Autonomous Obstacle Avoidance for Unmanned Aerial Vehicles (UAV)

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Sammy Nayhouse

Spring, 2022

Technical Project Team Members

Samir Chadha

Patrick Hourican

Chase Moore

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harry C. Powell Jr., Department of Electrical and Computer Engineering

Statement of work:

Samir Chadha

I worked on the software side of the project. This included making sure the sensors could read in data and print out specific values to a computer, integrating ROS with the sensors so that the sensors could publish to specific ROS topics, working with the Gazebo simulator to test the sensor integration with the drone, creating python scripts to integrate the sensor readings with the drone so that the drone's movement would be affected by sensor readings, integrating the joystick with the drone to allow for teleoperation, implementing autonomous obstacle avoidance for the drone based on sensor input, and porting over the software from the remote server to the Jetson to allow for a real life drone to use the software we developed to autonomously avoid objects. I also engaged in debugging all of the scripts that allowed for the sensors to affect drone movement along with the teleoperation of the drone.

Patrick Hourican

I worked on the software side of this project, along with the 3D CAD design. On the software side, I set up the simulator and ROS environments on our computer to enable the testing of the software implementation with the sensor integration. Also, I aided in the implementation of the Joystick programs to allow teleoperation of the drone. Additionally, I aided in the development and debugging of other python scripts that were published to rostopics. Finally, I designed, updated, sliced and 3D printed the 3D designs for the project, creating several different iterations of specific structures and testing them to see which fit our intended design the best.

Sammy Nayhouse

I worked on the majority of our hardware implementation, including both the PCB design and hardware interface with the embedded microcontroller. I designed the 4-layer PCB, with two signal layers, a power plane, and a ground plane, using Altium designer software. I chose all the components for the PCB, as well as the headers to physically interface with the sensors, motor and microcontroller. I read through datasheets to ensure the I2C multiplexer we utilized would work for our sensor configuration, delay requirements, power requirements, and physical footprint requirements on the PCB. I also created a chassis around the PCB to create an effective Faraday cage, due to the large amount of EMF interference on the drone caused by WIFI and GPS signals being transmitted back and forth from the tele operational joystick. I also implemented the test plan for our PCB to ensure all requirements for the system were met and our software system could be interfaced seamlessly.

Chase Moore

I worked on a wide variety of sections of our project. The first task I worked on was choosing a battery that would be within the scope of our project. This involved doing several calculations such as the current and voltage needed for each component and then choosing a battery that would meet design specifications. On the circuity side of the project, I helped Sammy to design parts of the PCB such as all the headers to sensors and forty pin ribbon cable. Also, I implemented our test plan for the PCB in order to resolve any issue that existed and ensure that it would function properly. In terms of the software, I helped with interfacing with sensors and reading and collecting data. Lastly, I had the role of maintaining the budget for our project as well as ordering any of the components that we needed. This involved keeping track of a detailed spreadsheet with all of the components' prices, specifications, and distributors.

Table of Contents

Contents

Table of Figures/Tables

Abstract

Team SPSC designed and developed an aerial platform to enable shared autonomy and obstacle detection for unmanned aerial vehicles (UAV). A printed circuit board (PCB) was designed and manufactured to aid in UAV obstacle avoidance using a series of 1-D time of flight light detection and ranging (LIDAR) sensors [1][2]. A servo motor was utilized in order to increase the range and visibility of the front sensor. This allows obstacle detection within the range of the UAVs movements and simultaneously delivers real-time data through inter-integrated circuit (I2C) communication protocols [3]. A 3-D printed tower was manufactured to physically mount the PCB, Lidar sensors and motor to the aerial platform. An embedded Robotic Operating System (ROS) was used to visualize real time Lidar data and simulate obstacle avoidance for UAV systems [1][4].

Background

Flying is the dream of many but only a few can truly experience it freely. What was once seen as science fiction, however, is now becoming a reality as new advances in computation and sensing lead to the development of advanced robotic systems. Autonomous mobile robots, and in particular aerial vehicles, are entering our society and finding many applications like aerial photography, infrastructure inspection, surveillance, hobby application, and even search and rescue operations. This has led to a common debate among experts regarding if fully autonomous aerial vehicles provide greater benefits than human controlled aerial vehicles. Drone crashes, safety challenges, security issues and privacy concerns have led many to believe that fully autonomous unmanned aerial vehicles are not worth the risk or cost.

This project was chosen to address the common types of issues with fully autonomous UAVs. Robotics is an area that interests all members of the team, and previous project and internship work with autonomous technologies paved the way for creating a potential solution to current issues with fully autonomous aerial vehicles. Our team's goal was to investigate and enable a form of shared autonomy that incorporates both capabilities – human controlled input and onboard autonomy. This compromise allows for desired human input (e.g., flying a drone to an area of interest), while keeping the system safe (or performing other tasks) through onboard autonomy and obstacle avoidance. This will allow for a much safer and efficient drone that is less likely to crash due to operator error, which our team believes is a necessary development in the advancement of autonomous UAVs.

Similar projects pertaining to shared autonomy have been completed in the past. A commercial-off-the-shelf (COTS) aerial vehicle was enhanced to include mathematical models of the quadcopter dynamics for real-time improved flight stability [5]. The American Automobile Association did an experiment on semi-autonomous driving technology that utilized

systems that assist drivers in driving and parking functions and found that collisions with cyclists occur 33% of the time at four-way crossings [6].

This project is different from past work by others because of the simultaneous use of both human controlled inputs and onboard autonomy, instead of switching between one or the other. For example, if the human is manually flying the drone forward, the robot can override the manual input and stop the UAVs movement (or change the movement) in order to avoid an obstacle. Previous projects with shared autonomy usually have two operating modes, either manual (human) or autonomous. This method, however, analyzes both manual inputs from a human operator and onboard sensors/autonomy to avoid obstacles while completing the desired flight plan or performance tasks.

This project draws from a variety of previous coursework completed by our four team members. ECE coursework, such as the FUN series (ECE 2630, ECE 2660, ECE 3750), which allowed us to successfully design a PCB component using voltage regulators, filters, and so forth to interface with the motors, sensors and Jetson computer. The topics learned in the Embedded Robotics courses (ECE 3501, ECE 3502) will allow us to incorporate an embedded system by interfacing with a microcontroller and implementing object detection and data communication through I2C communication protocols. Coursework from the CS department (Samir Chadha and Patrick Hourican are CPEs), such as Operating Systems (CS 4414), will allow for swift Robotic Operating System (ROS) development and the implementation of Lidar data visualization and drone simulations.

Physical Constraints

Manufacturing & Usability

This project is safe and robust enough to ensure the electrical components are able to withstand the normal uses of drone flight and accidental bumps. The components on the drone are able to withstand all flying conditions in order to not be disconnected or altered during flight. The PCB will be inexpensive to manufacture as well as solder the appropriate components to meet design specifications. This design was manufactured and constructed in a way, so that it could be recreated and scaled to fit similarly structured drones that require obstacle detection. Such a design can be replicated and utilized for obstacle detection and adapted to avoid obstacles as the drone travels.

Part Availability

Due to our design's modularity, it could easily be implemented on other drones within the CoStar fleet. The parts we are utilizing in our design are widely used within industry, and all parts are readily available and able to be shipped quickly from well-known electrical part suppliers, such as Mouser, DigiKey, and other online suppliers.

Economic and Cost Constraints

The goal of this design is to create a modular design incorporating four sensors, one motor, an embedded microcontroller (Jetson), and an external power supply (battery). The components must be lightweight and meet weight constraints, while also being compact and capable of fitting on the drone. These are the primary constraints, as cost constraints are not as relevant due to the funding of this project by CoStar. All purchases will be itemized and receipts maintained throughout the design, but our team has been encouraged to purchase and test a variety of products that may make our design more efficient, safe and robust.

