance to found based on the motor because if too small then the ECS would overheat and not perform as well. This was part of the reason why it was recommended to make an ESC decision after making the motor decision. Motors had a max output of power and some ESCs could handle more power than the motor was giving but at a higher cost. To save costs was it better to stick to the max power of the motor and not much reason to pick an ESC that handles more power than it was able to give off [3].

Figure 29: ESC 4 IN 1



Add-on components were optional when choosing an ESC. This included selecting ESC with BEC and a 4 in 1 option versus an individual. A BEC was a battery elimination circuit that provides more control to the power given to other components. For drone projects, this was not a necessary option as the Power Distribution Board (PDB) takes care of this operation. Another optional component was purchasing individual ESC vs 4 in 1. The 4 in 1 was a board that had 4 ESCs already put together and ready to use, whereas individual ones needed were put together. With drone enthusiasts, it might be more cost-effective to purchase the 4 in 1 option, but for beginners, if one ESC was broken then the whole 4 in 1 board needed were replaced. Hence, why for this project, we took the individual route, but through researching our desired frame and motor design, we found a kit that included the ESCs needed to operate. A picture of the ESC was above in **Figure 29** and had a working current of 30A for the motor. While purchasing this separately was more expensive, finding it with a kit helped with those costs [4].

3.2.11 Coding Language:

For this project, we incorporated the basis machine learning code directly on the drone to the target and then building upon that with computer vision to accurately saw objects and found the best path around. Deciding on the machine learning language and library was relatively easy. The biggest debate was choosing Python vs Java as they both could perform the same end goal. With Java, some advantages were the security of the compiler and runtime environment, and because of how the language was structured it helps the performance of machine learning projects perform faster. Some disadvantages were that it requires a more complex project set up and could be harder to understood when working in a group and could took up more memory over time [5]. For Python, the language was known for its user learnability and easy-to-read syntax, which having two members not extremely familiar with coding, this was easy for them to followed along and learned. Python's ML libraries were also very well-known and had a large audience of users online and practiced a lot in the industry, so if we came across bumps with the understanding, we had resources to looked at. The main library we were looking to utilize was Tensorflow, which had an abundance of documentation and learning material for us to deep dive into learning how to adapt it to our project. Some disadvantages to Python that might arise was the ran time errors because it was a dynamically typed language and known for being slightly slower than other high-level languages. **Table 20** below goes into outlining some of the factors that were looked at when comparing Java vs Python. Noting the easier to code once for Java comment, it had been known that it was easier to maintain and set up a project that could be updated, but with Python, it might came with more hoops to jump through [6] [7].

The next software choice for our project included the firmware being used for the flight controller and so on. The two main options after research for this were Ardupilot and PX4. They both were based in C++ so when deciding which tool went with, we tried looking to the one that worked best with the flight controller. After further researching the flight controllers, it became apparent that the technology was very popular and the ones we narrowed down could handle both. This opened the doors back opened for either one. As students not knowing an extreme amount about the subject, we looked into which software had the more prevalent communities and helped forums to guide us when familiar. With Ardupilot, some of the cons when researching was that a lot of the information was dated back to 2017 and not very kept up and up to date. PX4 had a very nice user interface that made it easy to navigate and found the information easily. There was clear documentation and clearly stated or divided into subject matters for drones. Because of the clear directions and strong forum community we decided to go with PX4 for the time being. The PX4 software was based mainly on C++ when changing things to interact with the firmware and hardware, and then Python for the software and guidance systems.

Table 20: Python vs. Java

Python	Java
Easier to learned	Easier to code once
Dynamically Typed	Statically Typed
Slightly more popular	Faster than Python
TensorFlow	Weka, Mallet, Deeplearning4j, MOA

Platform dependent	Platform independent

3.2.12 Ground Control Station Software

Ground control software was the program needed for a user to interface with the drone during flight planning and during the flight. There were many different competitive options including QGroundControl and MissionPlanner. We selected QGroundControl for multiple reasons. Firstly, it was supported by the Dronecode Foundation, the same foundation in charge of PX4 autopilot. and MAVLink. Given that, it was very well integrated with the flight control firmware we were using, as well as the messaging protocol. It seems best to stay within the ecosystem rather than trying to integrate a separate ground control software with the rest of the system. It was also highly rated and known to have a clean intuitive user interface, which was another reason for us to choose this option. It was also the only officially supported option for PX4 firmware, so the decision was made for us.

QGroundControl supports all necessary options for setting up and configuring the drone. It allows for flashing the PX4 firmware and establishing the configuration parameters. After the drone was set up, the ground control software could be used to plan the mission, setting waypoints and home coordinates. The software was also used to interface with the drone during flight. It supports the MAVLink communication protocol, which was a way to send commands to the drone. On top of controlling, it with MAVLink commands, QGroundControl supports virtual joysticks to fly the drone. This means that we did not need a dedicated transmitter. This is what we thought for a while. However, after testing we came to realize this was false and had to buy one on short notice. During the flight, the software could display telemetry data from the drone such as altitude, flight speed, angle, battery level, and other information. It also supports video streaming which was interesting because we were able to see exactly what the drone sees and how it navigates. The ground control software displays the drone on a real-time flight map, including satellite data, topographic data, restricted flight zones, and other geographic data. Additionally, the software supports data logging and flight playback. This allowed us to saw all of the data, including the camera input, and analyze it if something goes wrong. QGroundControl was available on every major operating system, including Windows, Linux, OS X, iOS, and Android. This would give us the flexibility to set up and transfer our ground station to any environment.

3.2.13 Drone Control Communication Protocol

3.2.13.1 MAVLink

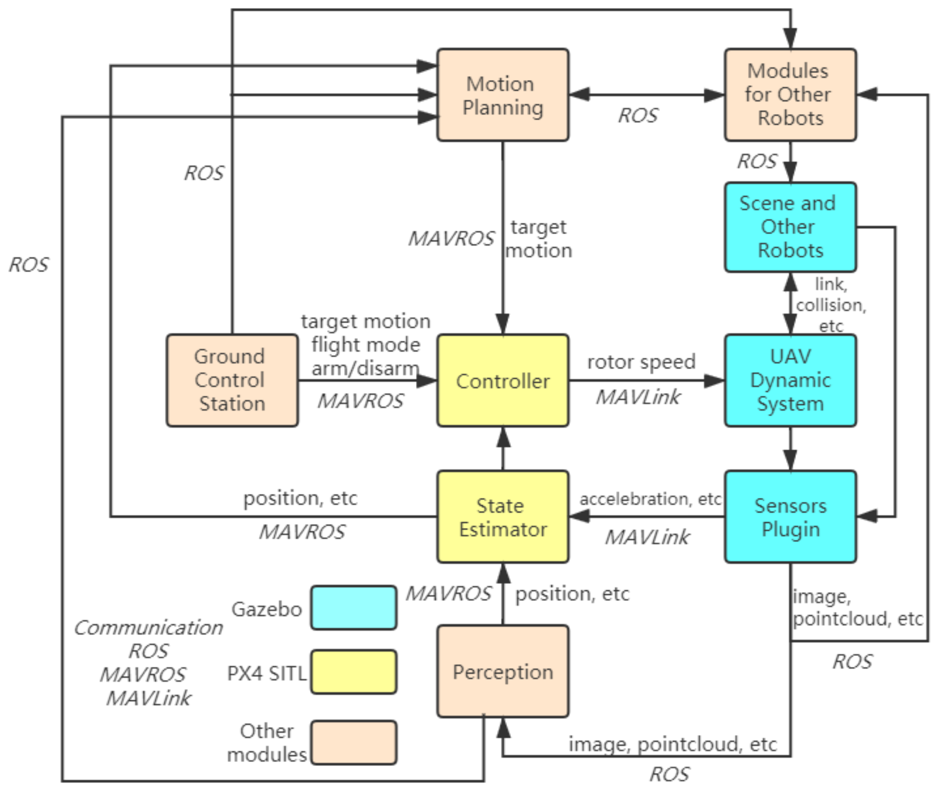
MAVLink was a lightweight messaging protocol designed for drone communication. It was useful for both drone-ground station communication and internal processes communication on the drone itself. It utilizes a hybrid architecture for efficient messaging. Data streams were sent using the publish-subscribe design pattern. On the other hand, Configuration sub-protocols were sent using a point-to-point architecture with retransmission. This combined with the efficient error detection and lack of framing makes the protocol very efficient with low bandwidth overhead. Additionally, the protocol was very reliable. MAVLink supports packet drop detection, corruption detection, and packet authentication. Another key feature was that the MAVLink command set was extensible. The commands were defined as XML files and were easily able were changed and added to. This protocol also allows us to used basic and programming language and operating systems. It supports C, C++, Java, and Python as well as Windows, Linux, OS X, iOS, and Android. Since this protocol was supported and maintained by the Dronecode Foundation, the same foundation that makes QGroundControl and PX4, it was already integrated with this software. This makes it an easy choice for our communication protocol. It also had many extensions and developer APIs like DroneKit, which would made development even easier.

3.2.13.2 DroneKit

DroneKit was a developer API built on top of MAVLink. It was designed to make the development of drone software using MAVLink even easier through high level abstraction. DroneKit enables the creation of Python applications that communicate with vehicles using MAVLink. It had functions that made it easy to connect and access the vehicles information. Some of the accessible information was vehicle telemetry, vehicle state, and parameter information. In addition to this, it enables easy mission management such as changing waypoints and direct drone control. It’s intended use were ran on a companion computer separate from the flight controller. However, it should not be too hard to set up a ROS node on the same board as the flight control software and had them both running concurrently. We expect DroneKit were a very valuable asset because it had a very active forum of developers using it for similar projects to ours. It was mainly used for high level cognitive functions such as our SLAM algorithm. These forums should be helpful in our learning how to integrate the different processes together and got everything working. Additionally, it was very well documented with examples for each of the features like the visual odometry package.

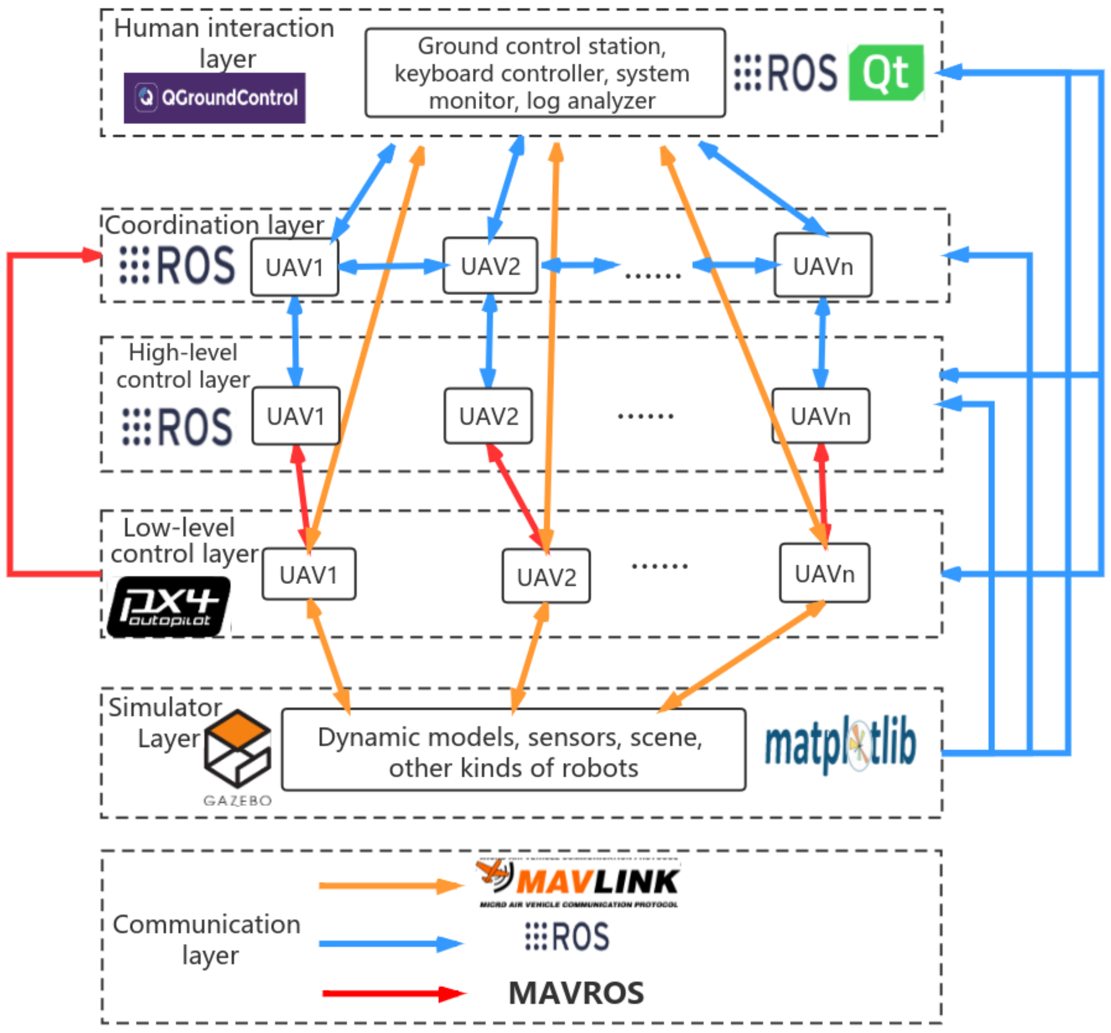
3.2.14 System Stack

Figure 30: System Architecture



Our overall platform was implemented in two separate systems. The first was a simulator and the second was the physical flight control board. The two systems would by functionally the same but used for different purposes. The simulator was used for testing our high-level cognitive functions such as the SLAM algorithm, the collision avoidance software, and interface between them. Once we successfully implement the system in software, we were safe to moved it to hardware. From there we would install and integrate all of the firmware and software on the Raspberry Pi and Navio2. Whether it was the simulation program or the physical flight control board, the software stack was the same. The lowest level software was operating system and the flight control firmware. The Navio2, our flight control daughter board, comes with a preconfigured operating system image. It was a version of Raspbian with a preemptive Linux kernel. This means that it would do a better job at meeting the real-time scheduling constraints of a safety critical system. It also had all of the Raspberry Pi GPIO pins preconfigured to interface with the Navio2. This overall system stack could be seen in the flowchart in **Figure 30.**

Figure 31: System Stack



Furthermore, it had ArduPilot, ROS, DroneKit, and GStreamer preinstalled. Since we were using PX4 instead of ArduPilot, we just had to swap the two software's out. ROS was a collection of tools and libraries that helped to simplify the process of programming robotic systems. This was helpful with in creating a unified system between the simulation, the drone, and the ground control system. It also implements useful libraries like OpenCV (computer vision) and MoveIt (path planning). DroneKit, as discussed in a previous section, was useful in integrating our SLAM algorithm with the flight control software. GStreamer was video streaming software that allowed us to stream the drone's camera view to our ground control station in real time, allowing us to saw what it sees. The software program flow as well as inter-process communication was shown in **Figure 31.** It illustrates the way that each layer of software builds on the layers before it, and how each of them communicate and send data between the layers.

3.2.15 Software Simulator

When creating a complicated system, it was important to test each component individually before integrating it into the whole system. The high-level functions in particular, such as the collision avoidance decision making and the path planning algorithms, needed were tested thoroughly as there were many ways that it could gave incorrect results. Before testing the cognitive functions on the physical drone, which could easily be damaged in a crash, it was helpful to test using a simulator. The program needed to run the flight control firmware as well as the 3D mapping algorithm exactly the same as it would on the physical drone. However, instead of sending the MAVLink commands to physical hardware like the ESCs, the software sends the commands to the simulation software which then interoperates them and simulates what a real drone was doing. For us it was important that our simulator was relatively accurate so that we could be sure it was transferable to our physical drone. Another requirement we had for the simulation software was that it works well for testing computer vision. The flight control software had already been vetted and knew were stable. Also, the communication protocol could be tested without flying the drone or putting it in danger. The only part of our system that really needed were tested before applying it to the drone was the collision avoidance and decision making. For that reason, it was important that the simulation supports computer vision testing and also provides an accurate output for it. The last consideration we had for the simulation software was that it works we our existing stack. We did not have to change a particular aspect of our system to made it compatible with the simulation software.

3.2.15.1 Gazebo

Gazebo was one of the most highly regarded simulation software’s for drones and autonomous robots. It was designed were particularly suitable for testing object-avoidance and computer vision programs. It supports both software in the loop and hardware in the loop simulation. Software in the loop, or SITL, means that the flight control software runs on the same computer as the simulation, which was useful because it does not require us to actually had a working drone to test on. On the other hand, Hardware in the loop, or HITL, means that the flight control software and computer vision program ran on the flight control board, exactly how they would during a real flight. The only difference between HITL and a real flight was that the flight control software does not output the signals for the motors or other components to active. It was purely running the software on the board as if it was another computer. This was useful because different computers, especially when it comes to embedded computing, could ran code differently and had completely different outputs or bugs. One reason could be the kernel scheduler could change the order that threads ran. For this reason, testing on the real board that was running the software was the most accurate way to simulate our system.

Another reason that Gazebo was a good simulation environment for us was that it supports our entire stack. It was made were ran on Linux which makes it compatible with our ground control software. It was also designed to worked well with PX4 flight control firmware. As far as high-level functions went, Gazebo supports ROS, which we were using for several of the computer vision related tasks. The simulation environment was set up were easy to use and allows the developer to focus on the high-level functionality rather than the physics simulation and aircraft setup. It had several pre-configured vehicles including many different multirotor, fixed wing aircraft, land vehicles, and sea vehicles. This would let us pick the one that matches our aircraft the best rather than having to design our own aircraft. Gazebo also had pre-loaded environments to test obstacle detection and collision avoidance. Another nice feature was verbose debugging; We expect that our software may had some bugs that could cause it to crash. If this happens during flight or on the drone at any time, it was almost impossible to understood why it happens. On the other hand, using the simulation software with verbose debugging would tell us exactly what happened and why the software crashed. In addition to the previous features discussed, Gazebo was opened source and extensible, which means that, in the event that we ran into a situation where we needed a feature that does not currently exist, we were able to create it. This gives a lot more flexibility because it means that we were not particularly constrained by the software's current feature set. Overall, Gazebo seems like a great option for our drone software testing.

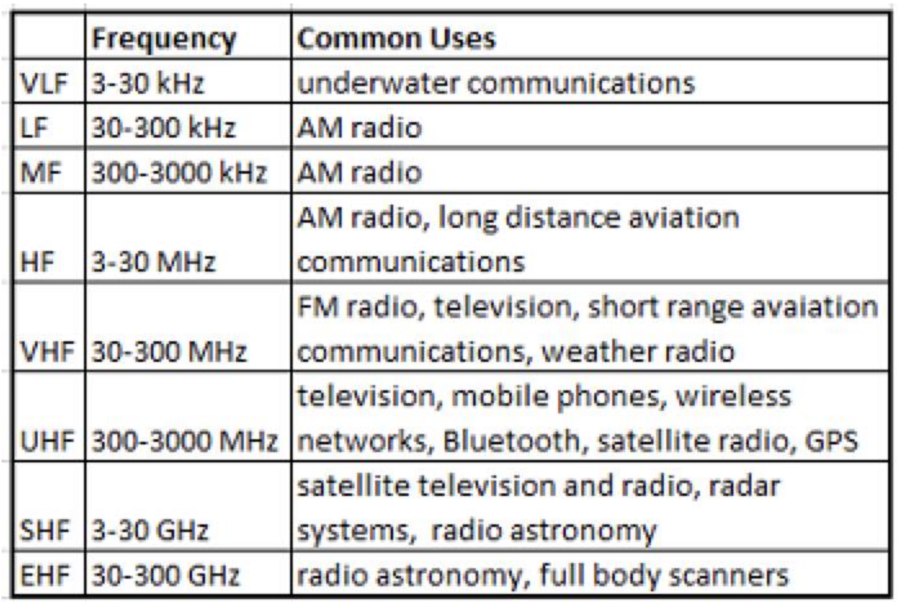
3.2.15.2 XTDrone

Even with all of the features that Gazebo had, XTDrone seems like an even better option for our simulation purposes. As mentioned in the previous section, Gazebo was opened source and extensible. XTDrone was one of these extensions. It seems like a better option for us because it was particularly targeted at our feature set. It was created as another student groups senior computer science project. XTDrone was a computer vision simulation software based on ROS, PX4, and Gazebo. It was conveniently already set up to worked well for what we tested, which was the SLAM algorithm and the collision avoidance algorithm. Like Gazebo, this software was designed were accurate and applicable to the real hardware that we would deploy it on. One of the main reasons we were selecting this software over the standard Gazebo was because of documentation. Surprisingly, XTDrone had significantly more extensive and more detailed documentation than the official Gazebo release. It was also more relevant to what we were doing. It also had pre-configured computer vision algorithms which were much more advanced than the scenarios in standard Gazebo. For these reasons, XTDrone would made it much easier for our group to test our actual drone software, rather than fiddle with the simulation itself.

3.2.16 Communication Types:

Choosing the form of transmission that was used to communicate to the drone comes with a couple options. The common ones were Wi-Fi, Bluetooth and regular radio frequency. For Wi-Fi, this option was our number one pick because of the commonality and built-in securities with setting up an RF network. Doing just straight RF set up could left the channel opened for others to overtake the drones, but essentially all these options were ways of communicating over RF. The next idea was Bluetooth as it had an ease of access with connecting to multiple controls, but Bluetooth typically had a range maximum on communicating to the drone, which could become dangerous if it needed to leave that boundary. Wi-Fi covered most of the checkboxes with being able to set up a network for the drone, cover a decent amount of area and available when purchasing the controllers instead of having to install the Bluetooth setup or spent more for the feature. We would also be using GPS to autopilot the drone to the destination, which operates on the Ultra High Frequency (UHF) levels with Wi-Fi and Bluetooth. Our flight controlling being the Navio 2 would have Wi-Fi installed already and made it the easier choice to went with. This allowed us to set up our own network when controlling the drone and giving it instructions. The **Figure 32** below shows that this would operate on ultra-high frequency.

Figure 32: Different Types of Communication



	Frequency	Common Uses
VLF	3-30 kHz	underwater communications
LF	30-300 kHz	AM radio
MF	300-3000 kHz	AM radio
HF	3-30 MHz	AM radio, long distance aviation communications
VHF	30-300 MHz	FM radio, television, short range avaiation communications, weather radio
UHF	300-3000 MHz	television, mobile phones, wireless networks, Bluetooth, satellite radio, GPS
SHF	3-30 GHz	satellite television and radio, radar systems, radio astronomy
EHF	30-300 GHz	radio astronomy, full body scanners

3.2.17 Sensors:

Sensors were going were a core part of the drone as it would assist in communicating to the drone that it was on the right path and also what objects was in the way. After research the list of sensor options included LiDar, ultrasonic, infrared (IR), GPS sensor, and time of flight. A common sensor used in Unmanned aerial vehicles (UAV) was ultrasonic. Ultrasonic uses high frequency sound waves to bounce off objects and detect the distance and size and so on. This helps and we tried to accurately determine the size of the object and the time it takes for the waves to bounce back determine how close it was. They helped detect objects because they were not a camera and do not have to identify color or transparency so works well in any lighting. They were restricted to a certain range depending on the one purchased and do not determine objects well with significant textures. Infrared (IR) sensors were another that were not effective by the amount of light in the area the drone was going through or detecting. This sensor emits an infrared light and then with a photodetector could determine where the object was positioned and how far away. Some advantages to IR were that they were smaller in size, which helps with not adding weight and size to the drone, and it could detect textures in objects better than ultrasonic. Again, it was limited to a certain range and could be affected by the environment and dense objects. LiDar comes in with having some advantages to the range and accuracy by being able to detect farther and could detect objects with texture and more 3D structure. Also, able to process the object faster as it uses smaller wavelengths and not affected by day or night. Disadvantages to the other sensors was the high cost and could be dangerous to use without proper eye protection. Time of flight sensors were another IR structured sensor but used an IR LED light to emit and calculate the object. Time of flight sensors had a higher cost like LiDar, but do not get affected by weather conditions, and if we had them could be used for 3D mapping and provide long range of sensing. In **Table 21** below, it summarizes some of the things to consider when choosing sensors. For our project, we were utilizing multiple sensors in order to made accurate decision making for the drone. Those sensors were finalized eventually, as we needed to evaluate which ones were better suited for the goals we had. Our goal was mainly to avoid objects that came in the way while also fight the best path of avoiding in order to kept it going to the same destination [7].

Table 21: Comparison of Sensors

	Long Range	Cost	Can handle External Conditions	3D Imaging
Ultrasonic	No	Low	No	No
IR	No	Low	Yes	No
LiDar	Yes	High	Yes	Yes
Time of Flight	Yes	Moderate	Yes	Yes

When comparing the sensors in **Table 21** and trying to decide what was needed for our project, Ultrasonic was the best for the job. Most IR was influenced by ambient light and with LiDar and Time of Flight being a little expensive, then ultrasonic was best for the job. Ultrasonic could detect objects and be very low cost to the project. Ideally, if there was a larger budget then LiDar was the next buy to made it more accurate for the job. Because of the choosing the Navio and the Raspberry pi then it included the Wi-Fi, GPS and altitude sensors and eliminated having to search for other options. We were using the Wi-Fi for security and communicating with the drone, GPS was used for navigation and location tracking, and altitude sensors would detect how low to an object or ground the drone was at.