Tools Employed

Altium Designer

Altium Designer was used for circuit design and PCB design. Both tools have been used in previous internships so a large amount of knowledge already existed in the team. Overall, the simplified process of making a PCB with Altium Designer involved creating a new project and PCB design, adding components to the design, generating the connections between the components, routing the connections to create the final layout, and then generating the manufacturing files that will be used to manufacture the PCB. There were no new skills learned within using this software, but our abilities were sharpened in that we could go through the above process quicker the more we used it.

FreeCAD

In terms of 3D printing, we used FreeCAD which is a free and open-source CAD program that can be used to create 3D models. The process involved printing incorporated creating a 3D model in FreeCAD, exporting the model as an STL file, slicing the STL file to generate G-code instructions, and then using a 3D printer to create the physical object based on those instructions. Again, there were no new skills learned within using this software, but our abilities were sharpened in that we could go through the above process quicker the more we used it.

Cura and Ultimaker S3

In order to 3D print our FreeCAD design we used the Cura Slicing software and exported the files to the Ultimaker S3 3D printer.

Robotic Operating System

Another tool we employed was using ROS (Robot Operating System). Essentially, ROS is a set of software libraries and tools that provide a common framework for developing robot applications. It is often in various areas such as in research and academic settings, as well as in the development of robots. In terms of our project, ROS provided a set of tools and libraries for building autonomous robot applications, which included support for low-level device control and high-level robot capabilities such as planning and navigation. In order to implement Python scripts in ROS, we first needed to create a new ROS package and make sure that it has a proper directory structure and build configuration. This usually involved creating a *src* directory, where our Python scripts would be stored, as well as a *CMakeLists.txt file* that specified how to build your package.

One major aspect that we had to adhere to was that our Python code needed to follow the ROS conventions for writing nodes, including importing the rospy library and using the rospy.init_node() function to create a new ROS node. This also involved implementing any ROS message types that were used to communicate with other nodes, as well as any ROS topics, services, or actions that are used to publish or receive data. From here, we could use the roscd and rosmake commands to navigate to our package directory and build our package. From our Python we could compile and generate the necessary files that were needed to run our Python scripts within ROS.

In order to run our Python scripts within ROS, we used rosrun command, followed by the name of our package and the name of our Python file. This would start our Python script as a ROS node, allowing it to communicate with other nodes and participate in the ROS system.

For this we all had little to no experience using ROS, so this involved all of us going through several tutorials on how to incorporate this technology into our project.

Gazebo and RViz Simulator

Gazebo is a robot simulation software package used to simulate a wide variety of different robots, including ground and aerial robots. It is used by a number of different organizations to test and develop robotics technology. Gazebo is typically used in conjunction with other software packages, such as ROS in our case, to create complex, realistic simulations of robotic systems.

RViz is a 3D visualizer for robotic systems. It is part of the ROS framework and is typically used to visualize data from a variety of sensors, such as LiDAR in our project. RViz allowed us to see a 3D representation of the robot and its environment, which was helpful for debugging and analyzing the robot's behavior. It was also used to display data from other sources.

Both Gazebo and RViz we had no experience using before, so we had to use several methods to gain this knowledge such as textbooks and online tutorials.

Societal Impact Constraints

Environmental Impact and Sustainability

In terms of building the drone the biggest environmental impact would be the use of lithium batteries. When creating lithium batteries there needs to be a large supply of water involved, specifically 500,000 gallons per metric ton of lithium. This can lead to a scarcity of water in places due to utilization of such a large amount for this process. For example, 65% of water is consumed by mining in Salar de Atacama which is a massive salt flat located in Chile [7][8]. Ultimately, this had led to locals finding sources of water from elsewhere to support the community.

When disposing of the batteries, they contain several metals, such as nickel, manganese, and cobalt that can be very harmful to bodies of water as well as the environment if they are not disposed of properly in say a landfill for example. In addition, these batteries can't be reused in new batteries because of the lithium cathodes becoming degraded after a certain amount of time. With this being said, lithium batteries need to be recycled in an appropriate manner to prevent further damage. According to the United States Environmental Protection Agency (EPA) it is recommended that lithium batteries be sent to a recycling center that is nearby to one's location. The EPA lists two resources to find a recycle which are Earth911 and Call2Recycle. Both of these specialize in finding relevant locations depending on zip code. The EPA describes how the batteries should be packaged before being sent off and this can be seen below [9].

When looking at our device in terms of sustainability there are several benefits it brings. One of which being the capability of drones being able to aid in inspection of agricultural fields and crops [10][11]. Due to the typical large amount of land on most farms this is very beneficial in being able to see what is happening all over the entire area in a relatively quick manner.

In addition, our device could be used to have an aerial view of solar panels and wind turbines in plants [12][13]. This allows one to easily identify problems such as malfunction of failures. Essentially, this would eliminate the need for someone physically having to inspect these systems which would require a lot of time and is unsafe in certain situations. These are just a few of the ways our device will be sustainable and especially in the environmental aspect.

Health and Safety

This project takes into health and safety by including a hybrid option that involves a person operating the drone as well as the autonomous capability. For example, if the operator was flying the drone and the drone sensed a car, the drone would take over and avoid running into the car. This aspect makes our drone much safer compared to ones currently used in industry today.

Ethical, Social, and Economic Concerns

Our project might affect society, both from a human interaction perspective as well as economic one in terms of the ethics surrounding automated weapons systems. In particular, the ethics of automated weapons systems, is a topic that has been widely debated currently by experts in various fields all over the world. One of the main ethical concerns surrounding the use of these systems is the potential for them to be used to carry out violence without human oversight or accountability. This could result in unnecessary loss of life, especially if the weapons are not able to differentiate between civilian and military targets. Another concern is the potential for these systems to be used as a tool for governments or other groups to carry out violent actions without accountability. Overall, the ethics of automated weapons systems is a complex and multifaceted issue that raises important questions about the role of technology in warfare, the responsibility of individuals and governments in the use of force, and the potential risks and benefits of these systems.

External Considerations

External Standards

Per the IEEE 802.1 Standard for Wireless Networks, having 4 streams of data (1 for each sensor) would require a frequency band of either 2.4 GHz or 5 GHz to be used, with a bandwidth of either 20 MHz or 40 MHz. Depending on the bandwidth, we would have a different allowed data rate per stream. For example, if we utilized a bandwidth of 20MHz, the potential data rates per stream that we could use include 7.2 Mbps, 14.4 Mbps, 21.7 Mbps, 28.9 Mbps, 43.3 Mbps, 57.8 Mbps, 65 Mbps, or 72.2 Mbps. For a bandwidth of 40 MHz, we are allowed to use a data rate per stream of 15 Mbps, 30 Mbps, 45 Mbps, 60 Mbps, 90 Mbps, 120 Mbps, 135 Mbps, or 150 Mbps [11].

In addition, under the Federal Aviation Administration regulations Part 107.7 rules, the pilot for our system must have a remote pilot certificate with a small UAS rating and identification when flying the quadcopter [14]. Under section 107.12, however, it indicates that someone without a remote pilot certificate could pilot the quadcopter as long as someone with a

remote pilot certificate could take control of the quadcopter if necessary [14]. If the unmanned vehicle were to cause harm to someone or any damage, the individual with a remote pilot certificate should, under section 107.9, report if any person sustained serious injuries or if any damage was caused that exceeds $500[14]. Under section 107.29, we could not operate our quadcopter at night until our Lidar sensors allowed for the UAV to avoid obstacles, and under section 107.51, the group speed of the quadcopter cannot exceed 100 mph and cannot fly over 400 mph [14].

Intellectual Property Issues

While obstacle avoidance is a common topic for design with many different implementations already having patents, our project has several components that set our design apart from other patents. For example, the patent titled *Unmanned Aerial Vehicle Obstacle Detection and Avoidance* invented by Parag Mohan Kanade, Charles Wheeler Sweet III, and Jeffrey Baginsky Gehlhaar [15], detects and avoids obstacles using a camera that identifies the presence of an obstacle and notifies the operator to perform the avoidance (Independent Claim 1 and Dependent Claim 8). Our design differs from this by incorporating software to detect and avoid the obstacles autonomously. Additionally, our design uses LiDAR sensors rather than a camera.

Another related patent *Method for Training Heterogeneous Sensing System and Heterogeneous Sensing System* invented by 伊利亚·布雷瓦兹 [16] using sensors to detect obstacles in the environment of a system and analyzes this data through a processing unit (Independent Claim 1), leading to the adjustment of level of cognition for the sensor system to incorporate the obstacle conditions in the environment (Dependent Claims 5 and 6). Our sensor design and application differs from this design by processing the sensor data containing obstacle detection in the environment in real time. The information is processed and adjustments in movement are made in real time, rather than cognition adjustments.

The final patent to consider in relation to our projects design and patentability is titled *Method and Control Device for Identifying a Potential Collision Between an Unmanned Aerial Vehicle and an Object,* invented by Pablo Luis Guarnizo, Gabriele Michalke, and Thomas Michalke [17]. This design uses a camera to provide images to analyze the pixels representing the detected object, speed/position/direction information (Independent Claim 4) and detect the collision time between the vehicle and the obstacle in its path (Dependent Claim 6). Our design differs from this, as we incorporate the distance between the drone and the obstacle using sensor data (with the drone moving at a constant speed), and change the directory of the drone, rather than incorporating the trajectory of the obstacle through image analysis.

Detailed Technical Description of Project

For our project, we created autonomous obstacle avoidance for unmanned aerial vehicles,

commonly known as UAVs. Using a non-autonomous drone as our base (just a normal drone that a human would control with a joystick), and by adding our design to this base, we created a semi-autonomous drone with built-in safety features, capable of fully autonomous obstacle avoidance using data from real-time LIDAR sensors. At a high level, the logic is very simple. A human tele-operator begins navigating the drone towards a destination using a joystick. While the drone is flying, 4 LIDAR sensors (on the front, right, left, and rear of the drone) read in data and map the surrounding environment. If an object is detected too close to the drone (this could be a bird flying towards the drone or even human error, navigating the drone towards an unseen obstacle), the drone will fully autonomously avoid that obstacle using waypoint navigation and once an object is no longer detected, it will continue towards the destination like normal.

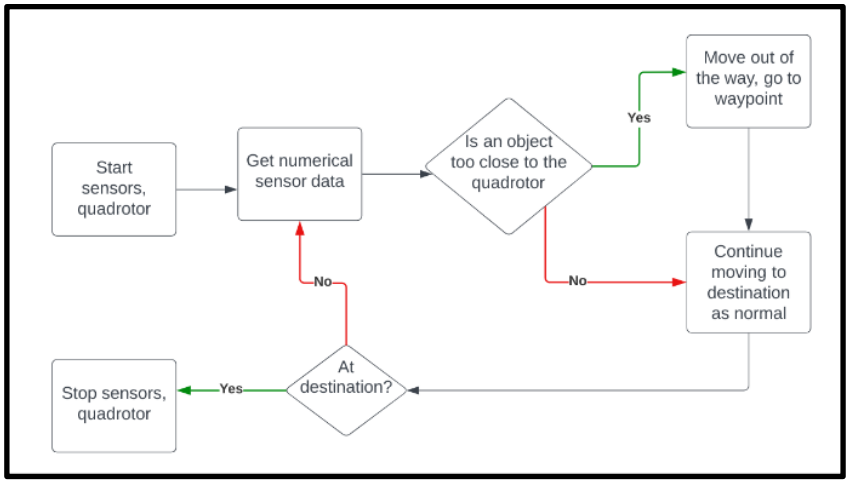


Figure 1. High-level Logic of System

To detect objects in the environment, four blind-spot Lidar sensors were utilized on the front, left, right and rear of the drone. The sensors were physically mounted to a 3D tower attached to the drone, and connected to our PCB which then filtered and sent the data from our sensors (as well as power the sensors) to our microcontroller. Once the microcontroller received the data, our software could then analyze the data and perform obstacle detection and avoidance.

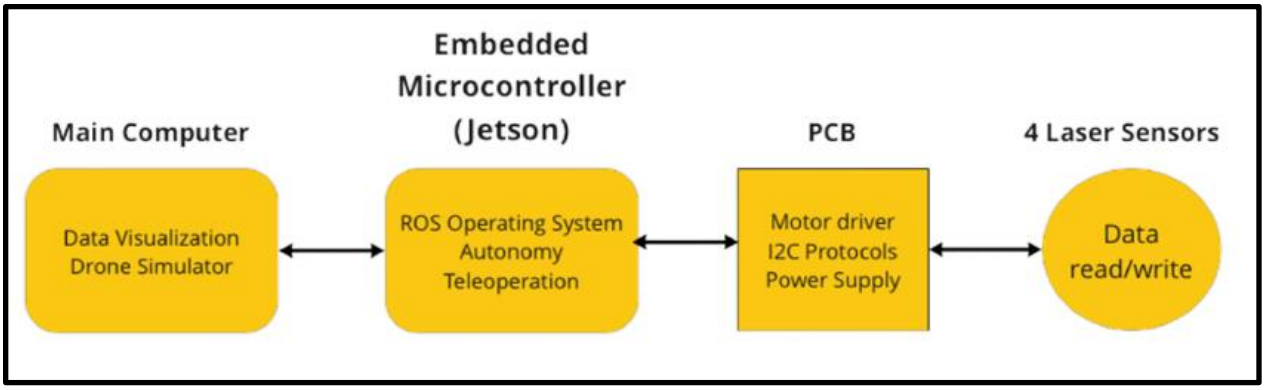


Figure 2. Connection of System Components

The specific technical implementation is as follows. For the sensors, each came with a yellow backboard that has a 9 pin I2C/UART output. We used the I2C protocol (SDA and SCL pins), and have physical connectors running from each of the sensors to headers on our PCB. Once the signals reach our PCB, we implemented a I2C bus using an I2C multiplexer chip that physically assigns an address to each of the four sensors, and then outputs one I2C signal to our microcontroller that will be used on the software side for autonomous obstacle detection and avoidance.

A servo motor was also implemented underneath the front sensor, which continuously swings back and forth (total view of 60 degrees) to increase the field of view of the front sensor. To implement this, our microcontroller (Jetson) outputs a PWM signal based on our desired duty cycle for a 60 degree field of view (with the sensor operating at 240 reading per second), and that PWM signal is sent to our PCB via a 40 pin ribbon cable, and then connected to Pin 1 of the motor's header through an onboard-trace.

To power all components, a Lipo battery was utilized. Our goal was to make the design modular so it can be added to a non-autonomous drone and just plug and play, thus we are using a separate power source for our modular design, which will only power the PCB and 4 sensors and motor attached to it. Our microcontroller (Jetson) and the rest of the drone are powered from a different battery source, external to our design. Each sensor and the motor operate at 5V, and the total current requirements equal 1.7 A (and that is the max current requirement for all 4 sensors and the motor, as well as the I2C multiplexer chip which is only a few microamps). Our 11.1V battery can sustain a maximum continuous load of 0.85A*45C = about 38 A, which is well above our current requirements. The final draw on the battery may be higher than the maximum draw listed for each of the components, so that overhead is good for our design. The flight time of the drone we are using (the Holybro X500 drone) is 15 min, which our battery can sufficiently power. We also decided on using a step-down switching regulator to output the 5V, 2A over a linear regulator because a linear regulator would produce too much heat and a heat sink would be required. Switching regulators are also highly efficient and available as modular chips which are compact and reliable. The regulator was implemented externally to our PCB and the resulting 5V signal was sent to our PCB via an XT30 power cable.