In the end, we decided that LiDar and Time of Flight sensors were out of our price range. Since one of our main constraints was reducing the price of the overall platform, this was not something we were willing to compromise on. This left infrared and ultrasonic sensors. We were concerned with the efficacy of the infrared sensors in daylight, as the sun emits large amounts of infrared light which could overwhelm the sensor. This left us with the ultrasonic sensors. We planned to have an ultrasonic sensor in each direction of the drone, for a total of six ultrasonic sensors. While this could have helped in indoor situations where the drone might need to know about a ceiling or floor, or if a door closed behind it, we realized that it would not be very applicable for an outdoor delivery drone. There is no situation that an ultrasonic sensor would help us above what our camera will already provide. For that reason, we changed our original plan from senior design one, and removed the ultrasonic sensors from the platform. This left us with only the camera for obstacle avoidance.

4. Standards and Realistic Design Constraints

Drones were strictly monitored and had many associations that came with them in order to ensure the safety and standards of the machines. With this comes a set of standards and constraints to followed based off our goals. In order, to create an efficient drone many constraints were considered too had it maximize its ability, which goes into further details below. We also research standards associated with certain aspects of the drone to made sure it was not outdated and would not cause harm to the environment and people around.

4.1 Standards

When mass producing any product in any industry comes a set of standards set by companies and organizations to uphold the quality and safety of the materials being used and the product being produced. Even a certain screw being used might had a certain standard it needed to uphold. In this section, we talked about the standards identified with certain aspects of the drones and where those standards were coming from.

For the following standards they came from sources of organizations. A big organization that creates engineering standards was IEEE. IEEE stands for Institute of Electrical, and Electronics Engineers and the history of the organization’s roots went back to 1884. From then to now they had expanded exponentially and broken up into many boards and committees that do the brute research into what the standards should looked like and had a process to have them approved.

Another set of standards to looked at was for coding. Having standards for coding would help kept the life span and ability to update easier. A popular standard that was used for coding was the PEP8 standard. This stands for Python Enhancement Proposal, which comes from Guido van Rossum, Barry Warsaw, and Nick Coglhan, dating back to 2001. (INSET C++ STANDARD)

4.1.1 Wi-fi

Wi-Fi standards had been observed by the IEEE organization since the 1990s. We saw this in **Table 22**. The first of the Wi-Fi standard series starts with the standard 802.11 that was released in 1997. From this point on the naming convention adds letters to the end of that standard number/name. The first in the series was 802.11a, which utilized the 5GHz band, but came at an expensive cost. After 802.11a was made in 1999, soon came 802.11b which was on the 2.4GHz band and lowered the cost greatly. This naming conventions continues down the alphabet and all indicate different requirements that came with it and had different purposes depending on the used. For example, 802.11ah had the ability to operate on frequency bands below 1GHz. The goal with creating this standard was to allowed Wi-Fi were extended for wider range and lower energy used, and this standard was made official through publication in 2017. Another very specific standard for Wi-Fi was the 802.11ad. This standard was created to provided extremely efficient speeds under the 60Ghz frequency, but whereas the example before had longer range, this standard had extremely short distance and only able were communicated up to 11 feet from access point [8].

When it comes to the common Wi-Fi networks and what was used for this project, we would look at more recent standards. Standard 802.11n was capable of using the 2.4GHz and 5GHz and goes on to be created in 2009. This standard was what more modern routers used in reference to dual band capabilities. More recent and modern one was 802.11ac, which uses the 5GHz band, but also could access the 2.4GHz band by using the 802.11n. Both these common standards took advantage of MIMO, which stands for Multiple Input Multiple Output. This describes how on routers would have multiple antennas in order were able to handle the amount of data and traffic that was happening through the channels [8].

For our drone, the Raspberry Pi 4b had the necessary equipment to used Wi-Fi with our software and the drone. The Raspberry Pi model being used was suited were used with the IEEE standard Wi-Fi of 802.ac/n, which were the two described above. This allowed for decent speeds and long ranges of communication with the drone as it was flying.

Table 22: 802.11n versus 802.11ac

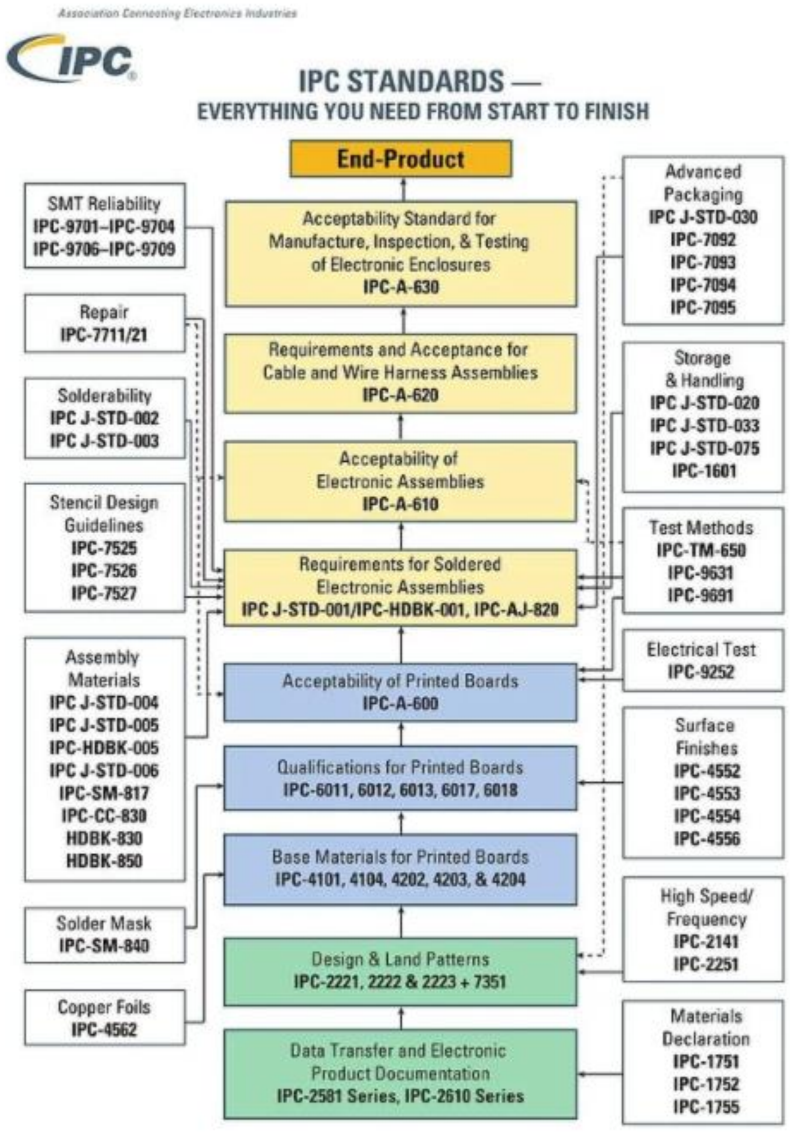
802.11n versus 802.11ac Features		
	802.11n	802.11ac
Band	2.4 GHz & 5 GHz	5GHz
MIMO	Single User	Multi-User
Channel Width	20 or 40 MHz	20,40,80,80-80,160 MHz
Modulation	64 QAM	256 QAM
Spatial Streams	4	3-4

4.1.2 PCB Standards

For our PCB or Printed Circuit Board, we used certain programs of circuitry to make our own Power Distribution Board or PDB. We used the standard given by Association Connecting Electronics. This company, just like another company talked about later in the paper, was made to produce and deduce standards for individual parts and methods for producing printed circuit boards.

We saw from looking into the standards provided by the IPC that it covers many aspects of PCB's and many such aspects we ourselves needed to use such as enclosures, cables ad their thickness, different types of soldering and which tools were used for such things, and finally different styles of documentation that we needed to submit for our PCB. We could see that every aspect of our PCB that we made for our own uses had to be up to standard given by the IPC.

Figure 33: Standard IPC Roadmap for PCB


Association Connecting Electronics Industries
IPC
IPC STANDARDS —
EVERYTHING YOU NEED FROM START TO FINISH
End-Product
Acceptability Standard for Manufacture, Inspection, & Testing of Electronic Enclosures IPC-A-630
Requirements and Acceptance for Cable and Wire Harness Assemblies IPC-A-620
Acceptability of Electronic Assemblies IPC-A-610
Requirements for Soldered Electronic Assemblies IPC J-STD-001/IPC-HDBK-001, IPC-AJ-820
Acceptability of Printed Boards IPC-A-600
Qualifications for Printed Boards IPC-6011, 6012, 6013, 6017, 6018
Base Materials for Printed Boards IPC-4101, 4104, 4202, 4203, & 4204
Design & Land Patterns IPC-2221, 2222 & 2223 + 7351
Data Transfer and Electronic Product Documentation IPC-2581 Series, IPC-2610 Series
SMT Reliability IPC-9701–IPC-9704 IPC-9706–IPC-9709
Repair IPC-7711/21
Solderability IPC J-STD-002 IPC J-STD-003
Stencil Design Guidelines IPC-7525 IPC-7526 IPC-7527
Assembly Materials IPC J-STD-004 IPC J-STD-005 IPC-HDBK-005 IPC J-STD-006 IPC-SM-817 IPC-CC-830 HDBK-830 HDBK-850
Solder Mask IPC-SM-840
Copper Foils IPC-4562
Advanced Packaging IPC J-STD-030 IPC-7092 IPC-7093 IPC-7094 IPC-7095
Storage & Handling IPC J-STD-020 IPC J-STD-033 IPC J-STD-075 IPC-1601
Test Methods IPC-TM-650 IPC-9631 IPC-9691
Electrical Test IPC-9252
Surface Finishes IPC-4552 IPC-4553 IPC-4554 IPC-4556
High Speed/ Frequency IPC-2141 IPC-2251
Materials Declaration IPC-1751 IPC-1752 IPC-1755

For our actual Board we paid more attention to certain standards that pertain more to us then others, such as 2221, which gives everyone the standard from which the general form of PCB’s were designed. The next, which when taking these into account of what their standards were for, were more of the regular laws that needed were looked at when designing any PCB’s, were 2615, and 6012. Going over them quickly, 2615 was the generic standard for how thick and wide PCBs were allowed were as well as the tolerances for how well these PCBs needed to perform under pressure and other conditions so that the impossible was not asked of a PCB. 6012 was the standard test that PCBs needed were able to reach were ready for consumer production. These things included performance and other qualification requirements.

Finally, when talking about the IPC and their standards for PCB we included their classifications for different uses of PCB and what that means for what kind of qualifications they had to pass were allowed into circulation and used. We were using ours for the power distribution of a drone’s power system, which means many things, but the most important thing was that our system was an aviation system when applied into the fullness of its operation, but when applied to this project only meets the needed of only a regular PCB system. That means that we put effort to reach the third level classification of PCB’s even when we needed to only reach the first level for used in our project. We saw through all of this in the different standard documentation our project reached, with the total standard roadmap shown in **Figure 33**.

4.1.3 Motor Design Standards

One of our key components of our delivery drone was our motor system. Without the proper motors were drone project we literally and figuratively would it made it off the ground. Due to the critical roles that the motors played we had done research on the Motors available for consumers well also looking into the standards in which motors were held to, to determine the motor specifications that were best for our drone delivery project. The motors that we judge were based off of the specification details provided by the manufacturer and with that information we compared the motors to make the best decision when we selected the parts for our delivery drone project.

Some of the important factors to notice on the specification PDFs provided by the manufacturer about the motors were electrical rating for voltages, power, and current. Other mechanical limitations for the motors to also paid attention to when comparing the specifications provided by the manufacturer were torque and RPMs variation. The specification details ascertained from the manufacturer were critical to making educated decisions on which Motors were best suited four our delivery drone project. The specifications helped us to elicit the best motors for the best outcome of our project.

There were plenty of regulations placed on motors, but it was the manufacturers obligation to meet these regulations. These regulations also known as standards were determined by industry and varying third-party associations like IEEE/ NEMA. Those third-party associations were IEC which stands for International electrotechnical commissions. The International electrotechnical commissions (IEC) had taken up the due diligence a fabricating standard practice technical restraint for electrical components internationally. Such electrical components that IEC had helped created standards for were Motors. The standards that IEC had created were to ensure that motors were being tested in rated accurately. The IEEE also helps check in regulate standards that were applicable to motors as well in regard to their testing, performance, and ratings. **[2]**

NEMA, also known as National electrical manufacturer association, had a very unique role in motor standards. For example, NEMA also design standards specifically to certain environments the motor was intended to operate within. Such as but not limited to certain enclosures to protect against like water and dust, so the Motors ran effectively in these environments. **[7]**

4.1.4 Flying standards

Drone flying had gone though many changes in recent years and would continue to change for years to come. Drones had been used in so many aspects of life that the FAA had passed many regulations and standards that operators needed to meet so that no laws were trespassed upon during operation of a drone. Prior to 2020, drones that exceeded a certain weight and size would require a pilot license that needed were earned through classes and money. This, at the time, even engrossed upon some sizes that commercial drones were able to achieve, so that many drone camera men and woman were hindered from my hobbies and job opportunities. Now that 2020 had rolled past, the FAA had enabled many drones that now were more mass produced in today’s age and used for different aspects were used without this pilot license, thus opening the doors for more different commercial drone companies, and our project itself, to took off without interference.

4.1.5 Programming standards

For our project we used Python and C++ for the most part to accomplish our goals. The Python would largely be used for the machine learning, autopilot and computer vision aspect of the drone. C++ was used for the code that communicates more closely with the hardware of the drone.

Python specifically had its own set of coding standards called PEP8 as mentioned in earlier section. This standard had been around since 2001 and helps Python projects be more consistent in nature since Python had an extreme amount of flexibility with how to type out code. This was one of the cons when choosing a programming language because with Java at its base level makes you followed certain requirements to perform different jobs needed. We knew we had to have a strict standard to abide to in order to had it be very understandable by reviewers and other teammates. PEP8 standard helps to outline how to space, naming conventions, the layout, even when to not followed PEP8. These guidelines and standards could be compared to how there were writing standards for papers in college and so on.

While these guidelines/standards went into extreme detail, **Table 23** shows a few of the rules that was followed or recommended. By looking at the details of the naming conventions below it shows how easier it was to distinguish elements like class, package, or functions. Benefiting the reader that had not touched it would provide ease of access for other members or future people observing. A technique we used to made sure this was getting accomplished was having a teammate who does not code often tried to read it over and followed what was happening so in the worst-case scenario that someone needed to come in and take over, they could do that with confidence. These provided a nice, neat guideline to easily scan and read and knew where things were located because everything had a spacing convention. All the elements were clearly labeled and laid out so there was no confusion on what might be happening.

Table 23: PEP8 Standards

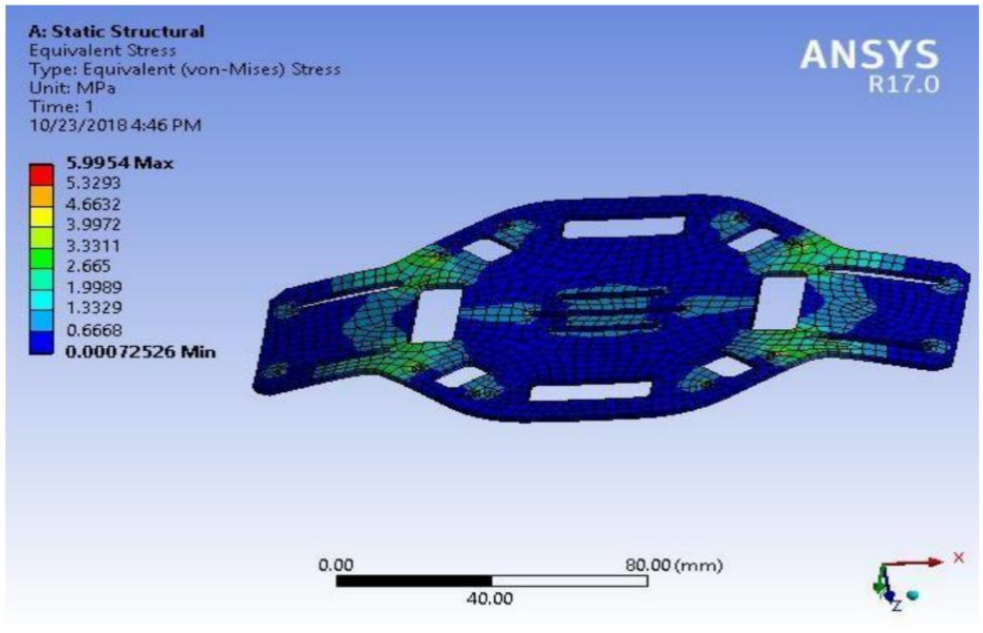
Item	Standard for PEP8
Function/Variable	Lower case, separate by underscores
Class	Start the word with capital letter, separate by underscores
Package	Short, lowercase word(s), DO NOT separate by underscores
Blank Lines	Top-level functions and classes with 2 blank lines; method definitions inside classes with 1 blank line; all other times – used sparingly
Maximum Lune Length	79 characters or less
Indentation	Use 4 consecutive spaces; used spaces over tabs
Line Break w/ Binary Operators	Recommended to break line before the operator, but could do after as long as all the code was consistent
Imports	Each import had separate line
Comments	Help with readability and be complete sentences
Source File Encoding	Use UTF-8

4.1.6 Battery Standards

Battery Standards for Lithium Polymer cells came from organizations such as IEEE and TSA, and even the US Postal Service. For the TSA we saw standards based around their transportation and storing used when in planes. The TSA requires that batteries that had equal or less than 100-watt hours in their cells when transporting or storing so as to not reach a state where the battery had enough charge within the cell to hurt passengers if damaged. Over 100-watt hours means one needed special permission to fly with these types. The US Postal service had the same kind of restrictions but for mailing and distribution of these types of batteries, with a further restriction saying these types of batteries cannot be mailed over air traffic but only upon land-based transportation. While the standard cell voltage of 3.7 volts was a “standard”, it was more of a physical standard than an organizational imposed one. The Lithium Polymer battery had more range than a set voltage. The 3.7 volts were the optimal and also usual voltage of battery cells in a LiPo. When a cell was near a complete discharge rate it was more in the range of 2.0 volts, and when fully charged it reaches 4.2 volts usually. However, both of these states could cause damage to the actual cell, so the standard was to hold it at the 3.7 volts mark so as to provide the longest-lasting product. We could state that the size and outlook of the outer shell of a battery was not a standard that was upkept as there were many different styles of batteries appearing, as well as many different styles of cell counts to different batteries.

4.1.7 Frame Standards

Figure 34: Tensile Strength of Baseplate under 20N of force



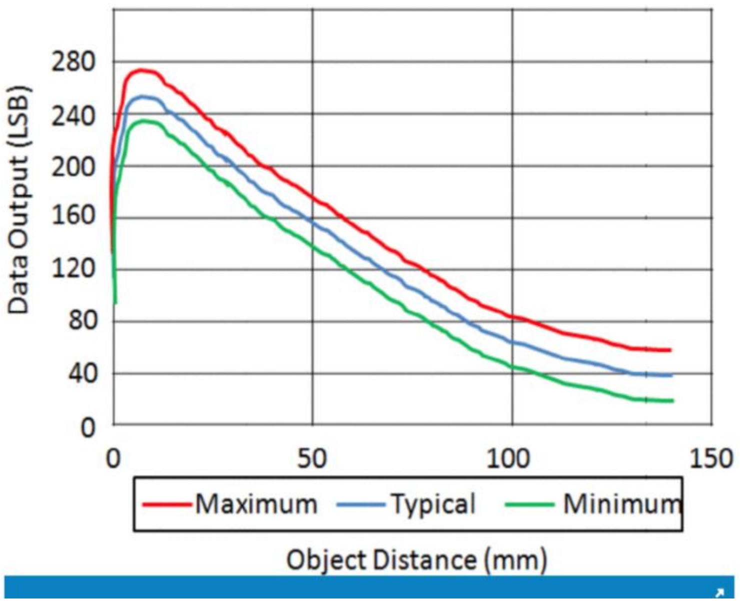
The frames we saw in the different sections of this paper all had a generic outlook to them that shows there needed were a type of standard to it. When looking deeper into the history and why this was so, we found the current frame was based around research done into the many different forces that would act upon commercial drones when in flight, such as gravitation force, deformation and stress upon the materials, and the weight of the drone and components attached led to the current design we saw in many drones today. This was not a standard per se as better frames could came to the market, and we saw there were slight variations to the frames themselves even in the frames we looked at in this paper, but through research done using a program known as ANSYS, which a simulation program used to stress-test different materials in differing ways, this frame style was the one that stood up the best. The branching out of the arms lets the mounted motors had more space and keeps the ratio of motor weight to propellor size to a more achievable quantity while giving room for components that needed were added into the drone for stable flight. We could saw the tensile strength in **Figure 34** of the base plate of a standard drone frame base plate when put under 20 Newtons of force. This shows the generic structural analysis and how the interaction of the base plate and the arms was well within what the materials and shape could handle.

4.1.8 Sensor Standards

IEEE had been able to provide standards for years, but for some of their standards, they were not made publicly available and required were purchased or subscribe to a subscription. They had many boards and committees that worked on creating these papers and researching what was the best to standardized and makes sense that they would need a way to capitalize on their product to keep the quality of what they were outputting and giving to people and companies.

One of the latest standards on sensors was IEEE 2700 which was released around 2014, but not released to the public. The description and overview of this requirement went over standards for sensors and how the performance should be, and these helped with integrating with systems, like how we were connecting them to a drone. The standard covers sensors including “accelerometers, magnetometers, gyro meters/gyroscopes, barometers/pressure sensors, hygrometers/humidity sensors, temperature sensors, ambient light sensors, and proximity sensors.”

Figure 35: IEEE 2700 Standard, Proximity Sensor



From researching the standard as much as possible it seems like it gives guides to how the sensors on machines would perform in certain circumstances and when set up a certain way would output stats like the below image. **Figure 35** uses the analysis of proximity sensors to show how at a certain distance it would give a max output of data, but then slowly started to decrease as the distance becomes larger. We were using an ultrasonic sensor as one of our proximity measurements to detect objects in addition to a camera. Having these standards implement, if possible, was beneficial to knew what restrictions we were dealing with. For example, if the below figure demonstrated an ultra-sonic sensor, then we knew that the quality of data and detection after about 40mm starts to extremely decrease.

4.1.9 Transportation Standards

When it comes to standards on transportation in reference to drones, they were two major organizations. One of the organizations to helped set the standards on transportation in the United States was the department of transportation otherwise known as DOT. The second organization that set standards on transportation for our project due to the fact that this was a drone delivery project was the FAA otherwise known as the Federal aviation administration. Now the FAA and DOT I'm currently in the process of beefing up standards around drone transportation because it was a fairly new revelation in the industry. Due to the fact that drone delivery was so new to the industry standards was still being created around it.

While the FAA does help created standards for drones, they were more worried about the airspace the drones took up rather than the transportation factor. So, the FAA was more concerned about making sure you as a pilot and the drone was also known as on manned aircraft systems or UAS for short were taking up the correct about of air space. The list of regulations that the FAA tries to enforce were as follows Do not fly over 400ft, or flying above groups of people, and above all else and the most important were registered and had passed the part 107 exam. **[15]**

The department of transportation (DOT) deals more specifically with the transportation side of our drone delivery project. One of the new standards the department of transportation had recently pass on drones was to have the drones had a form of ID attached to the physical drone. The reason that they wish to attach an ID to the physical part of the drone was that so if there were any damages or accidents it was easier to found who was liable for those damages that may or may not be caused. **[15]**

It was also considered to mention the department of transportation they only such regulations but also tries to push for certain agenda. One of which was artificial intelligence usage in drones. They incentivize artificial intelligence usage and drones by offering to fund and making the information easy and accessible.

4.2 Realistic Design Constraints

With building a new project and idea it was easy to get carried away with what features and items to include with it, but that cannot always be the mindset. When creating a product there were certain constraints to consider and thought about how to product was impacted by the following factors in order to made it helpful for society, environmentally sustainable and maximize its features. For this section, we had broken down the different categories of constraint topics that were considered when designing this.

4.2.1 Economic

We were developing a product that was the best used was bought in bulk and our target audience were distribution companies. While our current drone design budget was $1000, we expected to retail a single drone for $3000, and as you purchase more in bulk it becomes cheaper to produce, thus making the price of the drones fluctuate and become cheaper.

Table 24: Economic Budget

Item	Quantity	Cost
Drone Frame	~2	~$50
Battery’s	~2	~$20
Flight Controller	~1	~$200

Camera	~1	~$65
Blades	4	~$10
Motors	4	~$30
ECS	4	~$20
PDB	~1	~$30
Total		**~$650-$700**

In regard to our drone, we had capped our budget at $1000. With our budget cost entailing the drone frame, Batteries, Flight Controller, Camera, Blades, Motors, ECS, and PDB as seen in **Table 24**. With a relatively high budget, we were sure to produce a high-quality drone with high-quality parts. With the goal of this product were mass-produced, we believed the budget being $1000 would give us some wiggle room with other surprise costs that came with such things as manufacturing and selling of the drone otherwise known as postproduction.

For post-production, we would like to invest in a sales team to consult with a distribution company to sell them on all the benefits our drones were able to bring to their business model. With the estimated retail cost being $3000, we would wanted about $500-$600 of the total cost to went towards funding the sales team and to maintain good standing with our clientele.

Even with starting local, we still do have the vision to bring our drones to a global market. The potential for drone delivery in the highly dense community was limitless, as well as our product be trained and tested at this point, the retail cost in foreign markets would increase. This higher rate of $3,700 would also accommodate for the cost of shipping, international tariffs, the personal cost of going into a new market, and the design accommodation that may be needed for the new locations. Such as changing the user manuals and programs to used different languages specific to that country.