The PCB, which incorporates the four sensors, motor and power supply, is detailed next. The schematic diagram and board layouts are shown below. The PCB is four layers, with two signal layers, a power plane, and a ground plane. The logic of the PCB is very straightforward. A 2-pin XT30 header is utilized to connect the 5V power supply to our PCB, and apply a 5V signal to the power plane. 9-pin headers allow physical connection from each of the sensors to the PCB. The relevant pins are the SDA and SCL pins, which are needed for I2C communication. The 5V and GND pins are used to power the sensors. Each of the sensors' I2C outputs (SDA and SCL) are connected to an I2C multiplexer chip, which gives a physical hardware address to each of the sensors (which have the same physical address out of the box). The chip then creates an I2C bus,

which can be used to differentiate between each of the slave devices (sensors) and read the I2C data. The I2C data is connected to the SDA and SCL pins on the 40-pin header used to connect to our microcontroller (Jetson). The 40-pin header for the microcontroller also includes 5V and GND pins for logic level signals, as well as a GPIO pin for the PWM signal coming from the microcontroller, which controls the speed and orientation of the servo motor through the 3-pin header used to physically connect the motor to the PCB. The 3-pin header has the PWM signal pin, as well as a 5V and GND pin to power the motor. Jumpers to the 5V supply were included for the SDA and SCL pins on the I2C multiplexer chip in case a pull-up is needed, in which a shunt can be used to create the electrical connection. Jumpers to the 5V supply on each of the three hardware selectable addresses on the I2C chip (A0, A1, A2) can be used with shunts to change the address of the chip itself, effectively changing the address of the I2C bus in case more than 8 sensors are used and other future applications where multiple I2C chips are needed to be differentiated. Due to the large amount of EMF interference on the drone caused by WIFI and GPS signals being transmitted back and forth from the tele operational joystick, a chassis was added, made up of a conductive material and ground, around the PCB to serve as a Faraday cage. A Faraday cage is a conductive enclosure that is used to block out external electric fields. This creates a conductive barrier around the PCB, which helps to shield it from external electric fields. The result is that the electric fields are diverted around the chassis and do not affect the components on the PCB. The PCB size is 75mm x 90mm in order to fit onto the top plate of the drone, within the 3D printed tower walls surrounding it.

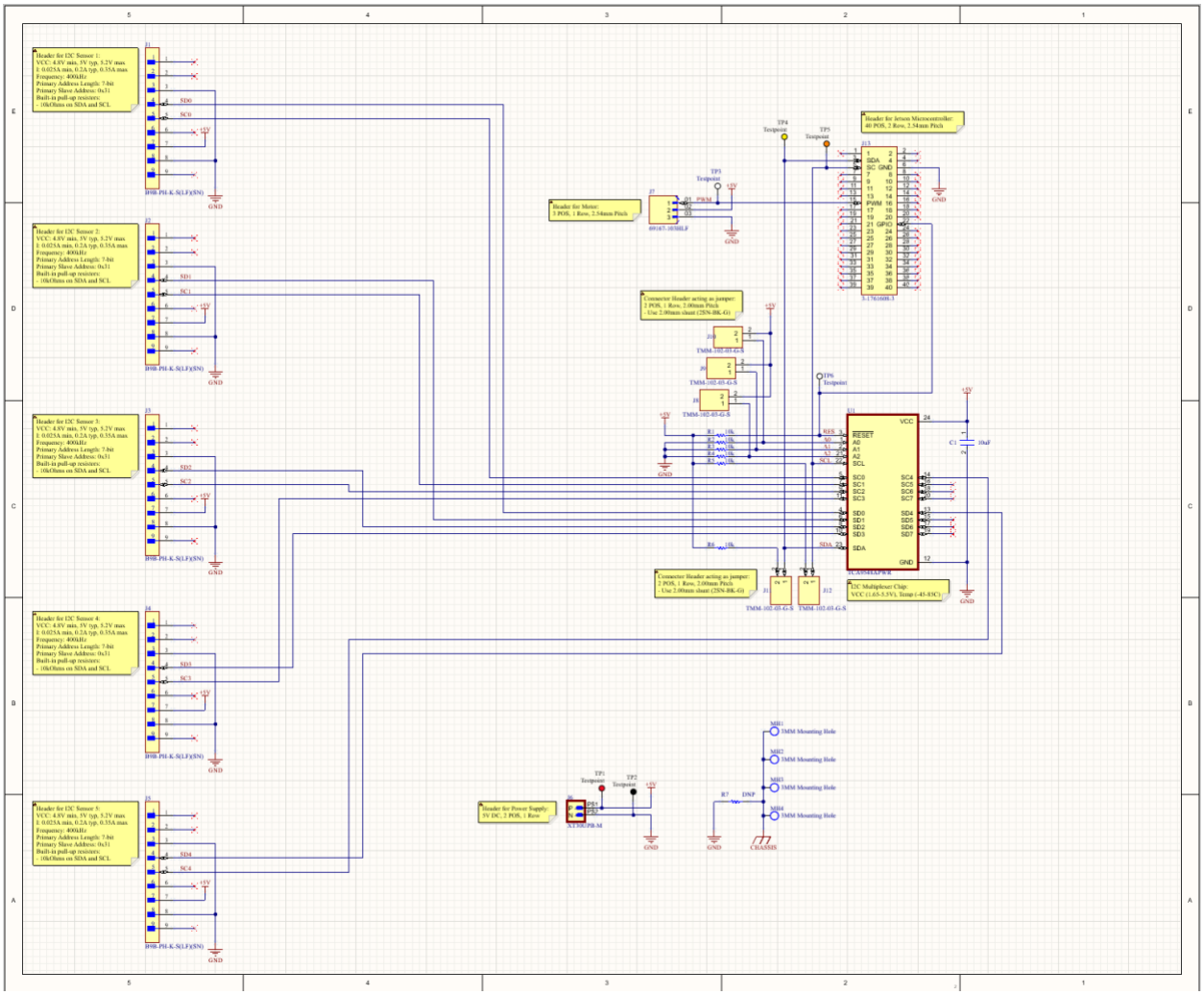


Figure 3. Schematic Diagram

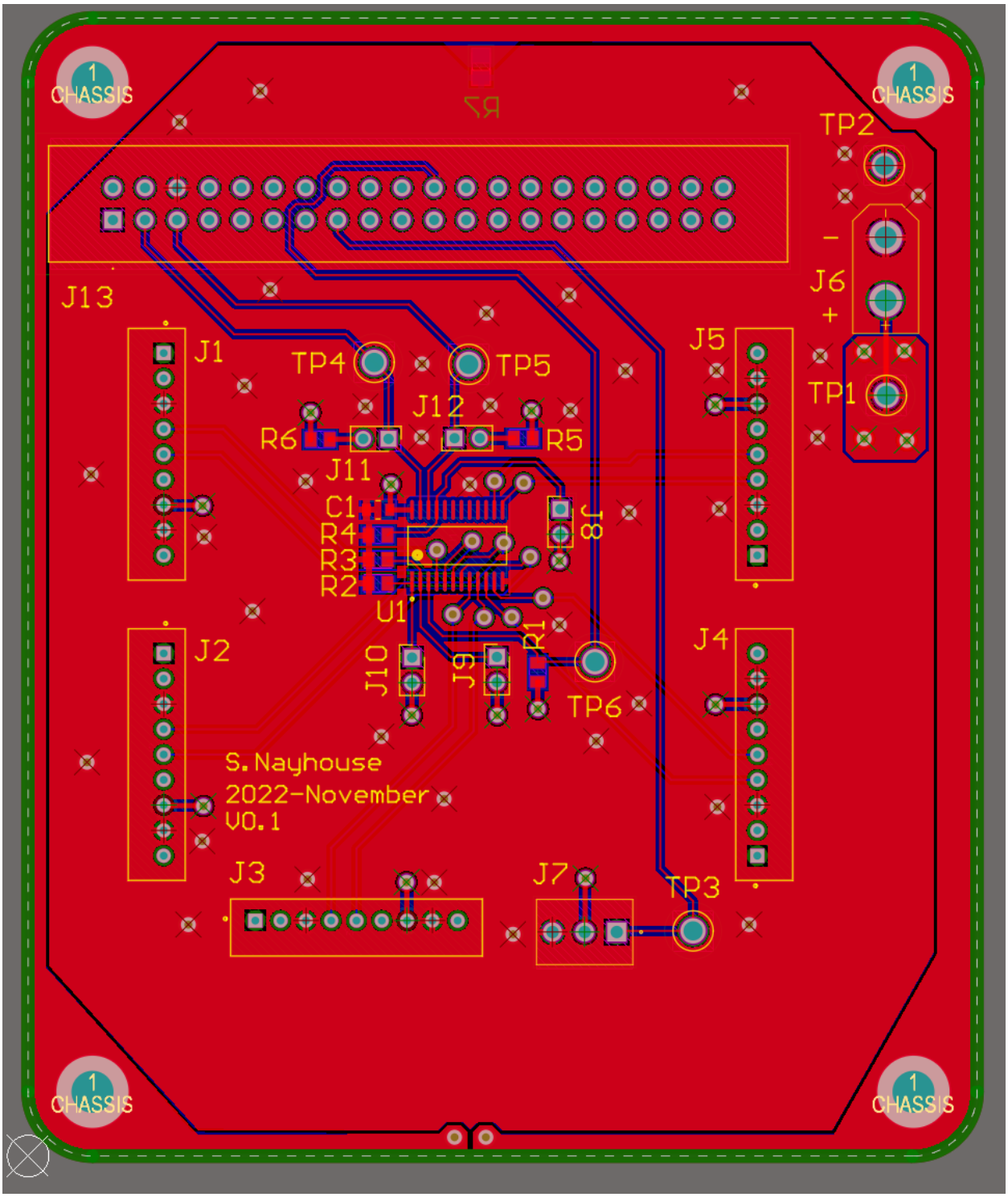


Figure 4. Top layer of board (2D)

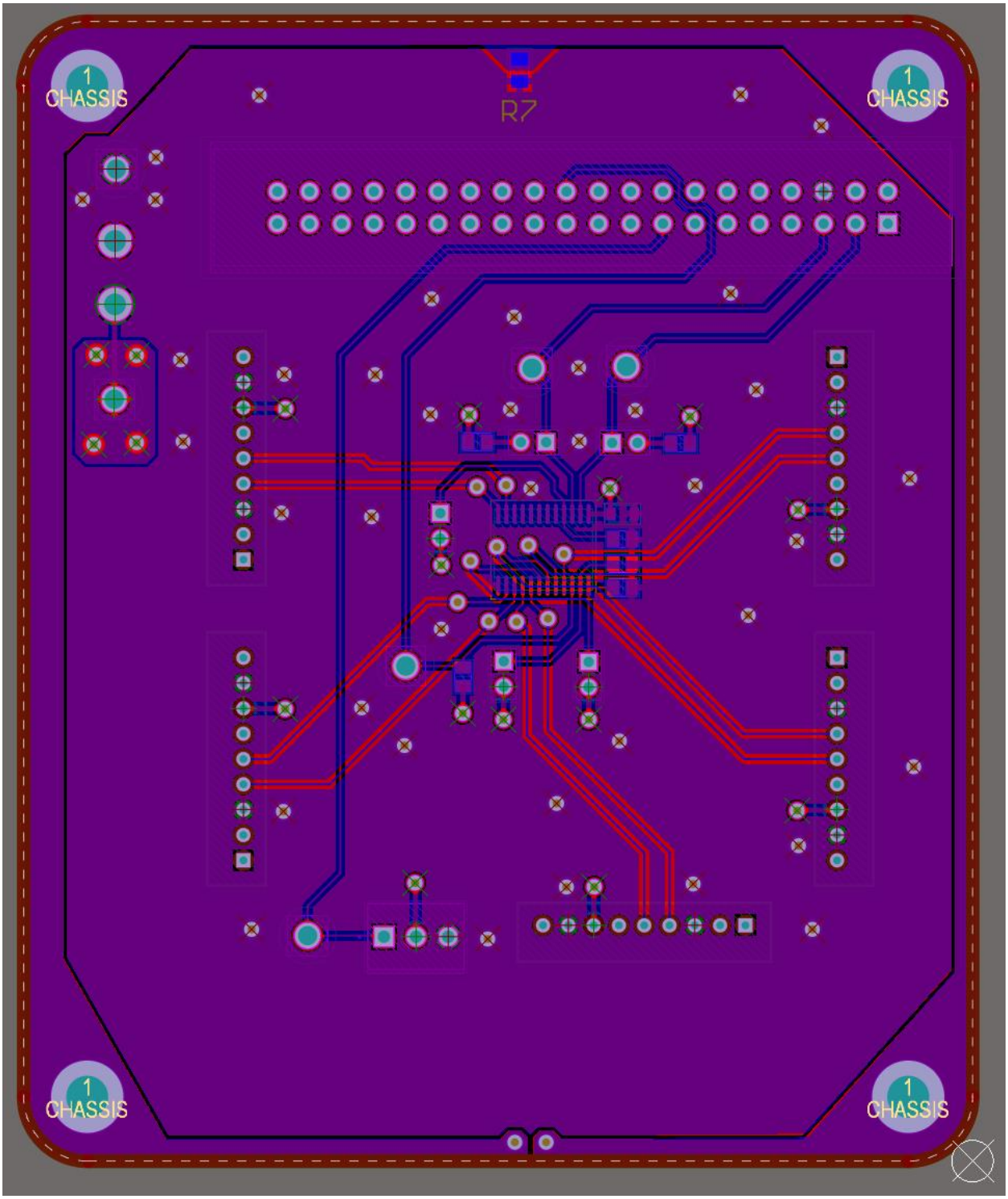


Figure 5. Bottom layer of board (2D)