4.2.2 Environmental

When introducing any new technology, it was important to investigate its environmental impact. Especially technologies which were meant were deployed at a large scale. Understanding the impact of our product would enable us to focus on the aspects that benefit the environment while simultaneously minimizing the negative impacts.

When analyzing the environmental impact of our drone, it was important to compare it to competing solutions. Our product does not exist in a vacuum. In our particular case, this means that packages were delivered whether we deploy our drone or not. Last-mile deliveries were typically carried out by large trucks which ran on diesel. In order were considered a net positive for the environment, our product simply had been less harmful than delivery trucks. As it turns out, this was not a clear-cut scenario.

In a study from the University of Washington, it was shown that delivery drones emitted less carbon dioxide than delivery trucks on the same route, even though the drones traveled a greater distance. This was due to the lighter design and more efficient electric motors. However, this effect was only seen on shorter routes with light packages. The opposite was shown when large packages were delivered over a longer distance. A similar study reported by the LA Times showed the same results. They explained that this was because drones could only carry single packages to a single location, whereas trucks could carry many more. This increases the efficiency per package for trucks. They showed that trucks were also more efficient in densely populated areas whereas drones were better for sparser areas.

A third study by the Smithsonian reports similar findings in terms of the effects of geography on environmental impact. They also concluded that the method of power generation for an area's power grid had an effect. Because drones rely on batteries that needed were charged, the power grid's efficiency was a major factor in the overall efficiency of drone delivery. In some areas with less harmful power generation, the drones had a 54% decrease in greenhouse gas emissions over delivery trucks per one kilogram of payload. However, in more harmful areas that used coal to generate power, the drones only produced 23% fewer emissions. This means that the environmental impact of delivery drones was dependent on where they were deployed. It also means that as our power grids became greener, the delivery drones would as well. This was good compared to delivery trucks that rely on diesel engine innovation to pollute less, which had remained relatively stagnant.

4.2.3 Social

Thinking about the social implications of delivery drones it could be thought of in pros and cons. The pros to having delivery drones were faster packaging and more efficient delivery from company to customer. There were cons that surround drones which provided constraints when approaching the design and style to them. Within the US specifically, many people value their right to privacy hence why drones were limited by the government and had extremely strict rules when it comes to flying in certain areas. Because of these constraints if we wanted to deliver in certain areas or places then we would have to research those social constraints otherwise the idea could fall flat.

From a customer standpoint, if people were to trust the product and trust that the drone's purpose was for delivering packages and not for spying then in our designs, we would want to keep it as secure as possible. Having an extremely secure drone would help customers felt like they could trust the product and company did what was it intending and nothing more.

Thinking about the constraints with the software design, it had to have to be held to a high standard of accuracy and correctness. These drones had the potential to crash into objects or people and potentially started fires if crashing in the wrong place at the wrong time. Because of these protocols would need were considered for taking the best decision. Worst case scenario it might even needed to make the best worst decision that causes the least damage.

These drones were meant to provided ease to everyday delivery and was flying around everywhere. With constantly being around people a social constraint to consider was the noise effects. A lot of environments and social happiness comes from peace and quiet, but if the drone was obnoxiously loud then it would disturb the peace of some people. We were trying our best to have a quiet drone.

4.2.4 Political

Now it wouldn’t be very American if the government didn’t try to legislate and tax every aspect of our lives, and drones were no different. To be able to fly these drones someone would have been certified with an FAA part 107 exam which costs about $150 per exam. Also, because laws around drones were still new and not many had tried in court, we and the company we sell to would want a lawyer. For this reason, we made sure one of our group members got a license.

The reason why you would want a lawyer was because of drones flying in residential areas and over people and roads. The chances of something unpredictable happening was high. They the law was currently written was that whoever the owner of the drone, was liable for the damages. There were also some areas that flying over was illegal such as theme parks, military facilities, and airports to paying attention to geographical location and path routes were important.

Even then these drones were opened to many liabilities so we would take plenty of preventable measures to avoid litigation and to made sure we produce the safest drone feasible. Another concern was that legislation hasn’t completely caught up with the drone technology of today so the laws we face today could not be the laws we face tomorrow.

4.2.5 Ethical

Ethical livelihood was an important concept in the engineering world because of its implications when disregarded. Ethics were a set of “rules” or social cues we as a people were taught or inherently understood through living within society. For engineers this was slightly the same, however, there were certain aspects related to research and development and how we should ethically do these things as engineers that was known to a set style of rules. These included quality insurances so that when doing research and finding results your findings and techniques could be reviewed and recreated and thus prove to the wider engineering community in your truthfulness. Also, pro quo because of this outlook was the ability to have the moral “high ground” when doing research, thus providing reliability to your research and worked. This was done by not testing on Humans, and only testing on animals when absolutely needed, hence when this invention or research directly affects human’s health and lifestyle, thus such a level of research was needed and even then, this needed were done as humanly as possible. And when all other routes had been investigated and human trials were the only route, agreements needed were made between the researcher and the person were researched about what was and what was not allowed. This was handled through paperwork and a transparent outlook on what was happening.

Ethics yet entitle not just the aspect of human health and safety, but economic and social responsibilities as well. Engineers, specifical ones within government agencies and places that had given access to civilian tax dollars had an ethical responsibility to not waste money and supply reasoning for why they were spending money were certain things versus others. This was to kept engineers accountable and not buy their way out of problems that crop up in research and development that could be solved using inventing and innovating.

Engineers needed to post their findings in public discourse and with empirical evidence to show their honesty and freedom from bias. This format they submit their findings in needed were too standard and showed meticulous attention to detail so as were completely repeatable and easily understood or as it was known, “dumbed down” were better understood when given the opportunity. This format would also be completely transparent in the way this information was helped along by outside sources such as fellow researchers or graduates that helped brought this project to fruition so as to properly showed gratitude were due. Citations from sites, links, and acceptance from sources to use their research needed were given were ethical.

In the terms of our drone and its references to the ethical outlook and needed for research, there was an aspect of research into the guidance and accident-avoidance system that may needed were researched in the future with regards to how it interacts with human heat signatures and outlines, and such that was, this drone needed were properly documented in the right format and transparently shown as were easily taken up by future researchers that needed to looked into future aspects of this project. To further looked into our drone and its potential ethical impact, we used many sensors to provide guidance and telemetry as well as used these sensors as the backbone of our accident-avoidance program, and these sensors could be used for non-ethical purposes if designed that way. To avoid this and continued ethically with the project, we would not be recording anything intercepted by the drone for future reference, so as to kept personal privacy to our topmost concern.

4.2.6 Health and Safety

Health and safety were paramount to any aspect of research and innovation. As such, products were always included with health and safety warnings whenever these two aspects could be affected detrimentally to a human or animal. These warnings included such things as drawbacks that came with using such products. These could be seen in commercials when pharmaceutical companies provided a list of side effects that could be incurred from using their product, no matter how minuscule the chance of happening, so as were transparent. This same instance could be inferred through materials used such as anything radioactive, chemical, or potentially hurtful when touched or inhaled. Such things would apply to our drones and products like it. The materials we used to make our product could had adverse effects on livelihood, such as when manufacturing the completed product, or when taken to more in-depth analysis, the mining and acquiring of materials used could had adverse effects as well.

To showed more of how health should be shown on product placement and usage, the aspect of using products could had adverse effects as well, such as the output of CO2 when using diesel or gas engines. For our product, this was not an issue as we were using drones ran by electric batteries and motors. This does not meant drones do not had impacts on the environment when in used, as the electricity to power this drone was needed from other resources on the power grid itself, but we could saw that when taken to the extreme case we could label anything and everything as detrimental to the environment and users in general. So, for a professional outlook, we kept to closely related subjects that affect personal health directly in the used of our drone product.

Now when showing the positive increase in health-related aspects of our drone, we could turn to statistics and materials used. The drone product was much smaller and more easily produced than delivery trucks used by most companies, thus keeping more materials out of the environment instead of needing them to fulfill higher material needed for bigger products. Continuing with aspects that were positive improvements, through statistics we saw a correlation to a decrease in suicides when taking the idea that there were fewer retirements for delivery workers when there were fewer smaller delivery routes (as this was the demographic used for our drone) thus decreasing needed were away from home and increase the mental health of delivery workers in response. **[2]**

Safety was an aspect of engineering that we as producers of products needed to look into steep consideration because it was our products and thus our actions that could affect the livelihood and safety of other people. This was shown in the way we process and package and generally affect our products. We needed to take into account the safety of not only the person, but the environment, property, and investments. We do this by applying warning labels and color-coded codes that showed certain aspects of danger. Such as the radiological sign for radiation leaking, or the skull and crossbones for chemical aspects that could hurt individuals. We needed to also give out safety manuals to show the correct way to operate our products and instructions on if the events that if safety was compromised, the user was able to help others and or themselves.

Our Drone had an autopilot feature that would enable the drone to fly from pick up point to drop off point with little human interaction, and while that means little chance of human error, that means that there was a possibility for machine error. This was taken care of with our accident-avoidance software that lets the Drone avoid and dodge upcoming buildings and other obstacles in its way. This program would limit property damage, and the height level of flight would made it safer against human accidents, with the only chance being when the drone was taking off and landing. These instances were shown in the provided safety manual that was our responsibility. This would provide the safety we needed to insure for property and human health. Regulations and more safety warnings were what would protect the environment, thus completing our aspect on safety.

4.2.7 Manufacturability

Manufacturability was an important consideration in any engineering project. This included our ability to acquire resources, tools, and supplies needed in the manufacturing process. It also involves the speed at which parts could be made and the ease of assembly. This was important to consider because it ensures that our product could be feasibly used to accomplish our goal. If we cannot manufacture our delivery drone fast enough or reliably enough, then it would not be able to serve its intended purpose, even if each individual drone met all of our requirements and specifications.

For this project, many of the mechanical components were standard and could be acquired off the shelf. This included the frame, motors, ESCs, and batteries. Some other components like the flight controller board were more specialized. For our project, we were using commercially available solutions. However, at scale, it was beneficial to create a custom flight control board. This would reduce unit costs and allowed for more flexibility. Our drone also uses off-the-shelf sensors such as ultrasonic sensors and cameras. These sensors were connected via standard communication protocols like UART which allows for ease of integration as well as the flexibility to swap them for other sensors.

Choosing the Navio2 and Raspberry Pi as our flight control board drastically increased the manufacturability of our product. Having an all-in-one solution, as opposed to a flight control board and a computation board, allows us to integrate that piece into the drone much easier. It also makes debugging simpler. Overall, our delivery drone was relatively easy to manufacture. It had no specialty parts, and the parts were designed to worked together well.

4.2.8 Sustainability

Drones had been proven were very sustainable on paper. Because of the amount of technology available and the emphasis on the environment, if drones were not kept to that environmental standard, then they would not be sustained. When designing these drones keeping that constraint of the environment would help led to more sustainability. These drones were predicted were the future and helped the environment as they left extremely small amounts of carbon footprints behind. Making sure to have a clean charge source was a constraint to carry on the sustainability because using renewable energy as the charge source helps with the longevity of the drone.

In order to kept it sustainable, it also needed were built as stable as could be so that it would last as long and also one thing known with drones was crashing so making that to not impact the area it was in would help keep it flying as long as possible. Having as many components as possible with similar lifespans did help the lifespan of the drone overall because if enough parts fail at once then mass production companies might saw it more cost-effective to throw away instead of salvage. This constraint was one that contributes to the design process of the drone and considering making parts that needed repairs, easily accessible for quick fixes, and lower the carbon footprint of broken or dead parts.

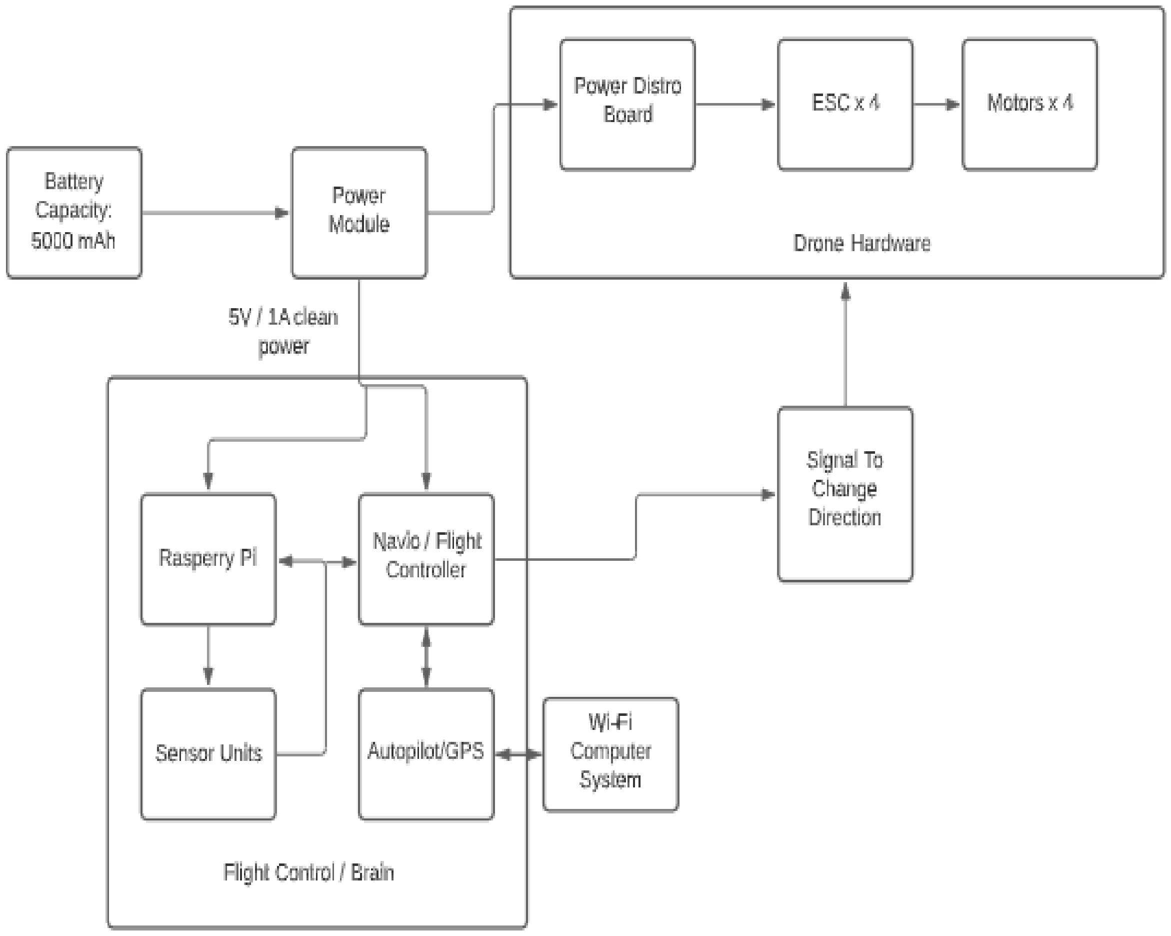
The software setup would require a lot more effort to ensure sustainability. Typically, Java was more known for having a better foundation of software set up for projects, which leads to less maintenance down the road. Despite this, we still went with the decision to used Python because of its ease of use and readability. We were hoping these factors would help made it more sustainable in the long ran as long as there was time and effort put into setting it up the best way possible. Knowing our goal was to make this software had a long lifespan and easy update feature, it would require us to had detailed documentation within the code and not just writing it to worked.

5. Project Hardware and Software Design

After diving into what goes into a drone and the components made up of it, there could be the starter of design and decisions. This section goes into what decisions we made and how we wanted to design our drone in order to made it an efficient automated delivery drone. The hardware designs here went over how we were connecting and securing everything into place and why we were using the parts we chose. The software was outlined how we were going were setting up connections and using the sensors with the software. These designs were important to the records of the project as to saw what we would need did if starting from scratch. Designing was where the main decisions were happening with the unique choices we were trying to implement and explain how to recreate the idea we were trying to portray.

5.1 Hardware Flowchart

Figure 36: Hardware Flowchart

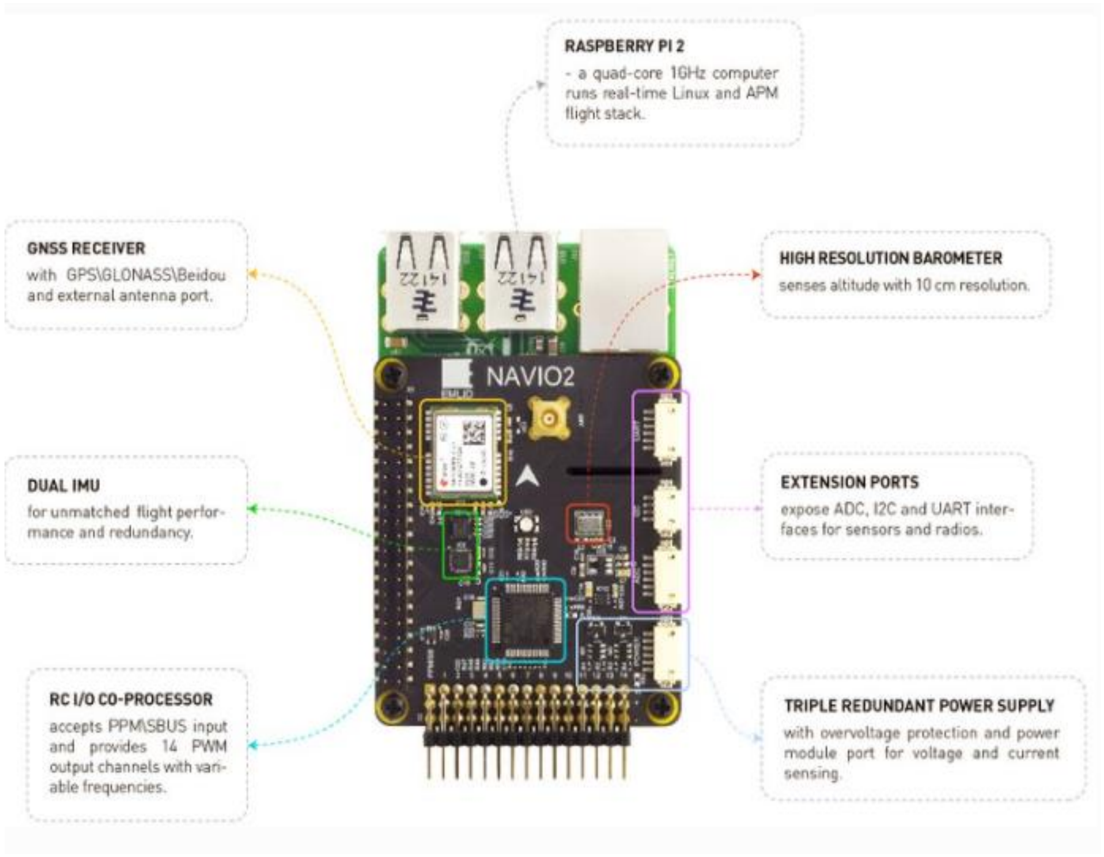


When we took into account multiple routes for data and power to flow throughout our system, we needed to input a flowchart to simplify this measure and made it much easier for the users to understood where and how our thinking was when designing such a project. Here we saw in **Figure 36** how we went about making our flowchart, including things like power lines and where certain currents and voltages would flow, to data lines and how our flight controllers and sensor readers would accept and send data to the rest of the system.

5.2 Controller Design

The flight controller was set up with the raspberry pi 4 b and be able to communicate with the software that had machine learning and computer vision for object detection. The design followed what was recommended in terms of what needed were connected for proper communication. We would also be using the Wi-Fi setup as a way to communicate instead of Bluetooth or other options.

Figure 37: More Detailed Navio Functions


RASPBERRY PI 2
- a quad-core 1GHz computer runs real-time Linux and APM flight stack.
GNSS RECEIVER
with GPS\GLONASS\Beidou and external antenna port.
HIGH RESOLUTION BAROMETER
senses altitude with 10 cm resolution.
NAVIO2
DUAL IMU
for unmatched flight performance and redundancy.
EXTENSION PORTS
expose ADC, I2C and UART interfaces for sensors and radios.
RC I/O CO-PROCESSOR
accepts PPM\SBUS input and provides 14 PWM output channels with variable frequencies.
TRIPLE REDUNDANT POWER SUPPLY
with overvoltage protection and power module port for voltage and current sensing.

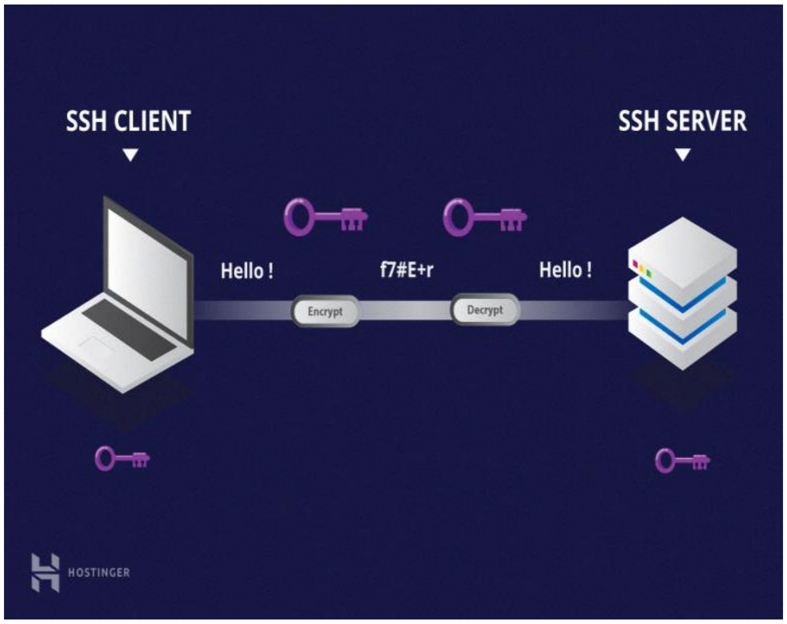
We had in the pin layout, that the total pin layout for our designated flight controller, the Navio 2, was very encompassing and could handle many different outputs as we could saw. It also shows in the pin layouts to the left of the board as seen in the board itself that this board was easily integrated into the Raspberry Pi and had no problems with having to share data between boards in heavy activity such as when camera vision was occurring, and the flight controller needed input to drive the drone. We also saw the ease of understanding what was on the board, because of how under the board we had an easy-to-understood guide to what each pin was mostly used for. We had ease of power input with the USBs and saw that our battery needed a power module that could convert from the leads given by the battery to USB format for ease of use and gives us future aspects of the board that we needed to think about. We could see that 5 Volt pins were spread through the Raspberry Pi integration section of the Navio 2 board as well as 3.3 Volts, showing these two levels of voltage were the most commonly used for this board and what we needed to ensure our power module that was converting power form our battery to our board was attuned too. Further on, we could see the receiving and transmitting pins, which lets us make an easy estimate for where our pin layout needed were and what pins were providing data to the flight controller when needed, thus giving us an easy-to-understand troubleshooting test for when or if we had problems providing and receiving data in between the boards.

We saw in **Figure 37** that the Navio 2 was a very high-class flight controller that included many different sensors included into the board such as a barometer and other such similar and more simple sensors, leaving the ports for more useful sensors that we included later. Our power USB ports also showed that the Navio and Raspberry could share power input because of a designated port from the Navio 2 to the Raspberry Pi. This would simplify how many lines we needed to run from the designated power module between the battery and the PDB and to the flight controller and the data processing board. Further, the built GPS receiver would let our autopilot system that would take input from our accident-avoidance system be very intuitive and more of a simplified process. We did not need to input our own system and be able to simplify our data processing and coding problems.

5.3 Wi-Fi Communications Design and Flowchart

Through hooking up the Wi-Fi component with ethernet and further downloads it had its own Wi-Fi network connection, which was used download controls and other parts to it. Both our ground control laptop and our flight control board (Raspberry Pi 4 B) were equipped with a Wi-Fi module. This normally this could be used to connect to an external Wi-Fi network. However, we were flying our drone in fields outdoors, where it was likely to bad poor, or no signal. This was not a problem, though, because the Wi-Fi modules could be used in another way. They could be set up to actually transmit a Wi-Fi network similar to how a hotspot works on a smartphone. We would set up a hotspot on the ground control station laptop and then had the drone’s computer connect to that Wi-Fi network. This means that even though we do not have a connection to the internet, the two computers could talk to each other wirelessly. After the two computers were connected on the same Wi-Fi network, we could remotely log into the drone via Secure Shell Protocol. This would allow us to securely access and issue commands to the drone’s computer without the risk of being hacked and taken over. The architecture of the secure shell protocol was shown in **Figure 38**. It works by encrypting traffic between two clients on the Wi-Fi network, so that any other computer cannot read the messages, even if they were intercepted. The secure shell protocol exchange was used to send and receive MAVLink and ROS messages between the drone and the ground control station. This allowed us to control the drone manually as well as send it updated mission requirements. It also allowed us to receive telemetry data about the drone such as the battery voltage, altitude, pitch, and speed.

Figure 38: Secure Shell Protocol Architecture



5.4 Limitations

Limitations were something that engineering projects would always try to minimize but would never succeed in completely removing. There was a factor of any good product and could cause growth and ingenuity in the design of products. Here we worked within our limitations for the future of our drone project and showed that we could overcome such limitations with careful planning and ingenuity.

5.4.1 Low Power Mode

Power was not infinite and needed to always be regulated to ensure that proper coordination of our limited power supply was directed to its needed target and that we as engineers perform a style of triage to our own systems so as to knew where and when our power was needed most for continued operations.

Now our drone would of course in the lifetime of its operation came to a point of battery life that it would enter such a state as low power. This cannot be avoided and was even encouraged to some degree because of our batteries style of design, and most others as well, that showed that depleting a battery to the limit of its capacity and then charging it to fullness extends battery life to a figure worthy of this procedure was taken into consideration.