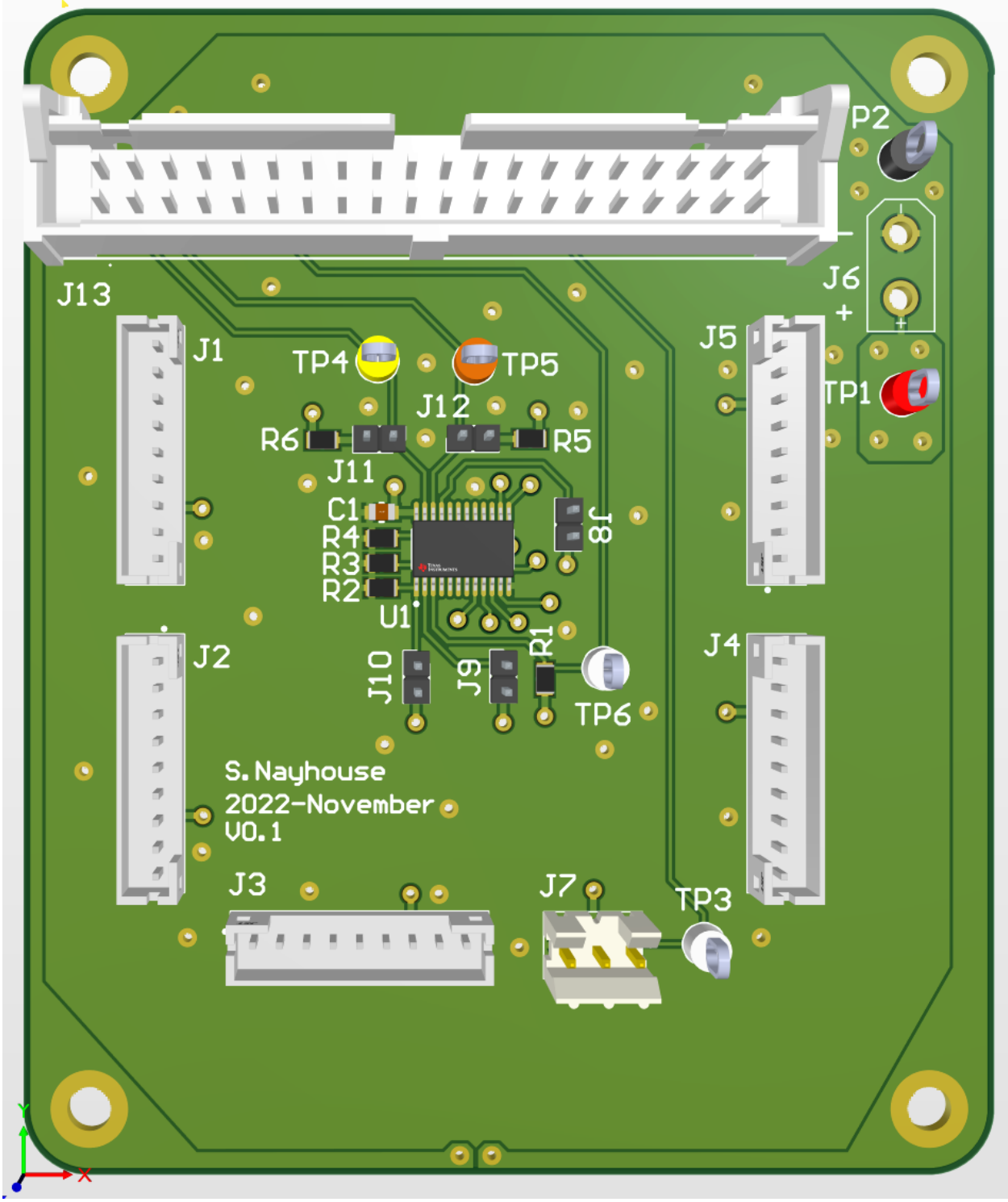


Figure 6. Top layer of board (3D)

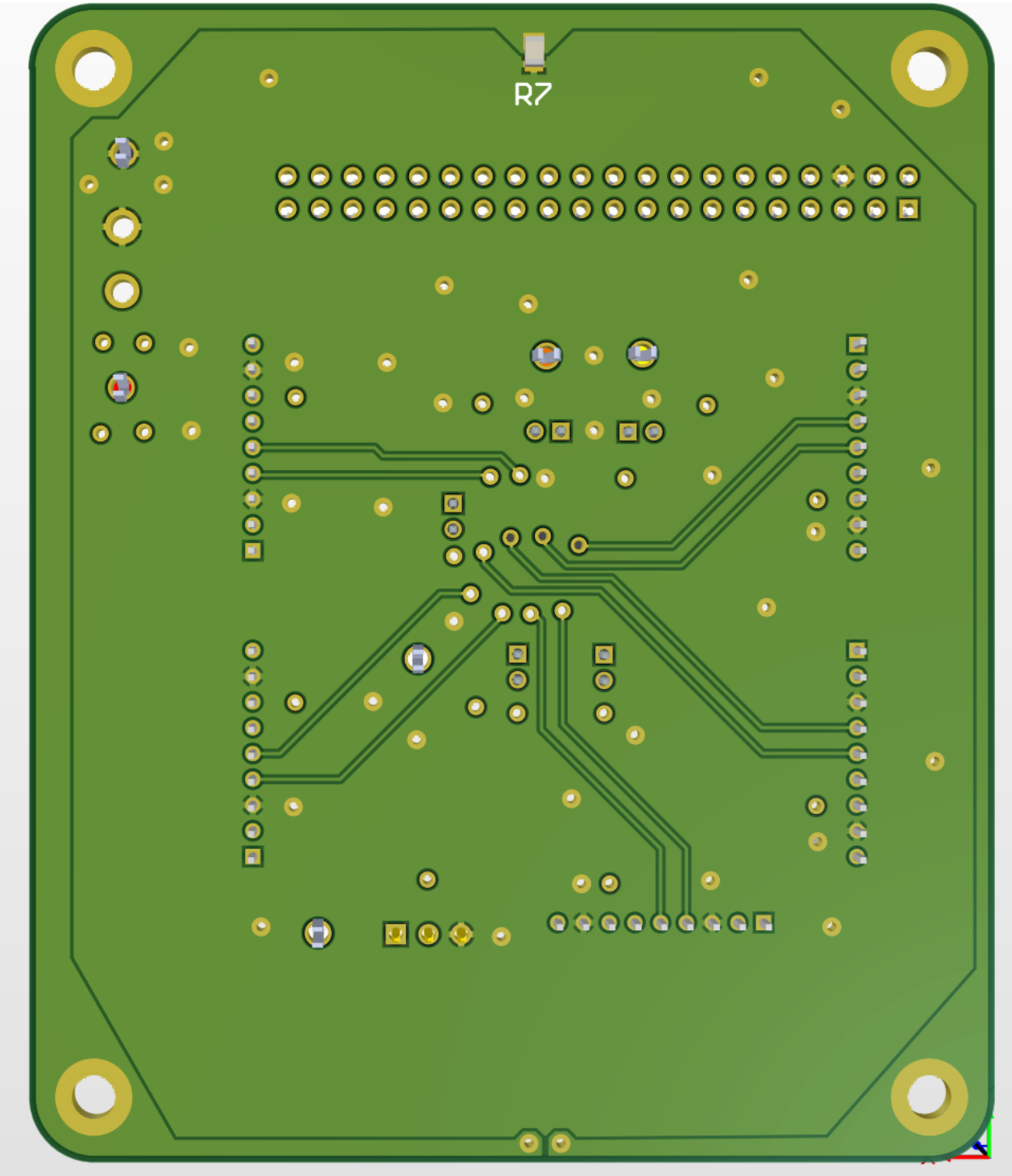
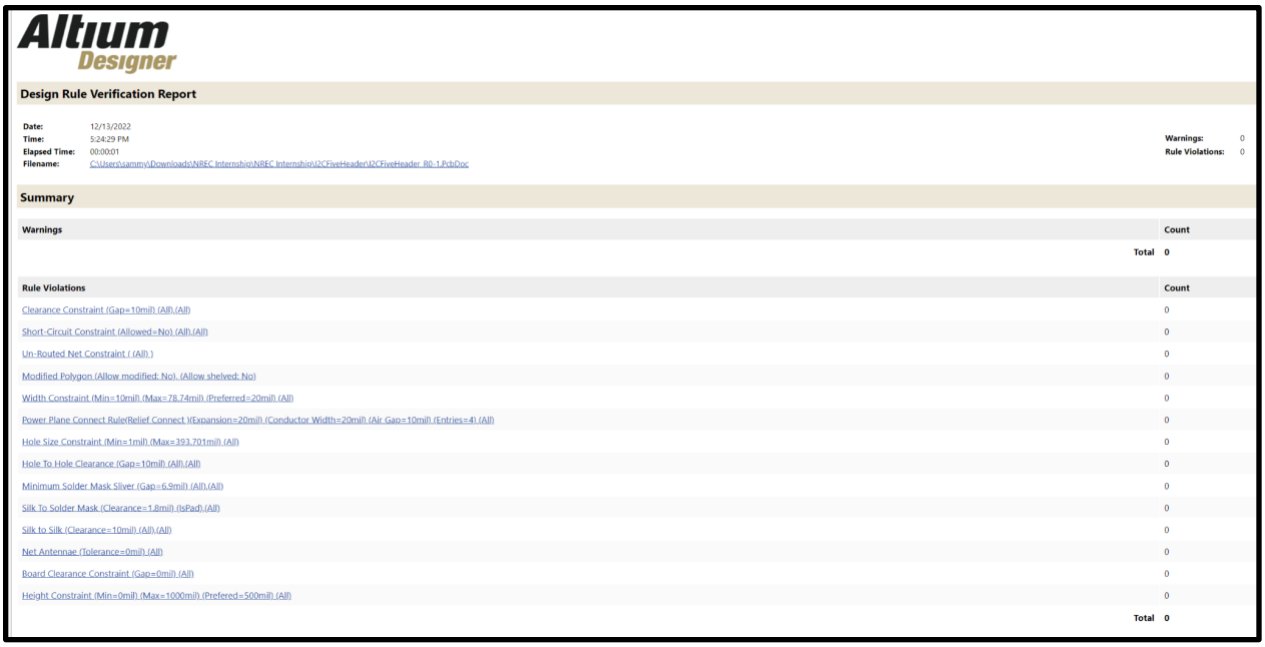


Figure 7. Bottom layer of board (3D)



Altium Designer

Design Rule Verification Report

Date:	12/13/2022	**Warnings:**	0
Time:	5:24:29 PM	**Rule Violations:**	0
Elapsed Time:	00:00:01		
Filename:	C:\Users\sammy\Downloads\NREC Internship\NREC Internship\I2CFiveHeader\I2CFiveHeader_R0-1.PcbDoc		

Summary

Warnings	Count
Total	0

Rule Violations	Count
Clearance Constraint (Gap=10mil) (All),(All)	0
Short-Circuit Constraint (Allowed=No) (All),(All)	0
Un-Routed Net Constraint ((All))	0
Modified Polygon (Allow modified: No), (Allow shelved: No)	0
Width Constraint (Min=10mil) (Max=78.74mil) (Preferred=20mil) (All)	0
Power Plane Connect Rule(Relief Connect)(Expansion=20mil) (Conductor Width=20mil) (Air Gap=10mil) (Entries=4) (All)	0
Hole Size Constraint (Min=1mil) (Max=393.701mil) (All)	0
Hole To Hole Clearance (Gap=10mil) (All),(All)	0
Minimum Solder Mask Sliver (Gap=6.9mil) (All),(All)	0
Silk To Solder Mask (Clearance=1.8mil) (IsPad) (All)	0
Silk to Silk (Clearance=10mil) (All),(All)	0
Net Antennae (Tolerance=0mil) (All)	0
Board Clearance Constraint (Gap=0mil) (All)	0
Height Constraint (Min=0mil) (Max=1000mil) (Prefered=500mil) (All)	0
Total	0

Figure 8. Design rule check showing no errors

Figure 9. FreeDFM report showing no errors

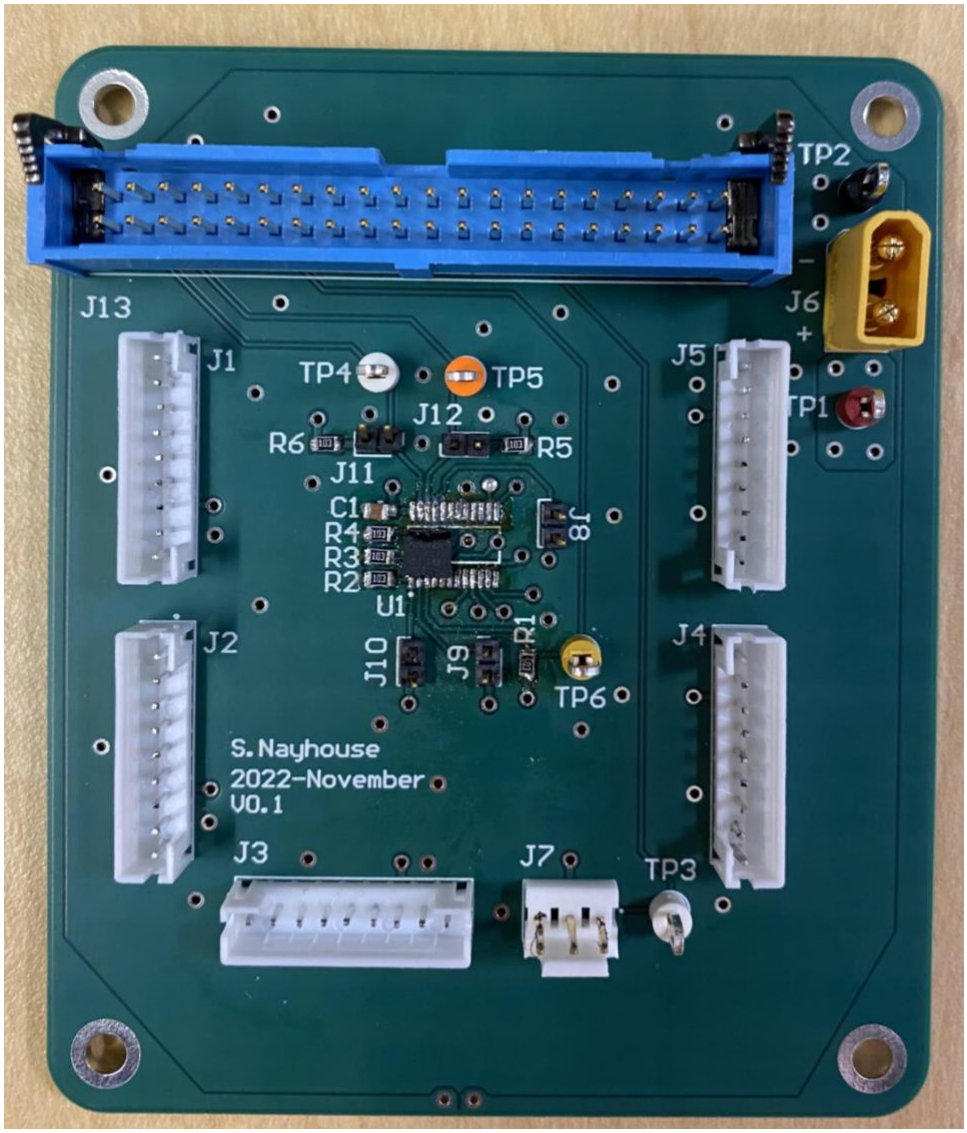


Figure 10. Picture of fabricated and assembled PCB

Each of the physical components (sensors, motor, power supply, PCB, microcontroller, drone, and 3D tower) and associated components are described in detail below.