During operation, our drone would enter a low power state sometime during operation, and it would send a signal to ground control with its last known location as well as the notice of low power. It did not continue operating as it would objectively weigh itself a higher priority than the timely delivery of its package and would turn around and return to the known location of ground control. This was a standard procedure and was followed and implemented numerous times during flight operations to ensure the program was in full working order.

5.4.2 Data Limitations

Bluetooth was a very low bandwidth style of data transfer. We would never be able to send large amounts of data through our ground control to our software system, namely our flight controller, at any reasonably fast speeds. That was a limitation that was being overcome by our drone's reliance on itself most of all. By giving our drone the ability to achieve decisions on its own and react in time to a fast-paced environment we were setting up our project for success within our limitations. When we needed to supply our drone with direct data input from ground control, our drone was a closer vicinity to ground control so as to limit data travel times and thus increase productivity.

5.4.3 Range Limitations

When talking about range limitations, we need to talk about not just Bluetooth and its limiting bandwidth, but the physical range limitations of our drone and how long it could run. We knew from Bluetooth's own specifications that the 4.0 variation had a range of roughly 300 feet. This was quite a distance achieved and shows limitations that our specific ground control had to adhere to. We knew then that our ground control could only achieve direct control up to 300 feet. Continuing past that range was directly enacted by the autopilot system.

5.4.4 Background Interference Limitations

Technology needed to consider the traffic going on around it constantly. Even when using home Wi-Fi networks, it did have interference from things like the walls, other computers, or even the weather. That same concept applies to the technology and components being used on the drone. We were using a Wi-Fi network for our flight controller, which was set up in a secure way to get rid of the interference from other people trying to connect or took over. This was especially important for the future of drones because, with package delivery, we needed to prevent packages from being stolen, and with that comes securing the drone from background interference that might made it vulnerable to people taking over.

Other background interference the drone could face was weather conditions from the environment and extreme data traffic. Data traffic could include cell phone towers, data being sent through other internet signals, or different wavelengths in the atmosphere. We did make sure the drone was wired correctly to have a strong connection to itself and the computer attached to it.

5.5 Sensor Chassis Design

When it comes to the sensor design, we placed them to have max exposure to what it was capturing. In order to securely do this, we needed to hold it down by having a chassis design, duct tape, or zip ties. To begin imagining the setup of the sensors we went with temporarily securing them with duct tape or zip ties and if seeing it needed a more permanent home then using the resources available to 3D print a chassis that would screw into the frame.

With the drones it was quickly moving at times to avoid objects and needed were able to endure that jerkiness or even withstand the object it was trying to avoid. Luckily with the technology now a days it was more accessible to had 3D printers through colleagues and fellow students that was able to print a design that could be securely attached to the drones. **Figure 39** shows an example of a bracket/chassis that was able to securely hold an ultrasonic sensor, which was what we were using it for also. As it was able to saw, there were holes for screws to attach it to a frame item and for the sensor itself, which would made sure it was held steady while trying to perform measurements and distance of objects.

We believed coming up with a chassis for a drone were critical. The reason why it was so important for a drone to have some protection from the elements were as follows. If we do not have a secure design that would protect the inner workings of our drone the chances of a component being damaged during testing skyrocket. The reason it becomes more liable to not had a chassis was due to the fact if the drone was to crash, then came into contact with dirt and water other debris it could had a chance of ruining A component or two and then delaying progress in our project. That was the reason why adding a casing over the main components of a drone was so important.

Now when it comes to designing the chassis for how the components fit there were quite a few requirements we would like it to have. We do intend on 3-D printing our chassis. When 3-D printing a Chassis that would cover the components of our drone, our chassis needed were durable enough to survive the crash but light enough not to cost too much drag and weigh down the drone. We intend to design our 3-D CAD of the chassis with 40% Fill. A 40% fill would do was save plastic, added structural integrity, while also not adding too much weight. What 40% Fill means, was that only 40% of the interior walls of the chassis was implemented, just leaving 60% of the interior walls were empty.

The next important decision that our group made about our chassis was what filament to print our chassis with. Now when it comes to 3-D printing filaments there were hundreds to choose from. The two most common were PLA and ABS. ABS filament takes a higher temperature for the extruder to properly lay down the ABS filament, but it also was more durable. The higher extruder temperature causes there were more plastic fumes, harder to print with, and overall, less user-friendly. While PLA was more user-friendly, requires less ventilation and less harmful plastic vapors emitted, and was overall more user-friendly. This was an important comparison and needed were carefully looked into. You could saw these compared in **Table 25** below.

Table 25: Filaments

Filaments		
	PLA	**ABS**
Printing Temperature (Celsius)	190-220	240-270
Harmful Fumes	No	Yes
Price per Kg	$19	$21
User Friendly	Yes	No
Durability	Fair	Very

So, in conclusion our group believes a chassis was important. The reason why was important was to protect it from water dirt and other debris if it was to crash as well as protecting the longevity of the components inside the chassis. Our chassis was made up of PLA filament printed at a 40% fill to made sure our chassis was light but also durable.

Figure 39: Ultrasonic Sensor Bracket

5.6 PCB Design

The PCB Design for our drone delivery project played a critical role. The critical role in which we would design a personal circuit board was to design a power distribution board.

The software that we plan to design our personal circuit board on was the eagle auto desk. With the eagle auto desk, we were able to upload libraries from Webbench. Webbench was a website that had already planned out schematics that we were able to use for our drone delivery project. Our power distribution board needed were able to take the power delivered buy the battery and distribute it to 4 different ECS evenly and safely. The way we intend did this was by using voltage regulators schematics from Web bench and implementing them in a functional Design on Eagle.

Once we had a functional cad design on the Eagle of our four-voltage regulators from the web bench we would then proceed with manufacturing our personalize circuit board. When it comes to manufacturing or personalize circuit boards there were a few options for us to choose from. Our values on which we made our decision efficiency, cost of shipping, and how long it may take to ship. As you could saw in **Table 26** below the companies we might potentially pick from.

Table 26: PCB design

Company	ETA arrival	Cost	Shipping	Total	Location
PCB Unlimited	4 Days	$23.64	$34.53	$58.17	China / Taiwan
PCB Way	4 Days	$38.00	$19.00	$57.00	China
JLCPCB	3 Days	$23.50	$32.30	$53.80	China

As you could saw in **Table 26** above. The companies above were the ones in which we were comparing to saw which was best for our personal circuit board. The companies in question were PCB Unlimited, PCB way, and JLCPCB. PCB Unlimited had the option of manufacturing in China and or Taiwan with the total coming out were estimated at about $58.17. The total for PCB unlimited comes from a shipping cost of $34.53 and a manufacturing cost of $23.64. PCB Way was manufactured in China with a total for the personalized circuit board being $57. The total for the personalized circuit board from PCB Way comes from a shipping cost of $19 and a manufacturing cost of $38. JLCPCB had its manufacturing done in China and the total for our Personalize circuit board was $53.8. The total for our personalized circuit board from JLCPCB comes from a shipping cost of $32.3 and a manufacturing cost of $23.5.

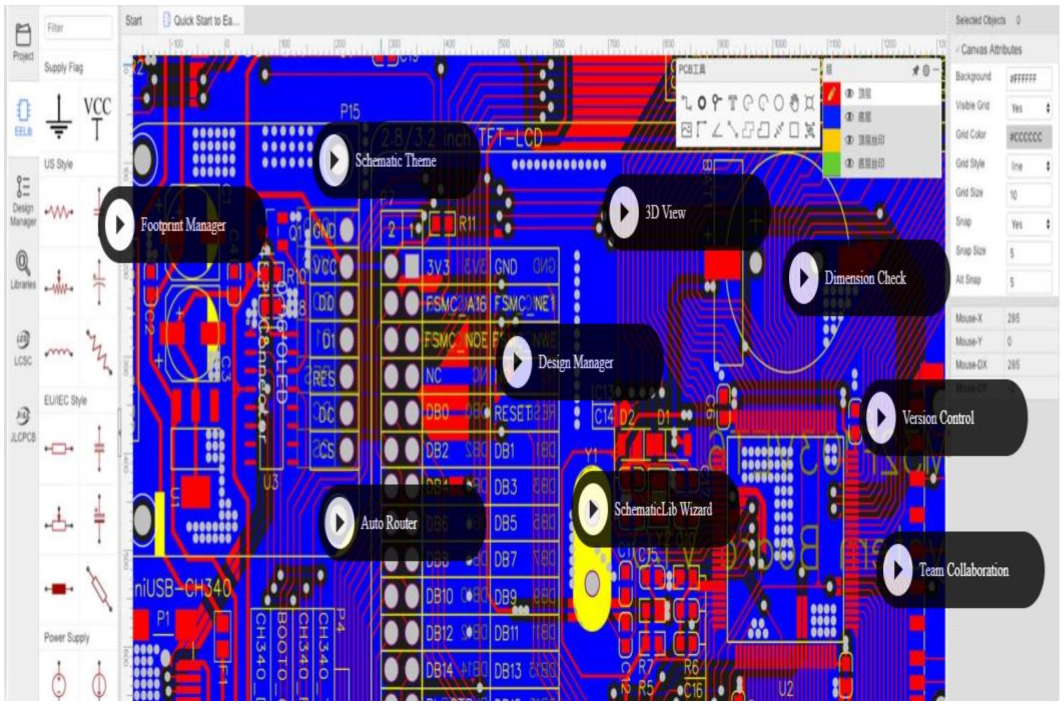
Our group believes the company that was best suited for producing our PCB was JLCPCB. The reason why we believed JLCPCB was the best company was for a variety of reasons. First of which they had quite competitive prices. The second reason was if they had quite an extensive list of components that would made designing our personal circuit board quite convenient. And lastly, the expected ship dates for a personal circuit board aren't excessively long. Our group also understands that in the day and age of COVID-19 that manufacturing dates and ship shipping dates may be delayed.

5.7 Manufacturing Design

For our manufacturing, we had needed for our project that needed were met for us to choose a certain company over any other. After looking through the companies that we went over in our PCB section we came to realize that our project had certain requirements that other engineering projects wouldn't had. Such as limiting a budget that's smaller than even some of the smaller engineering projects that some companies do. Another was our time frame. This was more similar to other professional projects but was still tighter than usual, where we had much less time for our parts and needed materials to came in, so we started testing and debugging. We needed a company that could handle our timeframe while keeping to our budget.

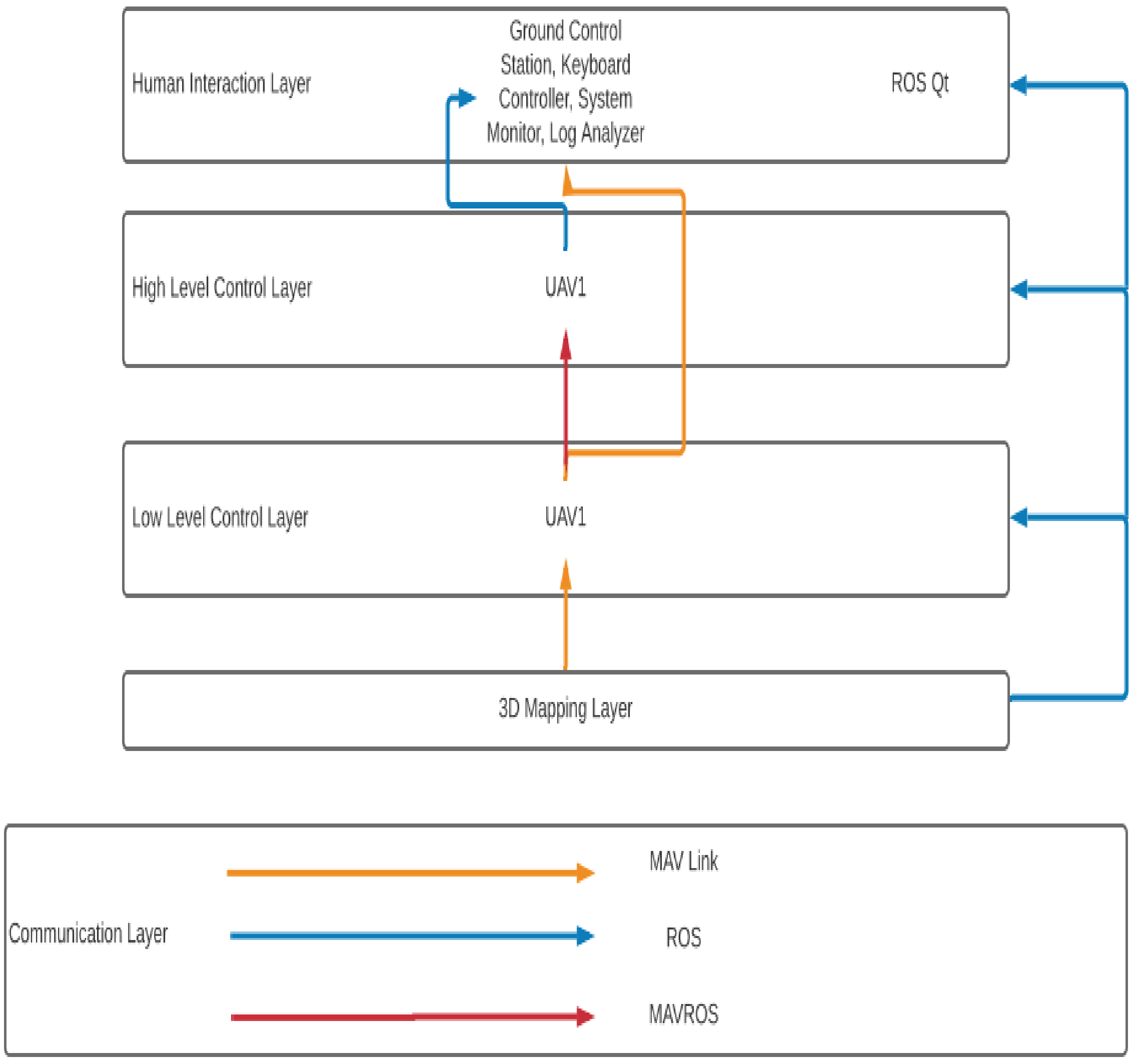
We talked earlier on how our company we were more favoring was JBCPCB, and that because of the short shipping times as well as the ability of their ordering program being well reinforced by their own design program called EasyEDA as shown in **Figure 40** that saves on time because when we design our PCB the program would cross-reference parts and made sure we don't order parts that aren't actually available, which was a very annoying problem many engineers face. Luckily our PCB was a very simple concept for our needed as power distribution boards were simple in design as they only needed to route power to certain areas and may be lower or increase the voltage as needed. We do this with capacitors and resistors and the simplest parts, so our PCB was a very simple design easily implemented. Our board was most likely the standard 1.6-inch-thick board and had copper power lines interlaced with voltage regulators to kept voltages where they needed were for certain parts and kept the design within scope.

Figure 40: EasyEDA Program for JLCPCB Manufacturing



5.8 Software Flowchart

Figure 41: Software Flowchart



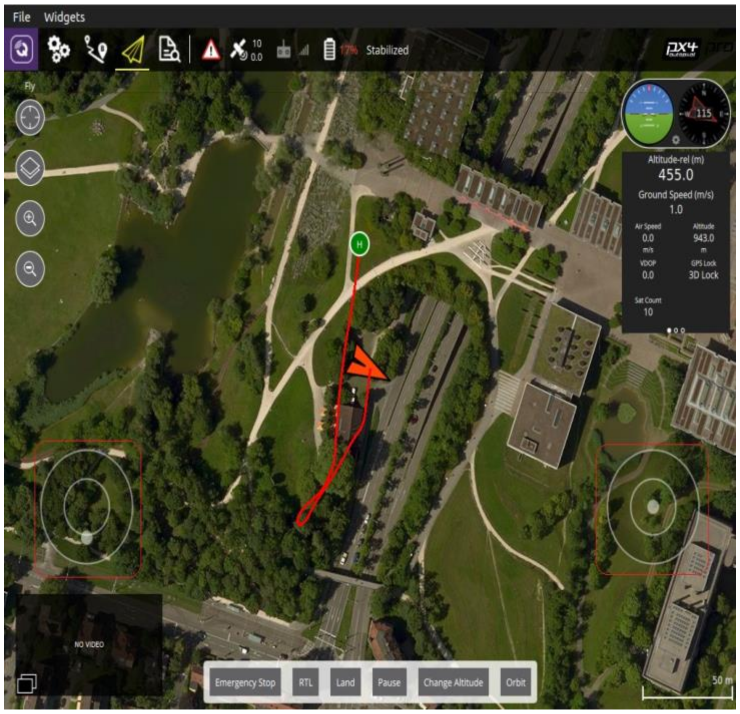
Our entire software stack was built on top of the MAVLink, ROS, and MAVROS communication protocols, with each one being used for different areas of the communication, shown in **Figure 41**. Our low-level control layer was running PX4, which takes in information about the drone such as orientation, rotation, and forces on the drone, and used to keep the drone in stable flight. Additionally, the low-level control layer was in charge of taking in GPS data and waypoint mission information and calculating the adjustments to motor speed in order to navigate to the correct location. From there, the next level of abstraction was the high-level control layer. This was a ROS node that basically acts as a router, taking in information from the flight control software, the 3D Mapping Layer, and the Human Interaction Layer, and then routing it to the correct destination. The low-level control layer sends telemetry data to the high-level layer. The Human Interaction

Layer sends user commands and mission data to the ROS node. Lastly, the 3D Mapping Layer sends the point-cloud information to the ROS layer. From there the ROS layer routes it to the correct software that was using that information.

5.9 Manual Pilot Controls

Since our drone was intended were fully autonomous, we would not had very much of a needed for manual controls. However, during testing, it may be useful were able to pilot the drone for certain things like positioning at the beginning of a test and returning home after a test. With such minimal usefulness, we do not feel that it warrants spending the extra money on a transmitter and receiver. Fortunately, our ground control station implements a virtual controller, so we would not miss out on any functionality. We had two options for this. The first was using the virtual joysticks as seen in **Figure 42.** This would require a touchscreen on our ground control station software.

Figure 42: QGroundControl Virtual Joysticks



Luckily, QGroundControl supports iOS and Android, so we were able to use any of our phones. However, by doing this, we were giving up useful features such as telemetry logging. We do had access to a laptop with a touchscreen, however. The other option was for users to plug in a USB Joystick or Gamepad. This could be achieved with hardware that many of us already had, such as an Xbox Controller. By using either of these two options, we were able to circumvent the needed to buy expensive hardware that had a limited effect on our end project. However, it was important to note that these controls were not as responsive as using an actual radio transmitter. This was because the commands were sent over MAVLink, which shares the bandwidth with the rest of the commands and telemetry data. Additionally, MAVLink commands were processed by the flight control software, which takes processing time. On the other hand, the manual RC transmitter was routed directly to the motors, avoiding the processing delay, though it was small.

5.10 Final Parts Decisions

The final parts decision section of this paper well in detail, explains how we came to the final decision for the parts. The parts discussed in this section would ultimately be purchased for our project. We believed these parts would produce the highest performing project we were capable of producing. As well as detailed explanations as to why these were the best parts to achieve the best possible performance out of our drone delivery project. Some of these decisions came from the decision of other ones because the parts interact with each other for proper performance. This was further explained in the decision of the parts.

5.10.1 Propeller Decision

When it comes to choosing propellers, it was no small or easy task and was very calculated. Propellers were a critical component of the drone because they were directionally correlated with the amount of lift you could achieve with your drone. There were a lot of reasons why you might pick a certain propeller over another.

The first reason why we choose the propellers that we chose was due to the size of the frame. Once we had chosen the size of our frame, we knew we needed propellers that provided a large enough lift for not only our frame but also our package. Now with a larger frame, you were able to install larger propellers. If we had chosen a smaller frame and larger propellers the radius of the propellers would intersect, and it was impossible to get the drone off the ground. With the drone frame we had decided upon being 450 mm but for easier to understood metrics could be converted easily to inches with the end results being 17.7 inches. With a drone frame of 17.7 inches in length, we had some flexibility and how long the length of our propellers we wanted. The standard size of propellers for a frame of our size was somewhere in the range between 8 inches and 10 inches.

There were drawbacks to either having larger propellers or smaller propellers. With smaller propellers, we had greater rotations per minute than our larger propeller counterparts. Now there were quite a considerable number of pros for having larger propellers. The first pro of having larger propellers was that we were capable of generating a larger amount of lift. A potential con of having larger propellers was that you had lower rotations per minute because you were rotating a larger mass when you had larger propellers. Lastly, it was believed that larger propellers were more beneficial to a drone’s battery life than its smaller propellers counterparts. With all of this information, our group had decided that we would much rather have larger propellers than smaller propellers. The length of propellers that we believed was the best fit for our 17.7-inch drone frame was 10-inch propellers. 10-inch propellers would have a 5-inch radius leaving 7.7 inches in the center of the frame to build upon.

Once you had the size of your propellers decided there was a final decision to made. The next decision to made about propellers was what material you would like your propellers were made out of. The two most common materials propellers were made out of were plastic and carbon fiber. Carbon fiber was a denser material. With a denser material like carbon fiber, you would have less flex as the propellers rotate. Also, with carbon fiber being denser it added more weight to your overall weight of the drone. Plastic propellers on the other hand were lighter but they would flex a little more when rotating. When doing a price comparison of carbon fiber versus plastic propellers, the carbon fiber propellers would end up costing more. Our group had decided that plastic propellers was the best fit for our project due to the fact that they cost less while producing the same amount of left and generally speaking being lighter.

In the end, the result for a journey to find the best propeller for our drone delivery project. My group had decided that the best propellers for the best possible outcome was 10-inch plastic propellers. The 10-inch propellors would provide the necessary lift and stability for the drone. These factors were considered important to the project because of the environments the drone was flying through. These drones could even be flying next to each other and had the potential to knock into each other and cause fires, dropped packages, and broken drones. These 10-inch propellers would alleviate that risk and helped it fly correctly. Sizes differences in propellers could affect how hard the drone needed to give energy off of the batteries and going too big could be unnecessary weight and cost of materials. We were having different colored arms on the drone, red and white. The different colors would helped distinguish the set-up of propellers because certain ones needed to turn one way and the others would turn the other, so having the two different colors would distinguish which direction they were rotating. We eventually used the arms to distinguish which side of the drone would be the front.

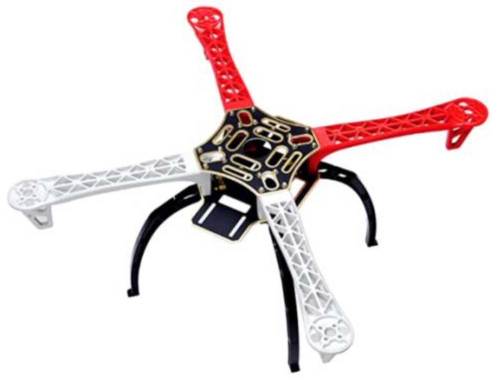
5.10.2 Frame Decision

We came to a decision about our frame through many looks into different aspects that would our continued efforts into the project, namely things like frame size for if the frame could even withstand the weight and flight style we were going to put it under, landing gear type because this style of the project at best was considered “pre-alpha” for many industry-standard professionals and we do not had a designated location with prime landing and took-off spots so we would needed a robust landing system to supply us with good potential took-off and landing success. Also, the frame itself had many holes integrated with the frame that let it be easily modified to fit our needed, for this we would most likely used them when having to input our sensor units and kept them stable and safe to our drone during flight, so we needed these holes and cabling to secure our vital components to the frame.

To continued, the frame also needed to compensate for its own weight that would take away from the lift, so anything the frame could got rid of and kept structural integrity was well-liked. Our drone does this by making the arms of the drone hollowed and instead of solid reinforced plastic, was honeycombed to kept structural integrity while reducing weight. We mainly investigated the frame size when looking for different drone frames when initially looking into what frame we would take. We did this by understanding that our drone would not be like other consumer drones that only had a certain weight they needed to lift and only had so many features. Ours was heavier than any consumer drone when we included all the other components like sensors, PDB’s, Batteries, and other components, but mostly things that would not be on a regular drone system like our second brain system. Thus, when looking into frames we decided the only possible frame that could held our system was 450mm and above, thus thinning the search for frames. And with all these qualifications, we came to the final decision of the QWinOut F450 drone frame with landing skid gear, as shown in **Figure 43**.

The frame chosen had a 4-legged landing gear that sits a few inches off the ground so the mainframe of the drone was not immediately coming in contact with the ground, which was useful for when there might be a puddle or something on the ground and protects the equipment from the elements. It also had two distinguishing colors of 2 red platform wings and 2 white platform wings which would help told which were the back ones or the side ones depending on how we decide for them. There was also a square-shaped middle part that extends down in order to store the equipment and made it compact. This was a pro when looking because it keeps the weight of the equipment in the center of the drone to tried and not throw off the center of gravity. Overall, this frame was very simple and looks like a great frame to started with as a beginner drone group builder. The frame chosen provided stability and a good layout of options to put the equipment being attached, the wings also had holes throughout so if needing to strap or hook it down to the frame helps for that security. The platform it comes with had a distinct area for the connectors of the motors, ESCs, and battery which helped with the organization of the components to secure them into place and not be crossing wires and getting confused with what connects where.

Figure 43: Drone Frame of Project



5.10.3 Motor Decision

When looking and deciding finally on what motor we used for our project, we were looking into many specifications that could affect performance, and the motor was one of the key components of this drone that cannot be sacrificed in terms of trying to find cheaper yet worse designs. We had to have a motor that was powerful enough to supply our needed lift requirements and also be able to fit on our frame. Luckily, most frames came with a motor specification attached to them so finding the right classification of the motor was as easy as reading which motor our frame needed. Taking this into consideration, to find our motor, was more of a challenge because different motors did the same thing but were less cost-effective because of outside factors that were out of strictly engineering outlooks such as shipping and other things. When we did eventually decide upon a motor it was not just because they met our specifications but because the materials came in a bundle deal to save on shipping cost. We bought a bundle that included not just the motor, but the frame, ESC, and all the plugs needed to operate these materials. We could saw the exact motor we bought here in **Figure 44**.

The motor that came with the frame kit was an A2212 1000KV Brushless Motor with a current capacity of 12A/the 60s, 2-3 Li-Poly cells, and KV of 1000. These specs correctly matched our frame requirements and what was needed to operate. It came with 4 orange and silver motors displaying the company name, QWinOut, and had the connectors with them. Having the motor be in a frame kit helped to alleviate the costs greatly because typically these were more commonly sold separately and quickly became expensive when having to purchase 4 of them. The frame kick comes with the 4 that were going were needed for the frames and propellers. Also, had the correct connections ready to hook up and organize correctly for the items to work together and started customizing how we went about communicating with the motors through the ESCs and software and such. We were using the max and mins of the motor to test accuracy and used them as boundaries for the code controlling the motor.

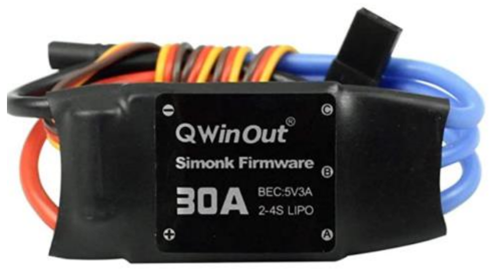
Figure 44: QWinOut A2212 Motor

5.10.4 ESC Decision

Our ESC, just like our frame and motor, came in a bundle to ease the expense, however, this was one of the hardest components that needed the most rigorous inspection to made sure we were providing our drone with an ESC that could handle the amp and voltage output needed to give our motors the power needed to achieve proper lift. The ESCs were the things that would directly be controlling the motors speed and RPM as the flight controller itself just sends a signal to these components and then they send the actually needed input voltage into the motors. To further protect our system, this ESC comes with its own software, taking away our needed to brought it up to operation and saves on precious time. It was protected from outside signals so that it couldn’t be interfered with by other RF sources, which was paramount in the urban environment that we envision our project taking place. **Figure 45** shows the ESC that we had decided upon.

Our ESC came with the frame kit ordered through eBay and does not need any soldering to put together. It even comes in a protective black casing and the wires attached to it were encased in protective covering. It was a 30A brushless ESC with a continuous current of 25A and a burst current of 40A for up to 10s. These were the exact specs we were looking for based on the frame and motor decision, which was great to saw it was included with the kit and it was going to fit the goals. This helps consolidate the worked needed to create ESCs. All the wires were different colors to distinguish what they were connecting to and the purpose. Without those colors, we would have to improvise and probably found a way to label them. This was used to control the motors and changed direction and much more. It was possible these might have had to be replaced if damaged, but luckily, they were not a huge expense and common to replace. And we did end up having to replace them after crashing. The ESCs also had all the information needed to found replacements for it with having the brand and specs of the ESC on the front of it. The biggest thing to note with the ESC was it was not necessary to went all out on the specs. The ESCs worked according to the motor and if an ESC could handle up to 50A, but the motor only goes up to a maximum of 20A then that was all that was necessary from the ESC. So, while it was important, we got good quality products, staying within budget was important and would save a lot of money in the end also.

Figure 45: 2-4S 30A ESC



5.10.6 Battery Decision

For our drone delivery project, it was critical that we used the appropriate battery for our design. The reasons why it was so important to obtain the correct battery for our drone delivery project were Weight, and the correct milliamp-hours (mAh) otherwise understood how long the battery could produce a current.

Now battery cannot be chosen at random they needed were very balanced towards your Objective. Due to the delicate nature in which batteries needed were balanced they often cannot be decided upon until other parts had been decided upon. So, to properly decide the correct battery you first needed to decide what motors and ECS you were implementing in your drone project. Since we had now made those decisions, we were able to narrow down our options for battery dramatically.

Now had been talked about previously the size and weight of a battery was very important. The size and weight of a battery for a drone delivery project were critical because if we were to hook a car battery up to our drone first would apply too much voltage and would have a weight in pounds so much that the drone wouldn't be useful. So, if a battery was too heavy it would create too much drag for the drone and would have diminishing benefits because as the drone struggles with drag it would use more electricity in the long run. That's why the weight and size of a battery were so important. The main way to gauge a battery's size was through the categorization of 1s-6s. With 1s being the smallest and 6s being the largest.

The most important feature and batteries were their milliamp hours. The reason why milliamp-hours were so important was due to the fact that more milliamp-hours were equal to more flight time. The longer the flight time the more you were able did with your drone in between charging the batteries. Now the amperage draw also plays an important part in flight time. The equation that shows their relationship was as given:

(Milliamp hours/ avg amp draw) *60= amount of flight time in minutes.

This equation played a critical role in why we choose the battery that we chose. For the amperage draw variable we mainly looked at our ECS's and What was the max amount of amperage they could took /handle and calculated a milliamp-hour range that we would need to use in the battery that we chose.

These were the factors that led us to choose the Turnigy nanotech 6000-mAh 3s Lipo pack as seen in **Figure 46** were the best option for were the drone delivery project. We believed the 3s were the perfect size range for what we needed with our battery and it not being too big or too small. Then we Calculated our milliamp-hour range were around 6000.

Figure 46: Final Battery Decision

With the Turnigy nanotech 6000-mAh 3s Lipo pack battery chosen. We Believe this battery would supply our drone with the perfect amount of average to ensure steady and efficient flight. This battery would also not be too big to way down the drone. With these factors maximized our drone had a much more sustainable flight time and was more effective during testing.

The battery was a rectangle shape that was enclosed in a blue rubber casing with the front having all the information of the battery. There was also a black, red, blue, and white cased wire that would helped distinguish which wire was being used. The battery provided the power needed for all the components and was rechargeable. We were going to have to place these in a way that does not affect the balance of the drone either because of its shape with the frame in order to keep the drone fly steadily and efficiently. We did this through many zip ties and using the frame as a stable connection.

5.10.7 Battery Leads Decision

When connecting our battery to our drone we had two methods to implement. The methods in which you could connect your battery to your drone were you either Sauter them together or you used connection leads. There were pros and cons for both, but my group had decided for a few reasons the battery leads was better than soldering together.

The reason why we believed connection leads was better was for multiple reasons. The first reason why we believed the battery leads was superior to the alternative of soldering was due to the fact that you could plug and unplug the battery from the rest of the drone. The ability to plug in and unplug the battery from the rest of the drone was superior to having it constantly soldered. With the ability to unplug the battery you were now certain that there’s no electricity flowing through any of the circuits or to the motors thus making it safer to handle. The safety benefit, as well as an easier-to-manage battery supply, was what led our group to decide to use battery leads instead of soldering.

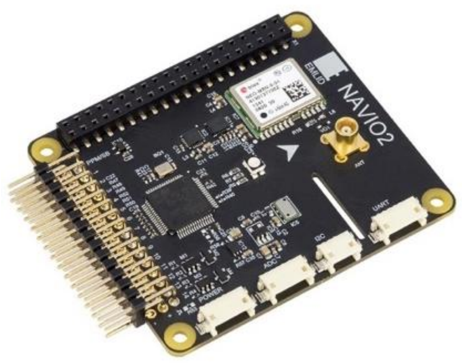
The connectors we went with were orange in color and had a hard casing for the structure. This would make it durable when we tried to use the batteries and possibly connecting and disconnecting them. Having that sturdy case would keep the battery wires protected from elements around it and even the case itself as it had space between where it was plugged in and the hard shell of the connector case. Having the connectors provided ease and convenience to the team when trying to construct what was basically a prototype drone. In the future, the decision to solder might be more appealing if it helps to secure the battery better and we knew that it was perfectly secured in that manner for production. Solder does help to eliminate weight as the casings and connectors added some fraction of weight on, but for this project there cannot be much harm did it and the for the right cost. Shown in **Figure 47**. We also used these same connections when designing the power module and our own power distribution board.

Figure 47: Final Leads Decision

5.10.8 Flight Controller Decision

The drone not only had mechanical hardware that it needed to operate correctly but needed proper processing power. We decide on a dual aspect of this component to save processing power and to give our drone the best chance of succeeding with all of the sensor inputs it would receive as well as when it needed to direct the ESCs to drive the drone. For the two-part system, only one of the boards was used as the designated “flight controller” and that was decided were the Navio 2. This board had all the simplistic design aspects that a young product could needed such as simple wire design and ease of input of pins. This board was made were in tandem used with the Raspberry Pi 4, however, that was used for a different aspect of control for the drone. The Navio itself was used to directly control the ESCs during flight and intake the sensor inputs from the more simplified sensors such as the IR and accelerometer sensors. As these would just be parameters that was read and interpreted were either in the allowable values or outside them. In **Figure 48** we see the board we were buying for our flight controller and the easy pin configurations that were shown on the board.

Figure 48: Flight Controller Board Navio 2



5.10.9 Data Processing Board Decision

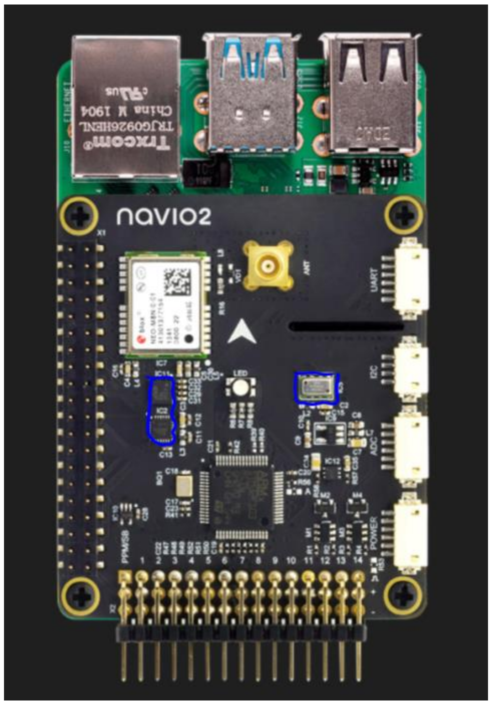
Figure 49: Data Processing Board Raspberry Pi 4b

As talked about in section 5.8.4, we acquired two distinct boards for our processing needed because of the intensity of our sensor inputs and decision-making needed. The flight controller was decided were the Navio 2. However, the Navio 2 was made were used in tandem with the Raspberry Pi 4. We decide on this configuration of Raspberry 4 so as to let the Raspberry Pi be used as our intense Data Processing Board. That was the designated title, however, what it really means was that our Raspberry was used strictly for the data input and used of our camera sensor unit and the used of robot vision that using this camera effectively would entitle. This was a very processing-intensive program so would needed almost the entire processing power of the Raspberry Pi which was why the Navio was used for everything else. We saw the Raspberry 4 and its ease of pin configuration and how it easily ties into the Navio 2 in **Figure 49**.

5.10.10 Sensor Unit Decision

The Sensor unit was actually many different sensors all packed together into one packet, but many were taken care of by being included in the processing boards we were using for our drone, namely the Navio 2 with its accelerometers, barometers, gyroscopes, and finally an included magnetometer which we could saw circled in blue in **Figure 50**. The sensors on the Navio were used to tell the drone how it was moving in space. The accelerometer, gyroscopes, and magnetometer collectively made up the Inertial Measurement Unit or IMU. This sensor unit gives the drone orientation, angular velocity, and specific force. Also, on the Navio there was a barometer that could detect a changed in air pressure, telling us the current elevation relative to the launch zone. However, all of these sensors only told us what the drone itself was doing in space. They told us nothing of the environment it was in, which was fundamentally important to autonomous navigation.

Figure 50*: Blue Circled Sensor Packages on Navio 2*



For this reason, the Navio sensors were not the only sensors we needed nor was using. On the Navio was many of the basic sensors that were included on flight controllers, but our needed went beyond the usual as we needed accident-avoidance aspects as well. We do this through the use of external sensors. For collision avoidance, we were using ultrasonic sensors facing in every direction around the drone. This provides us with data on the distance to the nearest obstacle in every direction. The way that these sensors provided the drone with distance information was shown in **Figure 51.** The transducer sends out an ultrasonic pulse. After some time, the receiver receives the reflected signal. The time of travel was measured to calculate the distance to the object. We would use this to ensure that we do not get too close to an obstacle. This still only supplies us with a relatively simple view of the world. For the actual navigation, we were using a SLAM algorithm to create a three-dimensional map of the world. Using a camera, we would record images of the environment and used computer vision to process them. This would provide us with a point cloud as seen in **Figure 52.** This was simply just a collection of points in three-dimensional space where an object was detected. With enough of them, the map starts to look like the real environment. This map would tell us what areas were safe to navigate through and what areas were not. While the 3D mapping tool didn’t come out perfectly, the object recognition worked just fine, but we later came to learn the ultrasonic aspect was not able to realize at the end because of limiting hardware. Namely the pins available.

Figure 51: Ultrasonic Rangefinder

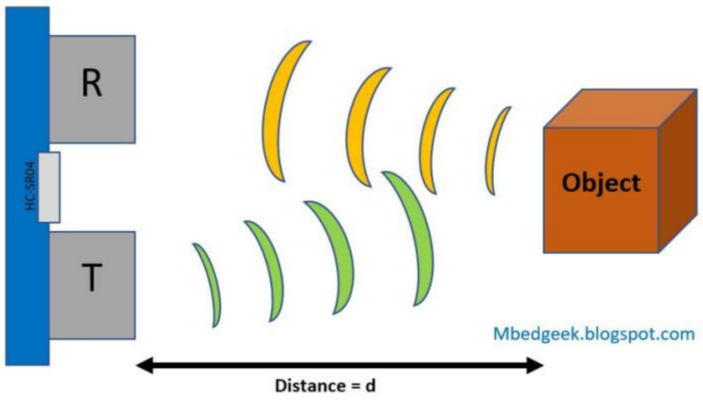
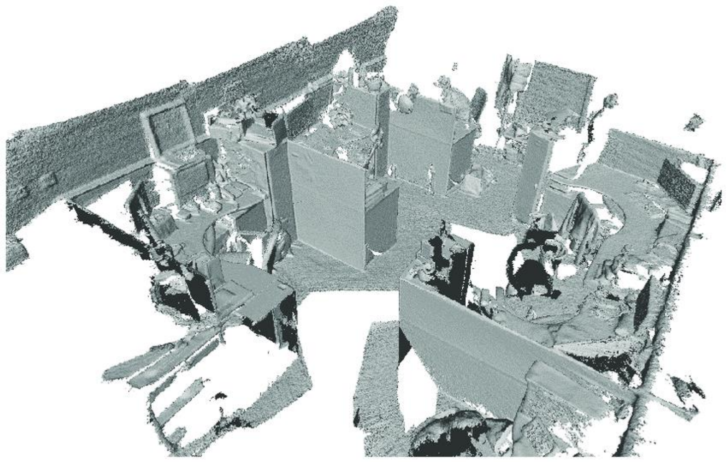


Figure 52*: Mapping Algorithm Output*



5.10.11 Coding Language Decision

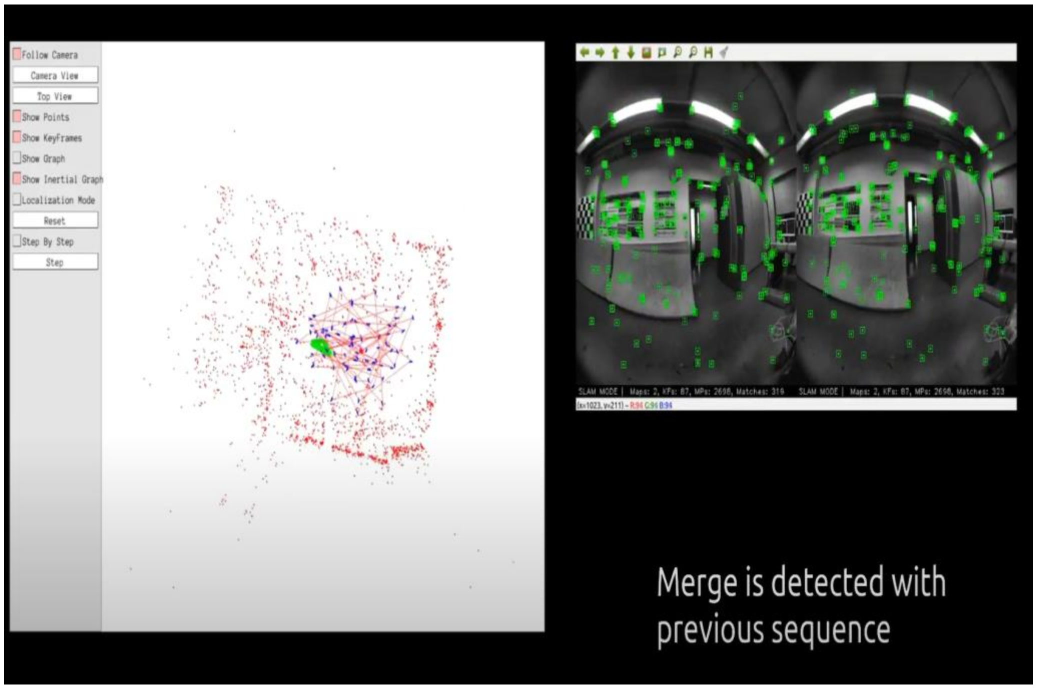
The coding languages being utilized for the drone was Python and C++. Python was utilized for the main software of the computer vision and controlling the drone. Python provides a style standard to followed and lots of documentation for completing projects of this magnitude. It was very popular in the machine learning and computer vision community for its feasibility of understanding and being extremely opened source.

C++ was utilized for a lot of the code between the hardware and firmware and lower language items. This would communicate how to set the hardware to certain numbers of performance. C++ was an algorithm-based language with object-orientated programming. This allows us to have a lot of flexibility in terms of things we could do with it because it would adapt to our software programs or with communicating to hardware.

5.10.12 Mapping Algorithm Decision

After carefully reviewing the pros and cons of each of the mapping software's in terms of features, speed, computational demand, and accuracy, we had ranked them from best to worst for our particular application. Unfortunately, because our computational resources were limited and it was impossible to estimate how much power we could afford to allocate for the mapping program, we cannot make a concrete decision on which one we were using. We were required to implement our setup and test it with various different mapping algorithms to saw what gives us a good balance of speed and accuracy/features. We eventually finalized on OpenCV as the tool used as the mapping tools we had were not working with our chosen camera at the time.

Figure 53: ORB-SLAM3 during operation

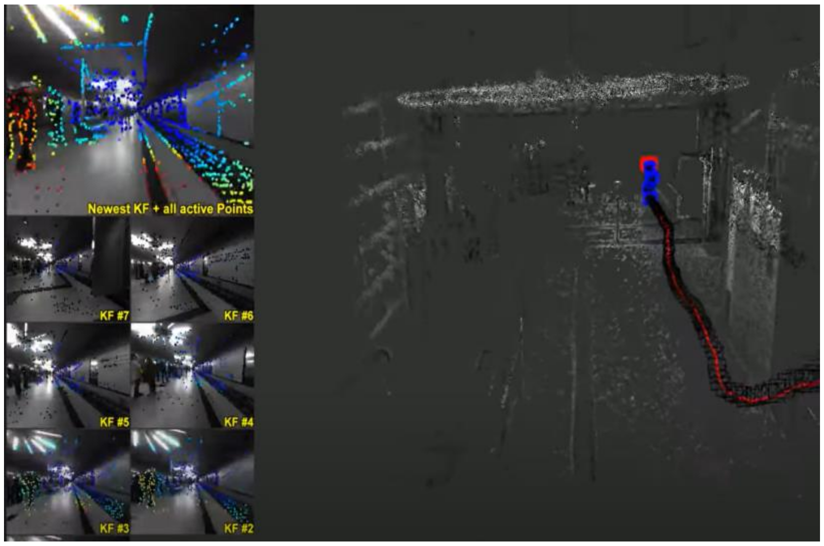


Our top choice for mapping algorithms was ORB-SLAM3. This application was the most feature-rich of the algorithms we looked at. It was a full SLAM implementation, which means it included features such as loop closing and key point reuse. These helped to ensure that our measurements do not drift over time, making our map more accurate. It also included multi-ran map reuse which would reduce the computational demand in high traffic areas such as around the loading warehouse, as the map had already been calculated for that area. This implementation was also more robust to periods of poor visual information. Whereas another algorithm could become lost if something like this happens, ORB-SLAM3 would allow us to pick up where it got lost and continued on with the mission. While these features were nice, they were not strictly necessary, and they do come at a cost. ORB-SLAM3 was significantly slower than some of our other options, which could limit the speed at which the drone could fly. For these reasons, we would attempt to used

ORB-SLAM3 as our mapping software. In the event that it does not ran fast enough on our setup, we would downgrade to the next best option. We saw ORB-SLAM3 in **Figure 53** in a working capacity.

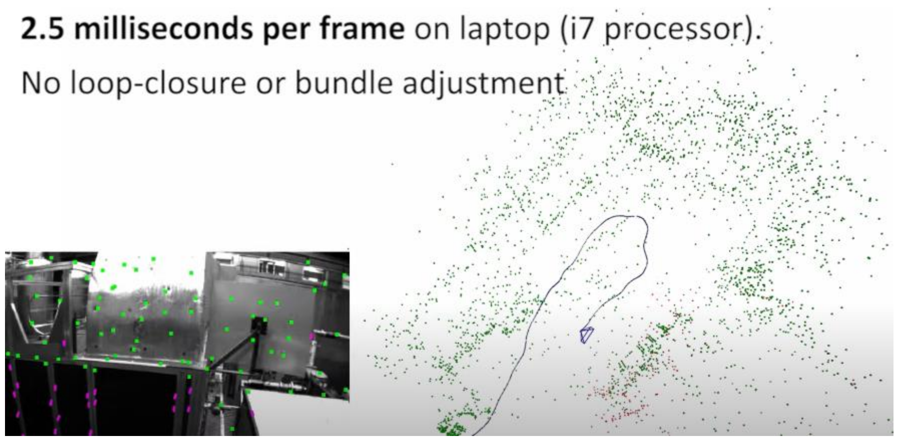
Our second-choice mapping algorithm was Direct Sparse Odometry. The base level of the algorithm was an implementation of Visual Odometry, which was not as fully featured as SLAM. It does not share some of the abilities such as loop closing and key point reuse. However, it was significantly faster than the ORB-SLAM3. Even though it was not a full SLAM algorithm, it was still more accurate and robust than the other options we looked at. The reason that this was our second choice was that it also comes with extensions such as IMU integration and loop closing. This allowed us to customize the feature set to achieve the perfect balance between efficacy and speed. If a certain feature does not appear were worth the performance decrease, we could simply take it out. This flexibility was very helpful in achieving a working final implementation. We saw in **Figure 54** Direct Sparse Odometry in a working capacity.

Figure 54*: Direct Sparse Odometry during operation*



Our last resort option was Semi-Direct Sparse Odometry. Again, this was an implementation of Visual Odometry, not SLAM. While it was not as robust or accurate as of the previously discussed options, it was significantly faster. It was an order of magnitude faster than ORB-SLAM and two to three times faster than the base implementation of DSO. This was our last resort because it was not as good as any of the other options, but it was faster. If we absolutely cannot run anything else, this algorithm was better than nothing. LSD-SLAM on the other hand was completely rejected as an option. It did not work as well as the other SLAM options in terms of accuracy or robustness, yet it was even slower than both of the visual odometry implementations. For this reason, this was the order that we would attempt to integrate the mapping algorithms. If a particular algorithm was too slow to run, then we would simply move on to the next best option. We saw in **Figure 55** the Semi-Direct Sparse Odometry in a working capacity.

Figure 55: Semi-Direct Sparse Odometry during operation


2.5 milliseconds per frame on laptop (i7 processor).
No loop-closure or bundle adjustment

5.10.13 Servo Motor

For our package delivery system to work smoothly we needed a servo motor to be implemented that would be able to release and lock in the package to the drone. The servo motor that we chose was the Tower Pro MG995 Digi hi-speed. The reason this server motor was chosen was due to the fact that it would come with a chassis that would allow us to easily secure the servo to the drone frame as well as the correct pin connections to easily attach to our flight controller. Our plan was to have a taut rope connect through the package and onto the pen of the servo motor so that the packages Center of gravity would be in the center of the drone. Well, having the other end of the tight rope be placed on the pin of the servo motor so when released the package would be carefully released and the drone would be able to fly away with the package no longer attached. The other advantage to this configuration is that the entire weight of the package would not be solely placed on the pin of the servo motor.

6. Testing Detail for Hardware and Software

Testing was imperative to product development because of how it prevents failures. Sometimes failures could cause a whole machine were destroyed but testing the software and hardware components could showed the flaws before interacting with the whole system. Section 6 goes into the details about how we tested the components for functionality and making sure the items were performing to our objectives and specifications before doing full launch tests. Then we were going over how the final tests was completed.

6.1 Hardware Testing

Our group believes in doing thorough hardware testing to ensure each part performs as it should. The reason we believed thorough hardware testing was necessary was for safety and to made sure we don't accidentally ruin any of our parts unnecessarily. Our group believes in measuring twice before testing once. What measuring twice before testing once entails was as shown below, through calibration testing, started-up sequencing testing, and field testing in multiple testing environments in between to make sure nothing was out of order or unsafe for us to continue before we tested our parts.

6.1.1 Calibration Testing

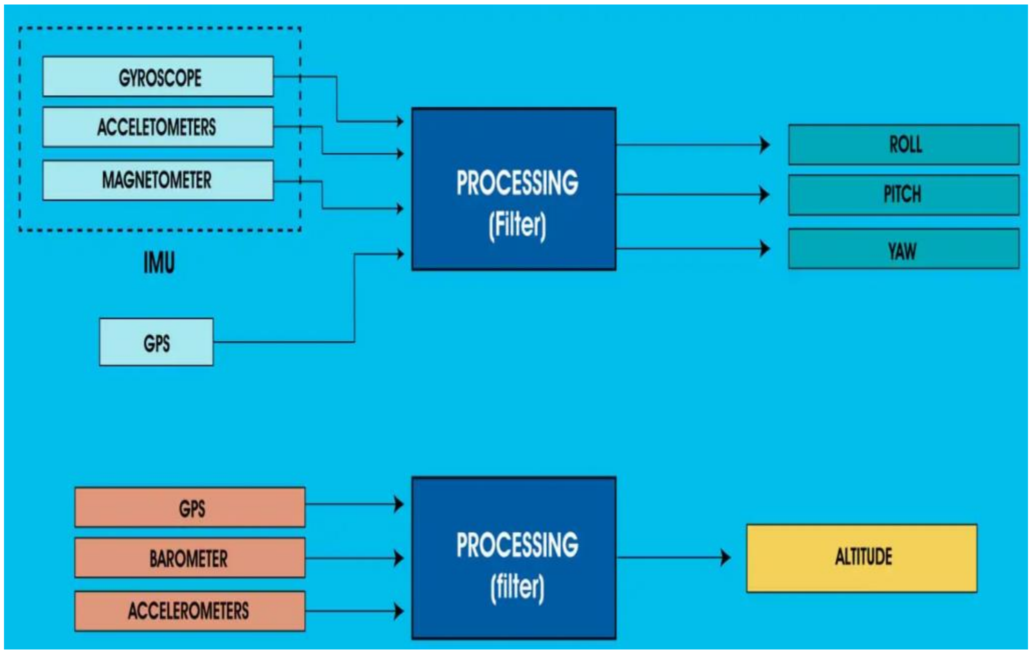
Calibration testing on a drone was critical. The reason in which calibration testing was critical for a drone because without the correct calibration your drone would not be able to level itself. As well as not being able to orientate itself in the correct direction if it was not calibrated correctly. From my hardware ‘s perspective, there were three points of failure to made sure were calibrated correctly. Those three points were the power distribution board, ECS, and flight controller.

For the power distribution board, it was important to calibrate correctly for a few reasons. First of which was that all the components on our drone were voltage sensitive. If our power distribution board did distribute the incorrect number of voltages to individual compartments they would not be powered correctly. As well as be at risk of overheating and frying themselves. For these reasons, it was important that our power distribution board be calibrated correctly. This was tested thoroughly through the use of multi meters and running current through the board.

The next piece of hardware to calibrate correctly were the ECS. The ECS Job was to take input from the flight controller and to distribute the correct amount of current to the Motors respectively so that the drone could orientate itself correctly. So, if the ECS were not calibrated correctly the drone was unable were balanced and hover as well as maneuver at all. To ensure that the ECS was calibrated accordingly we was sure to test them to made sure we were getting the same input out of all four ECS. This was tested during initial calibrations before mission take off.

The last piece of hardware was the flight controller. The flight controller’s calibration was critical. The flight controller uses a gyroscope to made sure that the drone was level. Not only that but the flight controller was also responsible for being the main form of communication to the ECS and the motors to made sure that the drone could maneuver properly. The flight controller was very thoroughly tested, as it needed many different parameter inputs before it could hope to take off for a mission.

Figure 56: Calibration Stylized Testing



For all the reasons listed above was why calibration was important in the way that we plan on testing it was through trial and error. As well as taking careful measurements before officially testing it to making sure that each part of our drone was calibrated correctly and doing its job properly before pressing the started button. We believed this method helped our drone overall be more calibrated and balanced and ensure quality testing.

We saw in **Figure 56** that calibration was needed for the system to understood certain variables and immensely helps other programs such as our flight controller, where if we didn't fully calibrate our sensors such as the gyroscope and the accelerometer and magnetometer, our flight controller would had much more difficulty flying using such parameters as roll pitch and yaw which could led to very harsh flying conditions and even to crashing if the parameters were too inaccurate.

6.1.2 Start Up Sequence Testing

Our group was a firm believer in measuring twice before testing once. What this entails for the startup sequencing, was that we plan on testing all the hardware individually as well as setting up to made sure all resistances, capacitances, and voltages were as they should be before turning it on. For example, when the ECS arrives we took a multimeter to all parts of the board to made sure we were getting the correct inputs and outputs throughout it before our formal testing started. This procedure would also be used for the battery, power distribution board, ECS, and Motor. This was done and worked to ensure power was going where it needed too.

There were a few reasons why we wanted were this thorough with designing our drone project. One of such reasons was that when we went step-by-step and check each part individually, we gain a better understanding of what that component was doing and processing and how it should function properly. With that information, if an error or a mishap were to happen, we would then be able to diagnose the problem sooner and more accurately.

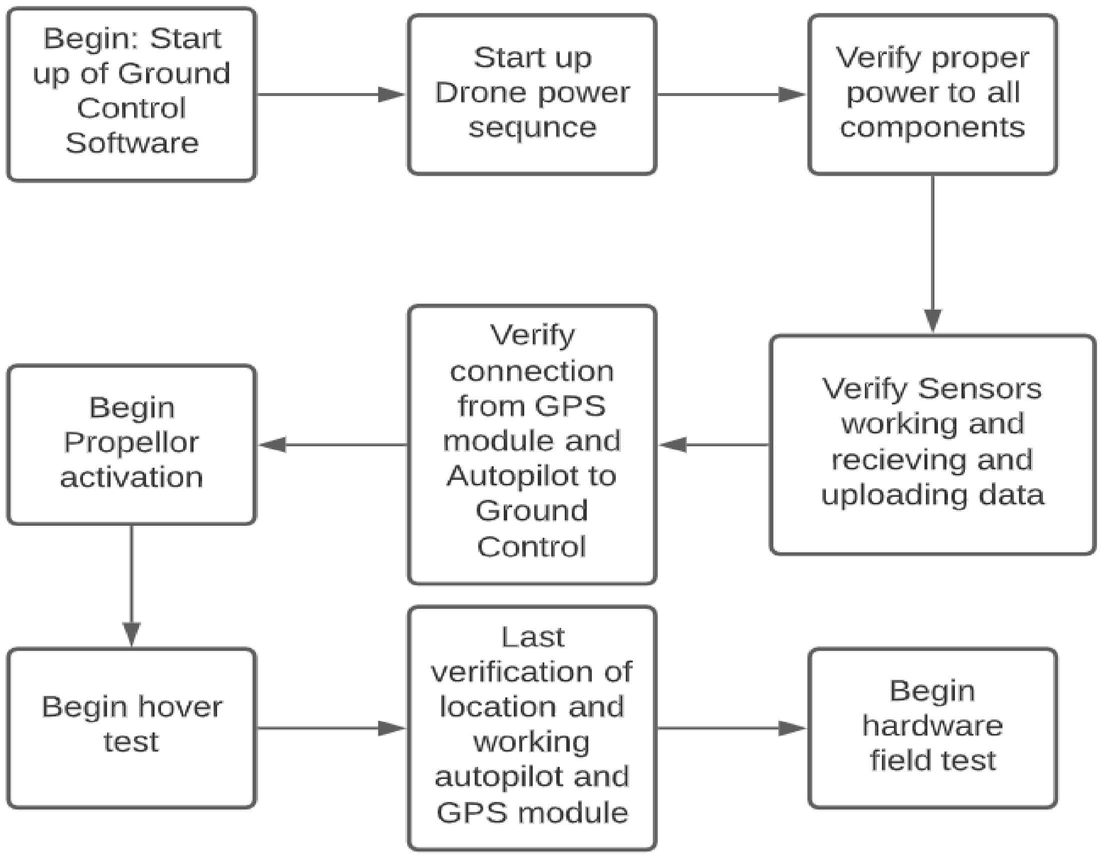
Once each part of our hardware had been tested to made sure was operating properly, we would then hook them up together in the order in which they were intended. Then test again to made sure we were getting voltages, resistance, and capacitance through each step of the drone and that everything was running smoothly as it should. Then capacitance the results to made sure they line up with our original result. After this procedure had been completed and we had tested the hardware twice. Once individually and once as a collective we believed it was safe to turned on the drone.

With these extra steps in place, the first time we set up the drone we had a successful start. On the off chance we did not have a successful startup with all of our testing beforehand, which we did not in hindsight, we hope to now have a better idea of where we would go for the next step when continuing to make our drone operational and useful.

We saw in **Figure 57** that our flowchart shows the intended sequence we were following for further testing when performing field tests and we was applying this style of testing for Senior Design 2 where we would have a working drone and sensor configuration. This would enable us to have a stable and routine started-up sequence while applying safety to all field tests so as not to damage the drone and hardware components on the drone itself, while also verifying that all data inputs were actually receiving data from surroundings and not facilitating a failed and useless testing environment. We could saw from our flowchart that our started-up sequence would go in the order of Ground Control activation, namely our QGroundControl Program that was facilitating our “talking” to the drone and all communication for route control and verification of completion of objectives such as testing or dropping off a package.

Moving on, we had the activation of the drone’s power system, or in other words the activation of the hardware of our project, with a further step of verifying that all components of our drone were receiving power and there were no broken or damaged lines in our power distribution system. This was needed for any further testing or processes in our started-up sequence because if even one of our sensors isn’t working it invalidates all the testing that could be done because of loss of data streams that could enable and proper test.

Figure 57: Start Up Sequence Flowchart


Begin: Start up of Ground Control Software
Start up Drone power sequnce
Verify proper power to all components
Verify Sensors working and recieving and uploading data
Verify connection from GPS module and Autopilot to Ground Control
Begin Propellor activation
Begin hover test
Last verification of location and working autopilot and GPS module
Begin hardware field test

After we had verified all components were working and receiving power, we verify all sensors were actually attuned and ready for testing, such as the ultra-sonic sensor was reading distances properly and the camera vision was properly on and detecting certain images, we would prepare such as a picture of a raccoon other objects to verify that it was reacting and reading properly.

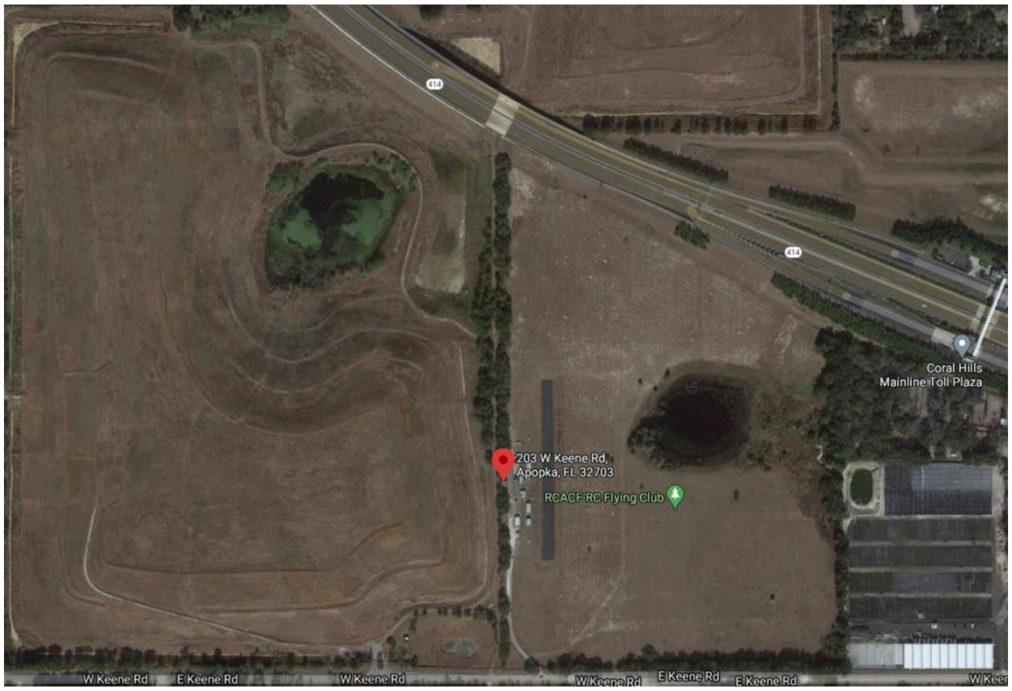
To continue with the started-up, we had the verification of the GPS system letting the drone and us operators wherein the location it was currently and verifying this was right and accurate to the level the GPS module could achieve. This helped and began the verification of the autopilot system. Showing that our autopilot had properly uploaded the correct route to the location we had routed for it, and it reads the correct location via coordinates for the drop-off point.

Finally, we began the final processes that could be done before a proper test could be fulfilled. We started up the propellors and verify all systems were still working and GPS was still receiving data and sending data even during the vibrations of the propellors were causing on the frame. We then achieve lift-off and enter our true testing phase. We verify the third and last time that the GPS and autopilot systems were interacting properly with the Ground Control system and that all sensors were working properly and finally began a true test of our drone accident-avoidance system.

6.1.3 Field Testing

During our testing, we would eventually reach the stage of quality in data that the only way to extrapolate more was did a field test and brought our drone to operate in a real environment, or as much as we could made our drone were in while keeping to the budget and constraints of the law. For our project, we came to the idea of showcasing our drone's ability to dodge buildings and other environmental stimuli and showcase our accident-avoidance software and how well it works when put under pressure while in tandem with how well our drone's sensor package could pull in data and compare parameters in real-time.

Figure 58: Field Testing Site Tangerine Field



Now to showcase these two things we needed for there were an adequate location, which in this case means that it needed a wide grass or opened terrain and no other major outcroppings hanging over it and were outside of downtown and campus limits as those places require that we would needed to acquire a drone pilot license for our operation, which if it could be avoided would reduce down the cost of our project and made it more viable. And we needed adequate stimuli so as to fully showcase and extrapolate a good amount of data from the drone's interaction with the course. We were then deciding did our field test in a wide-opened field, preferably a grass plot, so that means our best guess were a park or other opened space such as that, and we found through research into where other drone pilots that fly in the city went to that there was a perfect location in Apopka Florida known as Tangerine Field. It was opened land with plots of grassland and even a runway, cover for when thunderstorms hint so we could be in accordance with safety guidelines when operating in the opened air and was close enough to our home locations were viable for this project needed. We saw this in **Figure 58** that showcases the field as a viable option for testing.

Figure 59: PVC framework of Buildings for Field Test

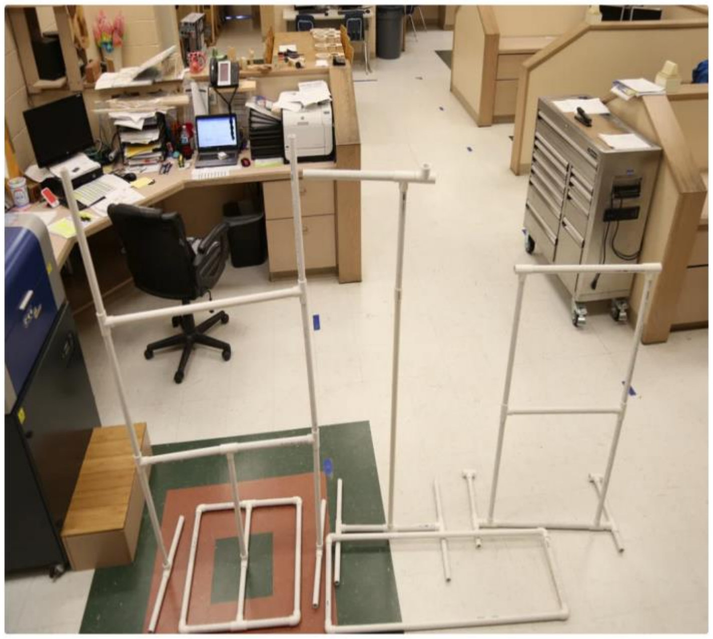


Figure 60: Box that would form the Skin of City

This means we now had an adequate location for testing, now we needed adequate stimuli for our drone and data gathering needed. We do this by recreating the most likely environment our drone was encountering during operation as a product in consumer's hands, or in other words, an urban environment. Now because of budget and site restrictions, this means in no way were we able to emulate an actual cityscape. We were recreating the tight spaces and restricted airways that came with cities instead. To do this we would made the framework of a building or in this case multiple buildings out of PVC pipes so as to give our site some sturdiness in case of high winds and other factors of the flight-testing site. We would then provide the “skin” of the buildings with cardboard to showcase the more rectangular aspect of downtown buildings, with variations in the different structures to provide a good difference in data input so that any field testing brings back multiple stimuli injection into our data network for our drones “learning” software system. We gave the basics of this idea in **Figure 59** and **Figure 60** to show the PVC framework and cardboard skin of our made-up cityscape. This ended up not being used even when we had the materials ready for the build as a crash stopped any testing we could continue do before our final paper was due.

6.2 Software Testing

For testing the software there was tools were able to measure output and then also testing scripts used to simulate the runs and also collect data. With drones, there was simulation software out there that was able to practice machine learning and object detection without having to use the drone yet. One simulation software we were utilizing was Gazebo. Gazebo had been highly recommended by PX4, which was one of the design choices and able to set up connections and simulate because PX4 uses SITL. SITL stands for Software In The Loop. This means that was able to run the software flight stack and used a simulator like Gazebo to created values and saw how the data was outputted. In order did these types of simulations, a MAVLink API needed were set up. **Figure 61** was an overview of how the data and telemetry came out of the simulation. The API allows sensor and message data to went through and then output telemetry based on our needed.

Figure 61: MAVLink API

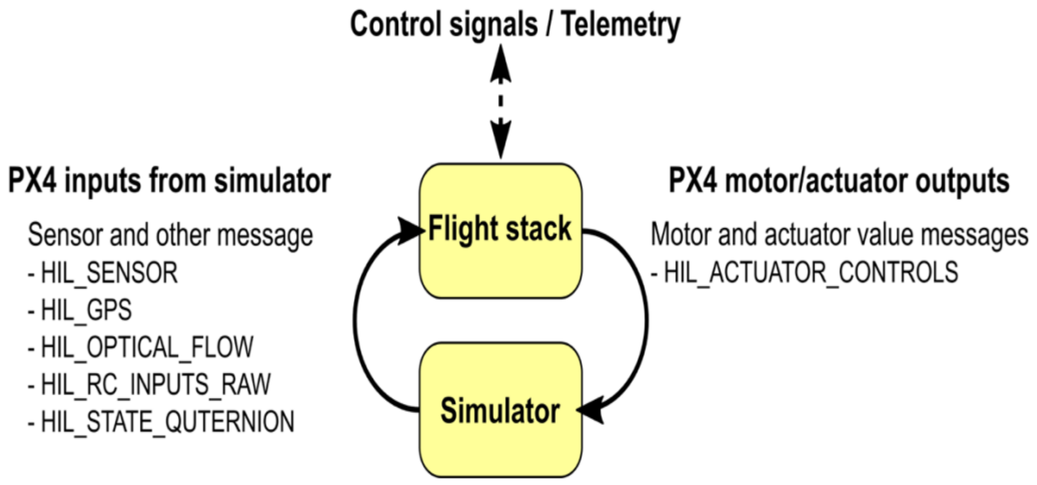
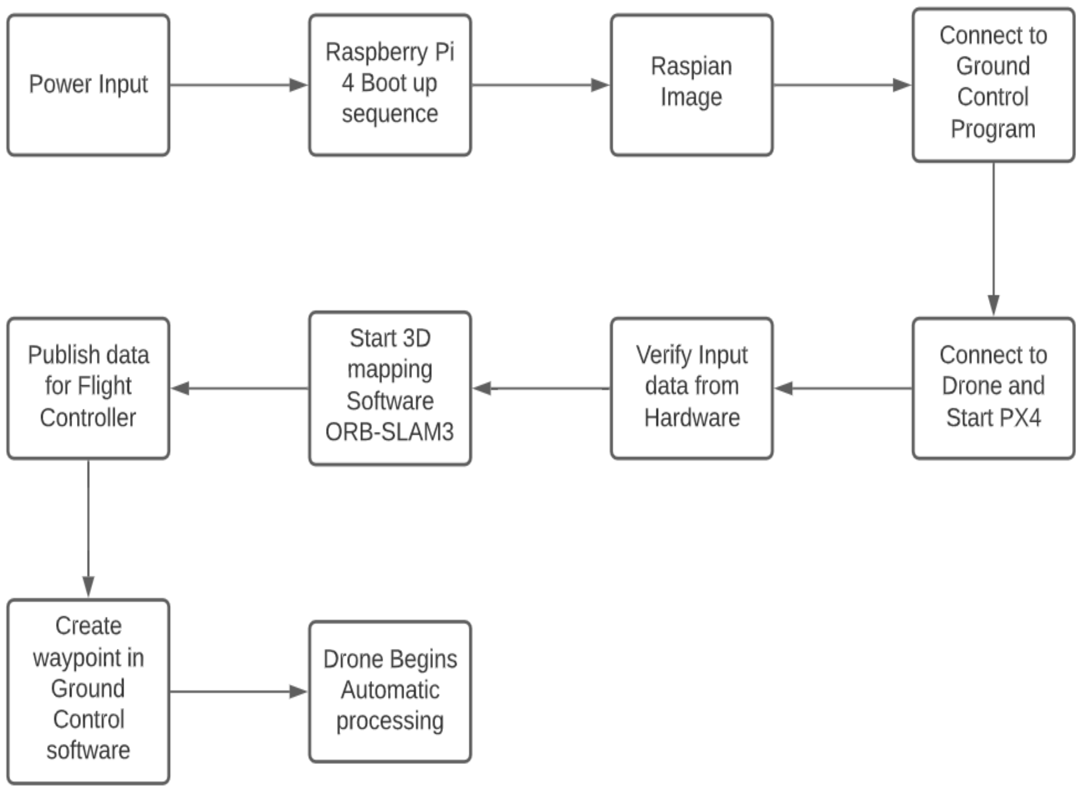


Figure 61 outlines the messages that was used to access certain parts of the drone connections. HIL_SENSOR was used for IMU readings, HIL_GPS was used to access GPS RAW values. These could be specified and used for debugging because the HIL_STATE_QUTERNION gives telemetry of the altitude, speed, locations and so on which could be analyzed more with what needed to happen and what happened. We were able to plug up our flight controller board of the Navio2 setup and then test it with our software to saw if it was working properly. It also allowed us to ensure that all of our individual software were working independently, and also that they were communicating effectively. Testing was extremely important to the design process of projects because, with projects just as drones and material items, hundreds of thousands of dollars could be put in these things, and if not tested because a demo could result in damage to the environment and the item at hand. All that time and money spent was destroyed and had to started over again. Having software like Gazebo also allows us were making progress on the software part of the project without having to worked with all the hardware parts or had the drone flying yet.

6.2.1 Software Startup Sequence

Figure 62: Software Start Up Sequence

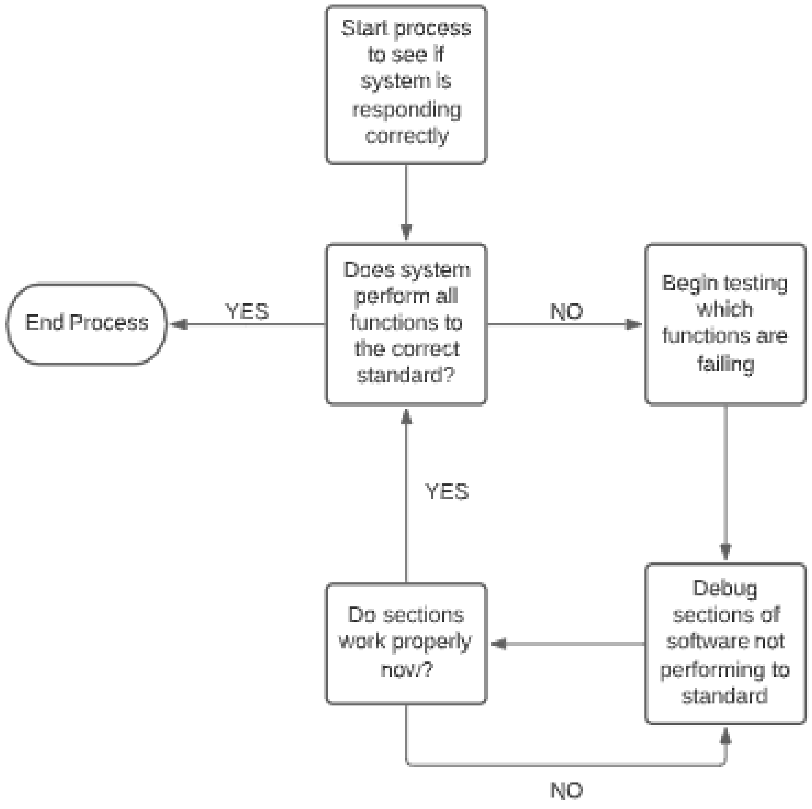

Power Input
Raspberry Pi 4 Boot up sequence
Raspian Image
Connect to Ground Control Program
Connect to Drone and Start PX4
Verify Input data from Hardware
Start 3D mapping Software ORB-SLAM3
Publish data for Flight Controller
Create waypoint in Ground Control software
Drone Begins Automatic processing

When the battery was connected and the drone receives power, the Raspberry Pi would automatically boot the installed Raspbian image. After the board was fully booted, it would connect to our ground control station laptop via a Wi-Fi hotspot. Once connected, we were able to SSH into the drone. We used this to started PX4 on the drone. When this software was up and running, we would perform a pre-flight check to ensure all of the hardware was working correctly and all of the telemetry data was being passed to our ground control station, we started running the 3D mapping algorithm. This immediately started publishing the point-cloud data for the pathfinding algorithm to used. From there we created a waypoint mission and set the drone into launch mode. The rest of the behavior of the drone was handled fully autonomously. We saw this process sectioned out in **Figure 62**.

6.2.2 Software Failure Testing

We needed to prepare for all possible outcomes during the intense operation of our drone's standard operating procedures. We had fail-safes and guidelines that we followed in case of failures that could cause catastrophic events to the drone. This section was going over the software style of failures and how we were approaching them. We saw this in **Figure 63** on how we took such action in the future.

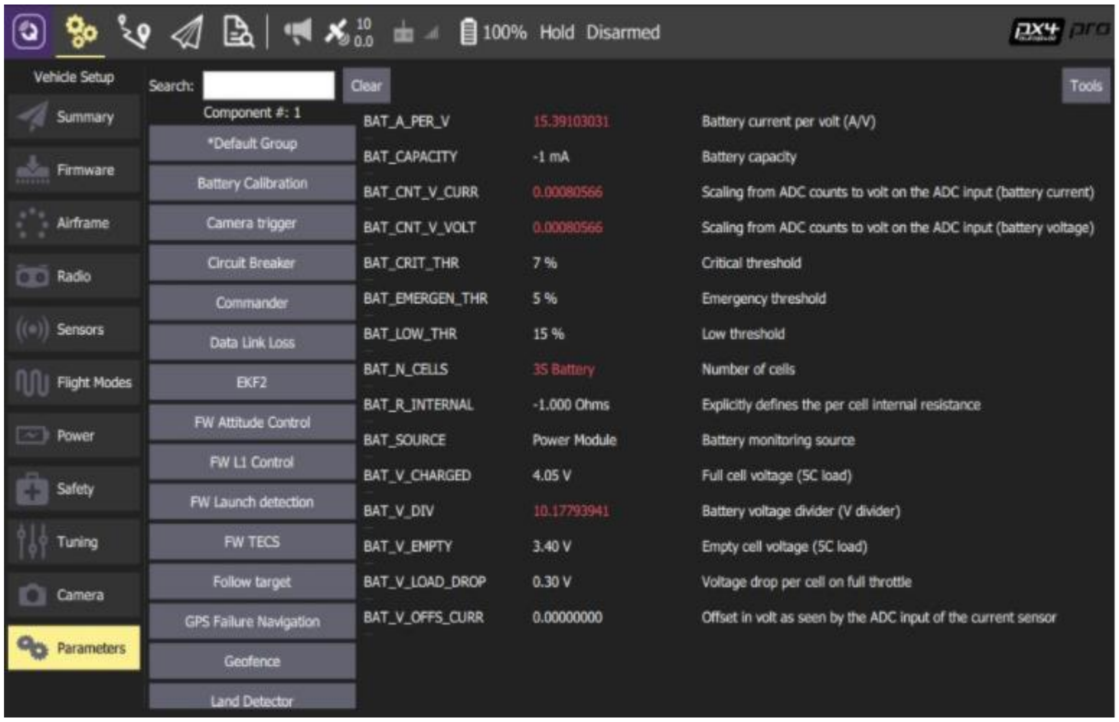
Figure 63: Software Fail Procedures


Start process to see if system is responding correctly
Does system perform all functions to the correct standard?
End Process
YES
NO
Begin testing which functions are failing
YES
Do sections work properly now?
Debug sections of software not performing to standard
NO

6.2.3 Parameter Testing

Our parameters were paramount to our successful implementation of an accident-avoidance system. These needed systems were fine-tuned to have any chance of a successful dodge. We were using a ground control program known as QGroundControl that had an easy-to-understood user interface. This program lets us control the parameters of our flight controller so as to provide a tighter detection range for our sensor. When entering the GUI for QGroundControl under the Parameters tap we saw we changed all kinds of systems within our flight controller's umbrella of control, as shown in **Figure 64**. Just one of these were the systems usual inputs for the return home command. We changed that to something more easily useful for us during testing.

Figure 64: Parameter Screen on QGroundControl

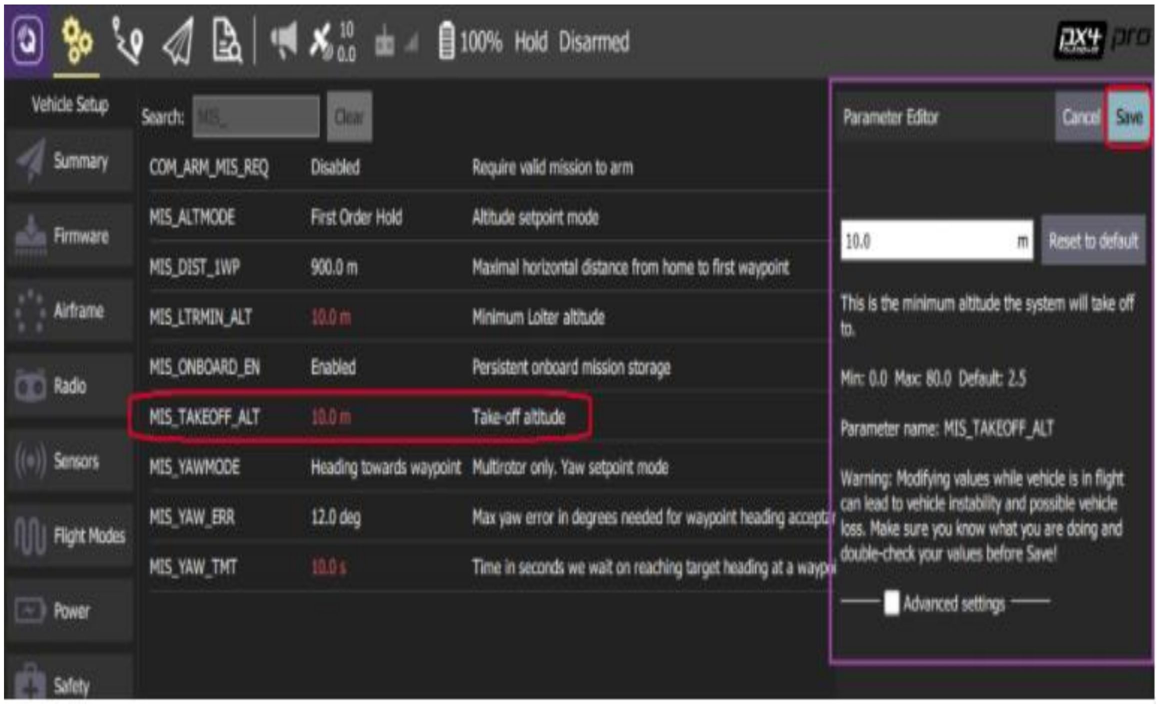


BAT_A_PER_V	15.39103031	Battery current per volt (A/V)
BAT_CAPACITY	-1 mA	Battery capacity
BAT_CNT_V_CURR	0.00080566	Scaling from ADC counts to volt on the ADC input (battery current)
BAT_CNT_V_VOLT	0.00080566	Scaling from ADC counts to volt on the ADC input (battery voltage)
BAT_CRIT_THR	7 %	Critical threshold
BAT_EMERGEN_THR	5 %	Emergency threshold
BAT_LOW_THR	15 %	Low threshold
BAT_N_CELLS	3S Battery	Number of cells
BAT_R_INTERNAL	-1.000 Ohms	Explicitly defines the per cell internal resistance
BAT_SOURCE	Power Module	Battery monitoring source
BAT_V_CHARGED	4.05 V	Full cell voltage (5C load)
BAT_V_DIV	10.17793941	Battery voltage divider (V divider)
BAT_V_EMPTY	3.40 V	Empty cell voltage (5C load)
BAT_V_LOAD_DROP	0.30 V	Voltage drop per cell on full throttle
BAT_V_OFFS_CURR	0.00000000	Offset in volt as seen by the ADC input of the current sensor

We could further search specific parameters we needed to truly implement our system using a given search bar, which was useful in our case because of the drastic number of sensors we were inputting into our drone's flight controller and data processing board. There could be instances of names being not perfectly cloned over to the GUI of QGroundControl so the ability to looked up specific parameters was a useful tool in our project. To continued further with actually changing parameters, this GUI had a helpful system for the user when one needed were changed; namely when clicking and changing a parameter a sidebar menu would populate with the already given parameters and what they actually meant and how changing certain ones changed the outcome of the system in total, as seen in **Figure 65**. This figure demonstrates what it looked like when trying to operate the parameters and search function making it a breeze to used.

This would ease our inputs so we could slowly and steadily raise and lower our parameters where they were needed so as to provide our accident-avoidance system with the adequate data inputs it needed to perform to the best of its abilities. We were able to implement test cases that changed parameters rapidly or drastically in order to test out some of the extreme cases and saw how the drone would react to them. It was important to utilize these sources instead of using real objects that in the case of a test failure would made the drone crash and became damaged.

Figure 65: GUI Parameter Help Interface



COM_ARM_MIS_REQ	Disabled	Require valid mission to arm
MIS_ALTMODE	First Order Hold	Altitude setpoint mode
MIS_DIST_1WP	900.0 m	Maximal horizontal distance from home to first waypoint
MIS_LTRMIN_ALT	10.0 m	Minimum Loiter altitude
MIS_ONBOARD_EN	Enabled	Persistent onboard mission storage
MIS_TAKEOFF_ALT	10.0 m	Take-off altitude
MIS_YAWMODE	Heading towards waypoint	Multirotor only. Yaw setpoint mode
MIS_YAW_ERR	12.0 deg	Max yaw error in degrees needed for waypoint heading acceptar
MIS_YAW_TMT	10.0 s	Time in seconds we wait on reaching target heading at a waypoi

10.0 m

This is the minimum altitude the system will take off to.

Min: 0.0 Max: 80.0 Default: 2.5

Parameter name: MIS_TAKEOFF_ALT

Warning: Modifying values while vehicle is in flight can lead to vehicle instability and possible vehicle loss. Make sure you know what you are doing and double-check your values before Save!

6.2.4 Sensor Input/Output Testing

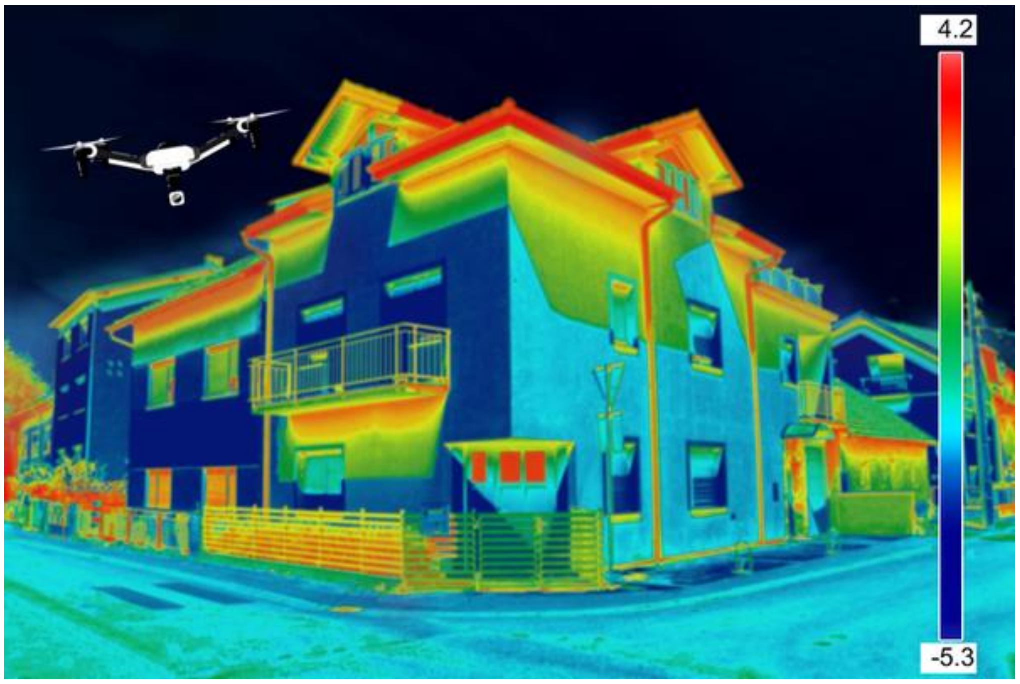
Sensor testing would require using different environments of "test cases" that they could came in contact with. Some sensors needed were tested in different lighting conditions, while others were speed and height changes to saw what data it was giving us.

To started off with this system we used simulation software that could connect to the sensors and gave it certain conditions to test the functionality of the sensors. Next, we would attach to the drone and do some test flying to saw what data it was giving us. For example, the accelerometer was attached to the drone while we test going in straight lines and having a speed detector on us to compare that to the data the sensor gives. So, if we saw the speed detector showing us 10 mph and the sensor showing 60 then we needed to work on calibration or saw if the sensor was faulty.

The barometer was used to helped test the position of the drone in addition to the GPS. We were testing these items to made sure they could give data back to a device that would translate it into coordinates. We then had these on the drone and place them on different objects with different heights and test what coordinates it gave. Testing on steady objects first was the very important starting point for these sensors because we were able to hand measure it ourselves, whereas once it starts flying was more complicated to accurately measure and needed to have confidence in the sensors that they could complete the tasks and checks.

Ultrasonic sensor testing would consist of having it on a steady object or table in a clean room and putting objects in front of it at different lengths. With everything standing still, we could measure out how far to put the objects to test the max detection rate and size of objects, if it makes it easier or harder. Then based on that feedback it would upgrade to testing it while hovering and slightly moving. The goal of testing the ultrasonic sensor was to gain confidence that while it was moving at a set speed it was able to detect the object in front of it, otherwise, this would result in a crash of the drone.

Figure 66: Heat Signatures with Numbered Parameters

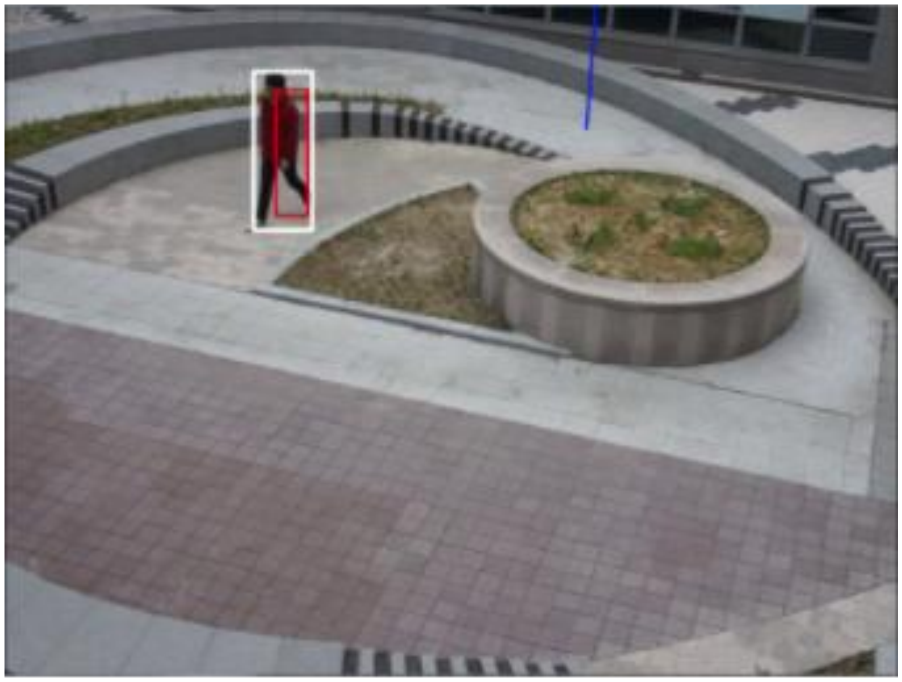


Our IR sensors were very instrumental in anything we would had to avoid that holds such a heat signature that it was hotter than the surrounding area, and to the surprise of most this included many different things that could affect our drone, not just alive things that were moving in tandem with our drone and could potentially hit it such as a bird but even air gust and spots of rising heated air that could destabilize our drone in flight if the drone ran into such a thing unexpectedly. Now IR was not the best sensor for daytime as this was wildly known as only the best IR sensors were sensitive enough to distinguish things from the UV heat of the sun during the day, but for nighttime flight, we could be assured our drone was very good at it. For as this could opened the market for 24-hour operations and helped to deliver companies everywhere with this specific problem. Now for our specific testing parameters, we would "teach" our drones flight controller system when to distinguish certain heat signatures from others and teach it what to classify as background versus what not to with parameters we would implement and changed within our ground control program and fine-tune during actual testing, using numbers such as shown in **Figure 66**.

6.2.5 Camera Tracking

Camera testing happened once when we got the camera and would wire it up to a device and test the functionality. We were looking for it to properly showed an image with the quality it had described in the description, if this was not achieved, we were sending it back and finding an alternate camera to used that was up to standards. We then were testing it further by using our object detection software and set up an environment that would easily detect objects to saw if it would put a box around the image to identify. When this test was done, we were testing how well the camera communicates with the software for it to accurately detect objects.

Figure 67: Image Testing for Camera Tracking learning program



For our actual testing of the project and camera specifically, we were in a field. The field had a route created from objects we had and a cardboard living situation to test if it could travel through a fake city or neighborhood area to deliver the package. The camera would have to detect the area it would land, trees, poles, and other common objects that we had to represent what it might came across in the real world. We had examples of other testing environments that we took ideas of testing from, such as shown in **Figure 67**, where they used images of people and started classifying them by parameters to steadily step by step teach their program what was a person and what was not.

6.2.6 Stress Testing Drone in Flight Operations

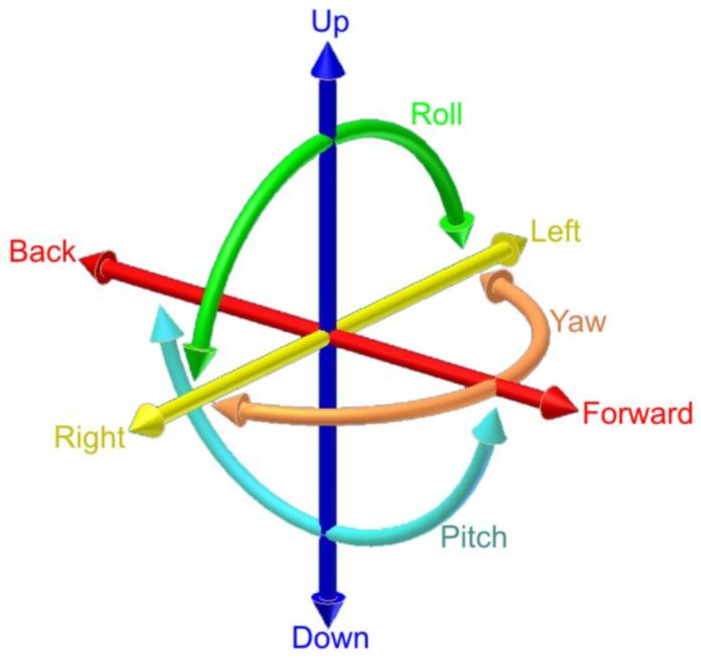
Now, during the testing and operation of our drone product, there was times when during nominal and normal flight conditions, there was unexpected factors that could interfere with the flight path. This does not mean gust of wind or movement of the package within its own casing and whatnot, as those would have been compensated for within the flight controller's normal processes and should be accordingly easy to account for. What these stress tests were for were the abnormal cases that were not taken into account by the flight controller's original programming. Such as actual objects interfering with the flight path or crashing into the drone during operation, which could cause the drone to unexpectedly have to overcorrect itself in regular flight which could led to crashing. To combat this, we used our accident-avoidance software to give the drone an additional set of "instincts" that took over and perform the correct procedures when such situations occur. This could be very stressful to the flight system and its ability to fly if perchance, the drone was not in a situation where it was in the way of something such as a building or tree that it could easily read and moved out of the way from, the drone would needed to took extreme evasive maneuvers to had a chance of dodging something that was going at any reasonable speed that it could catch up to a drone mid-flight. So of course, during these corrective course adjustments we were going to had more extreme G forces and higher variances in pitch and yaw when flying, thus leading to a much higher potential in crashing anyway if we do not find the parameters that perform admirably during such maneuvers. As we saw in **Figure 68**, crashes could be devastating for the future continuation of our project, and the needed to test parameters that could induce such crashes to needed to highly regulated in simulation and out of it so as to give us ample data to correct any instance this devasting impact to our project could had.

Figure 68: Drone Crash During Operation

We saw earlier in this paper that our ground control program, QGroundControl, had a robust and easily manageable parameter system. It also lets us easily saw what our changes to the parameters would do to our drone in summaries given for each different parameter that was easily understood by the programming. Things that were from sensors that aren't part of the flight controller programming aren't as intuitive but could still be found from other sources, such as the input of an infra-red sensor. So, the action of dodging may be carried out by parameters that were easily described and changed to control how any actual action that our accident-avoidance program took could be easily fine-tuned were more easily achievable for the best results.

For the best description of how our accident-avoidance program would achieve favorable results was to describe a pilot might actually dodge something in a regular helicopter, as that was the closest comparison in-flight standards that could be shown. In **Figure 69** we saw a graph showing how each variable of flight influences direction and could rapidly affect which direction a plane or helicopter could took.

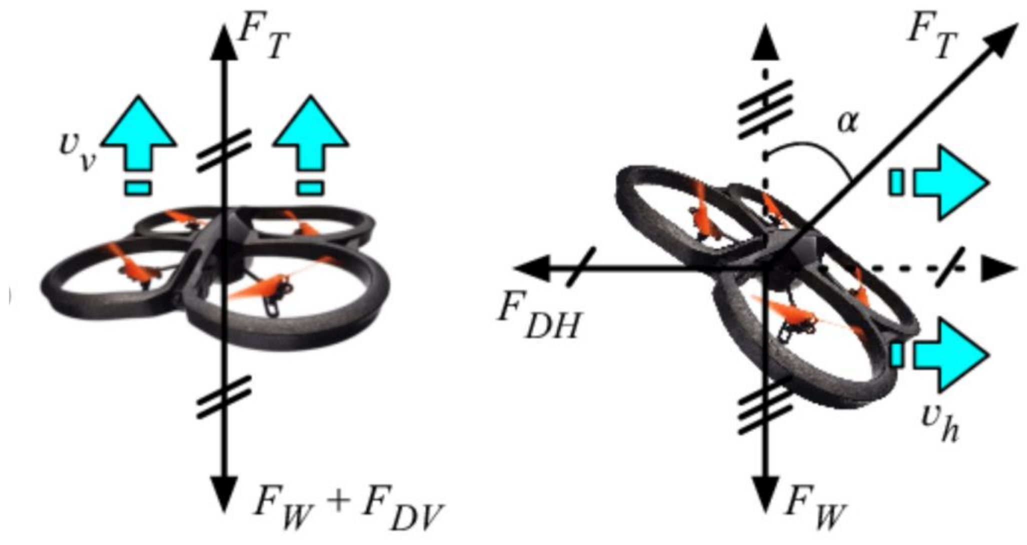
Figure 69: Pitch, Yaw, and Role



We knew from other instances of crashes and successful dodges in such situations that needed such fast-paced movement that it was the pitch, yaw, and roll were precise that affected the plane or helicopter's chance of actually dodging such objects that could interfere with their flight path. The pitch was the action that our drone took when moving the front of the drone up or down, thus affecting the speed with which it was traveling forward or slowing down. So, when we came into a situation where the accident-avoidance software activates and takes temporary control of the flight controller, this was the first action that such programing took. Slowing down by decreasing the drone's pitch would increase the time the drone had to react to the incoming object that would interfere with the flight path. Decreasing the pitch would lift the front of the drone so as to had more downward force from the propellors acting in the opposing direction of our previous forward movement. This action could cause stress upon the motors and possibly the package that the drone was carrying at the time, so we needed to test parameter adjustments for this action just as any other action. This action however was less intense than others as it does not require intense side-to-side movement that could affect the drone's ability to even had a stable hover. The reason we were testing this action in a testing environment was because of the other aspects of the drone's continued operation. Such as the motor, sensors, propellors, and even the controllers and data processing board that could be negatively affected by G forces that they were not designed to took. If these tests were done properly the components could be stressed to their breaking points. The G forces could have effects on those items that we did not understand until testing them. We needed to carefully test and perfect the act of slowing down with pitch as so to not hamper the drone's continued ability to fly to the objective. This did end up being one of the prime reasons we had our first crash during testing.

Continuing with the two parameters that needed to stress test our drone with was yaw. This was the action showed in **Figure 69** that enables the drone to fly left or right and permits the drone to change the direction it was heading when moving forward. As we were slowing down using pitch during any instances of the accident programming activating, the drone needed to adjust its yaw so as to slightly adjust any forward motion it was still achieving during the incident, as the pitch would drastically decrease forward movement, but it would not be able to completely erase it during operation in such a fast-paced instance. So, to made sure our drone was using the left-over forward motion to its own advantage we would increase the yaw to the left or right depending on location and other factors that the drone had to took into account in that moment of movement such as buildings or trees or objects around it to achieve an effective “dodge”. Parameters needed to adjust in the program during stress testing to made sure that again, during evasive maneuvers our sensitive equipment would not be so negatively affected as to not be able to continue with the designated mission and became less than effectively operable, as we would still need our sensors for future missions. These testing measures were imperative for making sure everything was ready to go because they would prevent causing damage to unnecessary equipment. Keeping the equipment as safe as possible adds to the longevity of the drone and lifecycle of equipment, which aligns with our standards and realistic design constraints of the project.

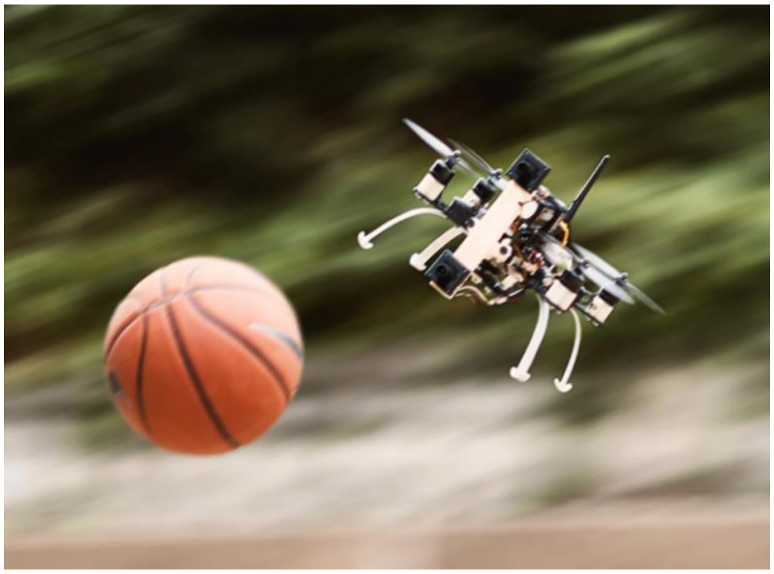
Figure 70: G Forces Acting Upon Drone



And finally, after we had tested both our pitch and yaw, so as were perfectly adjusted for evasive maneuvers such as slowing down with pitch, and turning left or right with yaw, we needed to adjust and test our roll parameter, which we saw in **Figure 69** changed the angle with which the drone was approaching its forward motion. The roll function could almost be considered our most important parameter to adjust and test as this action was the thing that could decrease our system G force and actual stress on our system during the accident-avoidance action. By decreasing or increasing our roll we achieve an angle that had the G forces acting upon our drone during such instances were less or more depending on the other two parameters at the time. We could saw this in **Figure 70** as the G forces working a drone during flight and when the angle of the drone changes the direct direction of the acting G forces changed as well. Keeping these factors in mind when creating the project was a part of the attention to detail needed for the accuracy that led to the drone performing how we wanted it to. It could be easy to misinterpret the variables or accidentally assign the wrong one to them and then the controls were completely backwards.

When we had adjusted and tested the exact stress factors that our drone and all its components could handle in a series of stress tests in simulation and in real environments, could we assuredly perform accident avoidances in the correct and effective outcome our drone needed in such instances where time was of the essence? We saw the correct and hopefully, future success of our drone in **Figure 71** where a drone had successfully adjusted all its pitch, yaw, and roll parameters to perfectly dodge out of the way of incoming objects.

Figure 71: Drone Correctly Dodging Incoming Object



7. Guidelines and Procedures

There was nothing more important to our project into ourselves than our own personal safety. We model our safety guidelines set by OSHA and common-sense policies and PPE experienced through members of our group who had a military background and Industry experience. In following our safety protocol, we would ensure no one in our group was injured during the testing of our drone.

7.1 Safety and Procedures while Testing

Our group intends I am having a very strict started-up sequence before testing our drone. We believed it was important to have a strict started-up sequence before testing the drone for safety reasons. Drones could be quite dangerous.

Some of the dangers or risks our group undertakes when testing a drone were as follows. Drones' propellers had very high rotations per minute and could easily cut or slice. One of the worst possible outcomes for a drone, the propeller was to take a finger of which was not unheard of in the drone hobbyist community. Other concerns when testing our drone was the drone accidentally running into us or falling out of the sky onto us. Our group intends on taking measures to minimize all these risks when testing its delivery drone project.

To started off the list of ways in which we plan to mitigate these risks was to followed suit in the ways that gun ranges minimize risks when changing targets. First, we had opened communication on when we were about to start the drone and that no one was in front of the person controlling the drone. While standing a safe distance away from the drone. As well as making sure everyone was aware when the drone was being placed or being handled to not started the drone and if the drone was not started that the battery to disconnect to further ensure safety when handling.

During the testing of the drone hardhats and safety, goggles was worn by all members of our group. The reason safety goggles, and hard hats was worn was due to the unlikely event there if anyone was hit or ran into by a drone that the damages was medicated by our safety gear.

After the test when the drone had landed, and we had no further intentions of flying the drone. One member of our group would approach the drone with leather gloves on before disconnecting the battery. Once the battery was disconnected from the drone, we then deem the drone were safe were handled without gloves on. Once the drone was deemed safe were handled without gloves on, we were then able to make any adjustments to the drone before our next test.

Table 27: Safety While testing

Step	Task	PPE
1	Communicate that the drone was tested	Face Mask

2	Connect Drone to battery supply	Gloves
3	Step a safe distance from the drone	Safety Goggles/ helmet
4	approaching drone to pickup/repair	Gloves/ Safety Goggles
5	Unplug the drone from the battery	Gloves

So, the sequence in which our group started up before testing the drone was as follows. First, ensure opened communication that the drone was being handled and placed. While wearing gloves whenever the drone was connected to the battery power supply. Then before starting the drone that no one was in front of the person controlling the drone. While wearing hard hats and safety goggles. Lastly when approaching the drone after it had landed in preparations to pick the drone up out-group would wear gloves before disconnecting it from the battery power supply. While making sure everyone in our group understands that the drone was now safe were handled before testing again and was disconnected from the battery. As seen in **Table 27** above.

7.2 Tool Use and Procedure

Our drone delivery project would require us to used very specific tools to accomplish our goal of having a drone be capable as well as delivering a package while detecting objects. The tools in which used to assemble our drone delivery project or just as important as the parts we used. Without the proper tools we would never be able to assemble, test, and modify our drone to the correct specifications. The tools that we were used to assemble our drone and to test various parts were as followed.

First, for the assembly portion of our drone, a screwdriver was critical in attaching all of the arms and legs to the body of the frame as well as securing the Motors and the propellers in the correct orientation of our drone. Secondly, we used wire cutters to attach the correct male or female end of connection leads accordingly to match the assembly of our drone. Third, we used a soldering iron to Sauter any parts that we may needed to add to our PCB design or other boards as they needed it. Lastly, we were using a multi-meter to check tire distribution throughout the board to made sure that we were getting the correct voltages throughout.

In terms of software, there’s also a tool that we had to used. The tool that we were using for software was a laptop. The laptop would not only run the software we needed to code the proper procedures. We would also be using the laptop as a controller for the drone so that we were manually able to maneuver the drone from the laptop.

So, list the tools that we were using in this project to build a package delivery drone. The tools were a screwdriver, wire cutters, soldering iron, Multimeter, and a laptop. As you could saw in **Table 28** shown below, the tools that we used were a screwdriver, wire cutters, soldering iron, multimeter, and a laptop and what those tools were and what their functions was for our drone delivery project.

Table 28: Tools

Tools and Functions		
#	**Tool**	**Functions**
1	Screwdriver	Assembly of drone frame
2	Wire Cutters	Spicing for battery leads
3	Soldering Iron	Hardware repair
4	Multi Meter	Testing Voltages
5	Laptop	Drone controller and Software testing

7.3 Parts Management

Parts management was a critical role for any project that wishes to meet deadlines. Without proper park management, your ability to obtain the parts you needed and to made sure they were available when you needed them was a delicate process. Our group had put in a lot of research to made sure the parts we used for this project was ready and ample for us to use at our leisure.

The procedure in which our group had called about part management was to minimize any bit of struggle to obtain a part. First, we looked online to saw how many available products we're listed as well as shipping time for those parts. The parts that had limited supply and longer shipping times were prioritized first. To prioritize them first we ordered them with ample time and ordered a few at once just in case anything got damaged in shipping or were damaged through our testing, case in point, this did happen and many of our original components had to be reordered to be replaced. Second, we purchased the parts that were not as scarce and had quicker ship times. For all parts, we messaged manufacturers to made sure they had no intentions of ending the lifecycle of any of their parts.

In terms of shipping, our group understands the climate in which the world sits with how Covid 19 was hurting the estimated time of arrival. As well as some international waterways being blocked could cause shipping delays. Our way of handling these and foreseeable challenges was to order the parts early and kept a count of how long they actually took were shipped. And if the parts took longer than expected to show up, we would plan accordingly to order more of those parts so that they showed up before they were possibly needed for a project.

In terms of storage and where to ship the parts we order. Our group had decided that it should be sent to one of our places of living to whoever was in the most centralized location between the four of us. As well as who was closest to campus in most convenient to reach.

Also, to whoever had the more available space to worked on the project if needed be. This keeps the items organized and in one spot for us to worked from and easier when contacting companies for replacements or such.

So, the ways in which we ensure proper parts management was clear as you could saw explained in **Table 29** below. We prioritize the parts that were less available and had a longer shipping date. Second, we kept track of how long it actually takes the parts to arrive at us and came to the conclusion if we needed to order parts again now before it's too late. Lastly, we store the parts in the most centralized location, so it was convenient for our project. With our parts managed in this orderly fashion, we believed we should run into no hiccups that would suck back our deadlines for a drone delivery project.

Table 29: Parts Management Steps

Parts Management	
step	**Goal**
1	Order the parts we believed might went out of stock soon and had the longest shipping dates
2	Note how long the parts actually took to ship in order accordingly if the part might break under testing
3	Keep parts in centralized locations

7.4 Finances

For any project were successful budgeting and financing were critical. The way our group had decided to budget and finance our project was it carefully thought-out system. We were a self-funded project so in essence the money put into this project was our own and we do not had sponsorship.

To made sure not one of our members was left holding the foot of the bell at the end of our project. We decided to implement a ledger for the person who bought the part tells us about the cost after shipping and then divides that cost evenly amongst all members of our group. Screenshots of the receipt the ship dates and the cost were all required to made sure we had an itemized list of where the cost was coming from. We also created a Google Docs form to fill out to helped created an itemized budget at the end of our project. We believed having everyone paid 1/4 the cost of every product we buy was the most efficient way of keeping a balanced budget without having any overlapping money exchanges.

We believed with following this procedure of financing that no one in our group was left footing the bill. In the end, we had a very organized and detailed Ledger of how and where the cost of our project came from. As seen in **Table 30** below the steps we took her as followed. First, purchase the part. Second screenshot of the receipt and shipping date. Third, fill out the Google forms with the receipt of the total cost and shipping dates. Lastly, receive 1/4 of the cost of your part from all project members.

Table 30: Finance Process

Finance Procedure	
Steps	**Action**
1	Purchase part
2	Screenshot part ordered and shipping date
3	Fill out Google forms
4	Receive payment

8. Administrative Content

8.1 Project Milestones

For our project, we created our milestones by starting with the hard deadlines given by the course and then dividing that up into smaller personal group deadlines that kept us on track. **Table 31** contained our deadlines for the first half of the project that concentrates on research and structure, then **Table 32** contained how we were splitting up the planning, decision making, and research that was going into creating the main paper and helping with ideas. **Table 33** contained a vaguer outline for how implementing and creating went in the second half. Our group consists of two computer engineers that concentrated on a lot of the software design choices and then two electrical students that concentrate on the power and hardware choices. We then took this knowledge and had meetings where we talked and had an understanding of what everyone was working on for the case someone needed to take over. This did in fact happen sometimes when the PCBs were being designed and needed to be continuously updated. We do not want to be too reliant on one person knowing something so specific.

Table 28 shows how we broke up accomplishing the paper deadlines and then in the behind the scenes we would conduct weekly meetings that had checkpoints for people to present their findings. We would also end meetings with objectives for the next time, which came in handy for the decision-making process because we had a decision needing were made and then took the week to all research and that following meeting present our findings and decide as a group which we went with for the project.

Table 31: Main Timeline

#	Task	Start	End	Status	Responsible

1	Ideas	1/11/21	1/29/21	Completed	Group B16
2	Project Selection and role Assignment	1/25/21	1/29/21	Completed	Group B16
3	Divide & Conquer 1	1/25/21	1/29/21	Completed	Group B16
4	Divide & Conquer 2	2/1/21	Feb 12	In Progress	Group B16
5	Table of Contents	2/1/21	4/27/21	In Progress	Group B16
6	20 Page Submission	2/1/21	2/16/21	In Progress	Group B16
7	40 Page Submission	2/16/21	3/9/21	In Progress	Group B16
8	60 Page submission	3/9/21	4/1/21	In Progress	Group B16
9	80 Page Submission	3/9/21	4/5/21	In Progress	Group B16
10	100 Page Submission	4/5/21	4/15/21	In Progress	Group B16
11	First Draft	2/1/21	4/20/21	In Progress	Group B16
12	Final Document	2/1/21	4/27/21	In Progress	Group B16

Table 32: Research, Documentation, and Design

Research, Documentation, & Design					
14	Frame design	2/1/21	4/2/21	Researching	Raymond & Austin

15	Sensors	2/1/21	4/2/21	Researching	Raymond & Austin
16	Motors and blades	2/1/21	4/2/21	Researching	Raymond & Austin
17	Controller	2/1/21	4/2/21	Researching	Raymond & Austin
18	PCB Layout	2/1/21	4/2/21	Researching	Raymond & Austin
19	Power Supply	2/1/21	4/2/21	Researching	Raymond & Austin
20	Computer Vision options	2/1/21	4/2/21	Researching	Ellie & Dan
21	Algorithm code outlines	2/1/21	4/2/21	Researching	Ellie & Dan
22	Languages to code in	2/1/21	4/2/21	Researching	Ellie & Dan
23	platforms for the code	2/1/21	4/2/21	Researching	Ellie & Dan
24	Set up Github repository	2/1/21	4/2/21	Researching	Ellie & Dan

Table 33: Senior Design 2

Senior Design 2					
25	Build Prototype	TBA	TBA	TBA	Group 16
26	Testing & Redesign	TBA	TBA	TBA	Group 16

27	Finalize Prototype	TBA	TBA	TBA	Group 16
28	Peer Presentation	TBA	TBA	TBA	Group 16
29	Final Report	TBA	TBA	TBA	Group 16
30	Final Presentation	TBA	TBA	TBA	Group 16

8.2 Project Budget

For our budget we were preparing for the worst so that if something goes wrong, we had wiggle room to buy more, we understood things would break or be defective, so we made sure to plan accordingly. As seen in **Table 34** we had allowed ourselves to had multiple drones' propellers and batteries in case things went south or break. In **Table 35** we allowed ourselves to had multiple IR Sensors due to our research showing they don't always work effectively, and it was cheaper to buy in bulk. Additionally, we used one camera and numerous altitude sensors to made sure we could fly safely and effectively. For **Table 36** we were planning to have a radio controller to helped with returning the drone and with troubleshooting to made sure our drone runs as we wanted it to. The controller having Wi-Fi had the communication standards that we needed to properly tell the hardware and software what did. Overall, we were staying optimistic about what we needed but prepared to give a little more as with drones, crashes did happen and resulted in breaks that meant buying new items.

Table 34: Physical Components

Physical Components	
Drone Final x 1	$150 - 200
Testing Drones x 2	$100
Propellers x 20	$20
Housing Compartment x 1	$50

Batteries (Drone and Sensors) x 3	$200

Table 35: Sensing Components

Sensor Components	
Altitude Sensor x 2	$30
IR Sensor x 9	$20
Camera x 3	$200

Table 36: Electrical Components

Electrical Components	
RF Transmitter x 2	$14
Controller	$130
Charger	$50

9. Conclusion

The D.E.A.R. Drone was able to show how object avoidance could be accomplished for autonomous drones used for package delivery. By optimizing the use of power and sensors we created a drone that could properly deliver a package while also avoiding objects. LiPo batteries were used as the power source to keep it optimized for efficiency of delivery. Ultrasonic sensor and a camera were used for the detection of objects and being able to saw and found the correct path. There were also sensors to detect the location of the drone and how fast it was going to provide accurate travel time. This project incorporated machine learning to optimize how the drone would travel and deliver while avoiding objects. On the electrical side, it worked on optimizing the batteries and controls with sensors. Embedded systems were used to helped communicate the system holistically and perform the mission. We also tried to maximize on a smaller budget than typical companies, which would hopefully lead to showing this advanced technology was affordable and reliable because whenever there were new breaks in technology it could be intimidating to the population and community.

We were able to create this idea by combining the skill set of everyone on the team as it was made up of 2 electrical engineer students and 2 computer engineering students. Our backgrounds all came together to helped guide us in a great direction with some people having backgrounds with machine learning, machine learning, electrical analog systems, or some past drone manufacturing experience. All of these subjects contributed to the decisions being made and trying to make the best choices for a quality delivery drone. We had also quickly realized that there were many parts that went into creating a drone and while we could have tried and made one person decide the motor, another person decides the frame, and others decide the flight controllers, it was seen quickly that they were all connected and sometimes one decision could not be made without another. So, this helped organized our plan and decision-making and created a domino effect. We started with picking out the frame, which led to the motor decision, which led to the ESC decision, and others because of how they all interact as a system.

The drone was incorporating a self-guided flying system that was able to go from point A to point B without someone using a controller but also uses a camera and computer vision in order to avoid objects and made new paths depending on the circumstance. From the software side, the project included using C++ for the embedded systems and hardware portions and then using Python for the main software coding sections that also had computer vision. The computer vision implemented Python because it was clear how easy and widely used it was in the machine learning and computer vision community. There were libraries of TensorFlow and more that was used to accomplish the goals for the project.

When deciding the parts, sitting down and discussing what was the most important aspect of the drone was important because if not having a priority, then the project became expensive, and some decisions needed sacrifices in order to accomplish the right thing. We started with our frame because that would decide the size of things going further and would decide how much battery and motor power we needed.

Automated drone delivery was slowly becoming normal and trying to enter the world were more common. Through research, we found that many delivery companies like Amazon and such were already starting to test drones in the US and other countries to test how accurate and applicable they were to the population. We were hoping to discover what goes into these machines and the improvements that could be made to them in order to perform more properly for the area the drones were traveling in. After going through this past year of Covid and living through a pandemic it was seen everywhere places shutting down to tried and protect people. Delivery drivers and people apart of the shipping process were put in harm's way constantly. Having improvements with delivery drones could have helped immensely with taking the people out of harm's way even if it was made accessible for them to used drones in the process.

Our budget going into this was hoping to spend around $1000 with about $200 wiggle room to go above if needed for accidental problems. We were able to research the measurements and types of parts we needed then once we knew what we were looking for specifically, used our resources of Google, Amazon, and eBay and proceeded to find the best prices for it. Sometimes this led to great savings because, for our frame, we were able to find a frame kit with the exact dimensions and accessories needed to put it together. This saved a tremendous amount of money because buying it all together saved from buying things individually. The next biggest expense seen was with the flight controller and Raspberry Pi. The computer brains of the drone could get extremely pricey as the technology that goes into these tiny minicomputers was expensive to created, but luckily this should be a one-time purchase and not be a burden on us later. It was also not necessary to immediately started working with this because there was simulation software that makes it easy to initially test code and what was happening to drone before uploading stuff onto the raspberry pi and flight controller.

Appendix.

Bibliography

[1] "Wing," [Online]. Available: https://x.company/projects/wing/.

[2] N. Heath, "Tech Republic," 4 April 2018. [Online]. Available: https://www.techrepublic.com/article/project-wing-a-cheat-sheet/. [Accessed 2 February 2021].

[3] "Drone Nodes," [Online]. Available: https://dronenodes.com/drone-esc-electronic-speed-controller/. [Accessed 1 February 2021].

[4] "eBay," [Online]. Available: https://www.ebay.com/itm/Qwinout-F450-4-Axis-Airframe-450mm-Drone-Frame-Kit-w-30A-ESC-A2212-1000KV-Motor/284005585210?hash=item42200d513a:g:4-IAAOSwENVfWIUH. [Accessed 1 February 2021].

[5] "PLCGURUS.NET," [Online]. Available: https://www.plcgurus.net/drone-programming/. [Accessed 10 February 2021].

[6] Y. Nader, "hackr.io," 08 January 2021. [Online]. Available: https://hackr.io/blog/python-vs-java. [Accessed 10 February 2021].

[7] Shawn, "SEED STUDIO," 2020. [Online]. Available: https://www.seeedstudio.com/blog/2019/12/23/distance-sensors-types-and-selection-guide/. [Accessed 5 February 2021].

[8] "electronicsnotes," [Online]. Available: https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/standards.php. [Accessed 20 February 2021].

[9] J. Reagan, "dronegenuity," [Online]. Available: https://www.dronegenuity.com/lipo-drone-batteries-users-guide/.

[10] O. Liang, "Oscar Liang," Feburary 2017. [Online]. Available: https://oscarliang.com/lipo-battery-guide/. [Accessed 20 02 2021].

[11] Alex, "Drone Trest," 2 September 2015. [Online]. Available: https://www.dronetrest.com/t/power-distribution-boards-how-to-choose-the-right-one/1259. [Accessed 15 March 2021].

[12] NA, "IEC," March 2021. [Online]. Available: https://www.iec.ch/government-regulators/electric-motors.

[13] QWinOut, "Ebay," Ebay, 14 April 2021. [Online]. Available: https://www.ebay.com/itm/Qwinout-F450-4-Axis-Airframe-450mm-Drone-Frame-Kit-w-30A-ESC-A2212-1000KV-Motor/284005585210?hash=item42200d513a:g:4-IAAOSwENVfWIUH#shpCntId. [Accessed 14 April 2021].

[14] NA, "IP and NEMA standard," March 2021. [Online]. Available: https://www.delvallebox.com/news/en/ip-and-nema-standards.

[15] NA, March 2021. [Online]. Available: https://www.delvallebox.com/news/en/ip-and-nema-standards.

[16] NA, "BatteriesInAFlash.com," NA, Janurary 2015. [Online]. Available: http://www.batteriesinaflash.com/c-rating-calculator. [Accessed 14 April 2021].

[17] A. Jacobs, "QuadQuestions.com," Janurary 2015. [Online]. Available: https://quadquestions.com/blog/2014/12/27/qav250-c-rating-explained/. [Accessed 15 March 2021].

[18] Turnigy, "HobbyKing.com," Hobby King, January 2020. [Online]. Available: https://hobbyking.com/en_us/turnigy-5000mah-3s-20c-lipo-pack-xt-90.html?queryID=&objectID=69450&indexName=hbk_lived_magento_en_us_products_hbk_price_stock_2_group_0_asc. [Accessed 1 April 2021].

[19] FAA, "faa.gov," Federal Aviation Admin, 1 January 2021. [Online]. Available: https://www.faa.gov/uas/recreational_fliers/. [Accessed 10 April 2021].

[20] NA, "DroneQuadCopterX.blogspot.com," NA, 2 January 2016. [Online]. Available: https://dronequadcopterx.blogspot.com/2019/01/quadcopter-power-distribution-board.html. [Accessed 10 April 2021].

[21] "U.S Department of Transportation," March 2021. [Online]. Available: https://www.transportation.gov/tags/drones.

[22] NA, "mttr.net," Matternet, 17 November 2020. [Online]. Available: https://mttr.net/. [Accessed 10 April 2021].

[23] NA, "uavcoach.com," NA, January 2015. [Online]. Available: https://uavcoach.com/where-to-fly-drone/orlando/. [Accessed 10 April 2021].

[24] March 2021. [Online]. Available: https://www.tesla.com/autopilot.

[25] M. Russia, "instructables.com," NA, January 2017. [Online]. Available: https://www.instructables.com/PVC-Drone-Obstacle-Course/. [Accessed 14 April 2021].

[26] R. Harris, "bls.gov," BLS, December 2016. [Online]. Available: https://www.bls.gov/opub/mlr/2016/article/pdf/suicide-in-the-workplace.pdf. [Accessed 14 April 2021].

[27] NA, "docs.qgroundcontrol.com," 1 January 2020. [Online]. Available: https://docs.qgroundcontrol.com/master/en/SetupView/Parameters.html.

[28] I. School, "iacschool," 13 Febuary 2017. [Online]. Available: https://icaschool.com/2017/02/13/drone-thermal-imaging-next-wave-uav-inspection-services/. [Accessed 15 April 2021].

[29] NA, "easyeda.com," EasyEDA, 15 April 2021. [Online]. Available: easyeda.com. [Accessed 15 April 2021].

[30] I. Y. S. L. J. P. Jaehoon Jung, "Object Detection and Tracking-Based Camera Calibration for Normalized Human Height Estimation," *NA,* vol. 1, no. 1, p. 10, 2016.

[31] NA, "embention.com," Embention, 21 Febuary 2020. [Online]. Available: https://www.embention.com/news/sensor-calibration-in-drones/. [Accessed 21 April 2021].

[32] M. Irving, "New Atlas," 19 March 2020. [Online]. Available: https://newatlas.com/drones/drone-dodgeball-obstacle-detection-system/. [Accessed 25 April 2021].

[33] NA, "Emissary Drones," NA, 25 Sep 2017. [Online]. Available: https://emissarydrones.com/what-was-roll-pitch-and-yaw#:~:text=PITCH%20%E2%80%93%20This%20moves%20the%20Quadcopter,which%20way%20it%20is%20tilted.&text=YAW%20%E2%80%93%20This%20moves%20the%20Quadcopter,stays%20level%20to%20the%20ground.. [Accessed 25 April 2021].

[34] Y. B. D. B. A. M. E. P. M. Chen, "A Case for a Battery-Aware Model of Drone Energy Consumption," *research gate*, vol. 1, no. 1, pp. 1-8, 2018.