Designator	Quantity	Manufacturer	Manufacturer Part Number	Supplier	Supplier Part Number	Mounting Type	Operating Temperature	Value
C1	1	TDK Corporation	C2012X7R1A106K125AC	Digi-Key	445-6857-1-ND	Surface Mount, MLCC	-55°C ~ 125°C	10uF
J1, J2, J3, J4, J5	5	JST Sales America Inc.	B9B-PH-K-S(LF)(SN)	Digi-Key	455-1711-ND	Through Hole	-25°C ~ 85°C	
J6	1	Amass	XT30UPB-M			Through Hole	-20°C ~ 120°C	
J7	1	Amphenol ICC (FCI)	69167-103HLF	Digi-Key	609-2410-ND	Through Hole	-40°C ~ 105°C	
J8, J9, J10, J11, J12	5	Samtec Inc.	TMM-102-03-G-S	Digi-Key	SAM10191-ND	Through Hole	-55°C ~ 125°C	
J13	1	TE Connectivity AMP Connectors	3-1761608-3	Digi-Key	A107237-ND	Through Hole	-65°C ~ 105°C	
R1, R2, R3, R4, R5, R6	6	Panasonic	ERA-6AEB103V	Digi-Key	P10KDACT-ND	Surface Mount	-55°C ~ 155°C	10kOhms
R7	1			Digi-Key				DNP
TP1	1	Keystone	5010	Digi-Key	36-5010-ND	Through Hole	-40°C ~ 185°C	Red
TP2	1	Keystone	5011	Digi-Key	36-5011-ND	Through Hole	-40°C ~ 185°C	Black
TP3, TP6	2	Keystone	5012	Digi-Key	36-5012-ND	Through Hole	-40°C ~ 185°C	White
TP4	1	Keystone	5014	Digi-Key	36-5014-ND	Through Hole	-40°C ~ 185°C	Yellow
TP5	1	Keystone	5013	Digi-Key	36-5013-ND	Through Hole	-40°C ~ 185°C	Orange
U1	1	Texas Instruments	TCA9548APWR	Digi-Key	296-34905-1-ND	Surface Mount	-40°C ~ 85°C	

Table 1. PCB components

Product	**Specification: Value**
Holybro X500 V2 ARF Kit (SKU30125) (Link) *Drone*	Wheelbase: 500mm Motor mount pattern: 16x16mm Frame Body: 144x144mm, 2mm thick Landing gear height: 215mm Space between top and bottom plates: 28mm Weight: 610g
HRB 4S Lipo Battery 14.8V 5000mAh *Battery*	Material: Li-Polymer Cells: 4S, Voltage: 14.8v Capacity: 5000mAh Discharge Rate: 50C Burst Rate: 100C Size(L*W*H): 6.10*1.89*1.26in / 155*48*32mm (0 - 3mm difference) Weight: 1.08lb / 17.35oz / 492g (± 2g)
PM07 12s Power Module *Power Management/Distribution*	Current: Total 120A Outputs (MAX) UBEC 5V Output Current: 3A UBEC Input Voltage: Total 120A Outputs (MAX) Dimensions: 68*50*10 mm Mounting Holes: 45*45mm Weight: 40.3g
Terabee TeraRanger Evo 15m *Laser Imaging, Detection and Ranging (LIDAR) Sensor*	Range: 0.5m to 15m Rate: 240 readings/sec Field of View: 2° Supply Voltage: 5V DC +/-5% Supply Current: 90mA-330mA Interfaces: USB 2.0 Micro-B, UART, I2C Dimensions: 29x29x22mm (sensor + backboard)
Parallax 360 Degree High Speed Continuous Rotation Servo *Servo Motor*	Peak stall torque at 6 V: 2.2 kg-cm (30.5 oz-in) RPM: +/-120 w/feedback control, 140 max (+/- 10) at 6 V, no load Voltage requirements: 6 VDC typical, 5–8.4 VDC max range Current Requirements: 15 mA (+/- 10) idle, 150 mA (+/- 40) no-load, 1200 mA stalled Control Signal: PWM, 3–5 V 50 Hz, 1280–1720 µs Feedback Signal: PWM, 3.3V, 910 Hz, 2.7–97.1% duty cycle Dimensions: 2.15 x 1.46 x 0.79 in (50.4 x 37.2 x 20 mm)
NVIDIA Jetson AGX Orin Developer Kit *Embedded Microcontroller*	GPU: NVIDIA Ampere architecture with 2048 NVIDIA® CUDA® cores and 64 Tensor cores CPU: 12-core Arm Cortex-A78AE v8.2 64-bit CPU 3MB L2 + 6MB L3 Memory: 32GB 256-bit LPDDR5 204.8GB/s Storage: 64GB eMMC 5.1 Power: 15W-60W M.2 Key M: x4 PCIe Gen 4 M.2 Key E: x1 PCIe Gen 4, USB 2.0, UART, I2S USB Type-C: 2x USB 3.2 Gen2 USB Type-A: 2x USB 3.2 Gen2, 2x USB 3.2 Gen1 USB Micro-B: USB 2.0 Dimensions: 110mm x 110mm x 71.65mm

Table 2. System components

For the software, we started by installing the Gazebo simulator, which was used to test if our software for the sensor readings would work properly for a real aerial vehicle. We then used

a script to takeoff the drone in the simulator since the simulator starts with the drone on the ground plane. A code snippet from it is shown in Figure 11 below.

```
if __name__ == "__main__":
    #initialize node
    rospy.init_node("offb_node_py")
    #set up subscriber, publisher nodes (read data from subscriber, write to publisher)
    state_sub = rospy.Subscriber("mavros/state", State, callback = state_cb)
    local_pos_pub = rospy.Publisher("mavros/setpoint_position/local", PoseStamped, queue_size=10)
    #wait for quadcopter to arm
    rospy.wait_for_service("/mavros/cmd/arming")
    arming_client = rospy.ServiceProxy("mavros/cmd/arming", CommandBool)
    #Wait for the set mode
    rospy.wait_for_service("/mavros/set_mode")
    set_mode_client = rospy.ServiceProxy("mavros/set_mode", SetMode)

    # Setpoint publishing MUST be faster than 2Hz
    rate = rospy.Rate(20)#Rate = 20 Hz

    # Wait for Flight Controller connection
    while(not rospy.is_shutdown() and not current_state.connected):
        rate.sleep()

    pose = PoseStamped() #get current position
    pose.pose.position.x = 0
    pose.pose.position.y = 0
    pose.pose.position.z = 2
```

Figure 11. Snippet of code from takeoff_land program setting drone position to [0,0,2]

The basic idea of this script is that it first engages offboard mode and arms the vehicle, which should allow the drone to take off and go from position [0,0,0] to position [0,0,2]. By the time the script is in its main loop, the drone should be airborne with the same x and y position as before, with that position being constantly published to the simulator via the Pose Stamped message from geometry_msgs in ROS [18]. Once the program is halted, the drone will safely land back to position [0,0,0], the vehicle will be disarmed, and offboard mode should be disengaged.

Next, 4 separate python scripts will be run to read values from the four Terabee EvoRanger sensors and publish them to ROS topics. An example snippet from the program for the front sensor is shown in Figure 12 below.

```
#need to read sensors continuously w/o rospy.is_shutdown
while not rospy.is_shutdown():
    sensors = Float32() #topic to be published to is a Float32 variable
    sensors.data = 0.0 #to be published to topic, will get updated later
    try:
        #get distance detected by sensor
        x0 = get_evo_range(evo)

        #If waiting for a frame header, temporarily publish speed as 0
        try:
            float(x0)
            sensors.data = x0
        except:
            rospy.logwarn("Error: "+str(x0)+" isn't a float")


    except serial.serialutil.SerialException:
        try:
            #open next available port (will be the ACM[0+4])
            evo = openEvo("/dev/ttyACM4")

        except:
            rospy.logerr("Device disconnected (or multiple access on port). Exiting...")

    #publish sensor vals to Ros Topic
    pub.publish(sensors)
```

Figure 12. Main while loop of front sensor script

In this script, the sensor ports are initialized along with the node that the script should publish to. Once these are all initialized and the port that the sensor is being connected to for a specific script, the main while loop begins. In this loop, the sensor range that is detected at that specific moment is read in and if that value is successfully read in from the port and is a valid float value, it will be published to the ROS topic for that sensor (for example, the sensor value read in from the front sensor program would publish to a topic called tera_readingFront). If the value read in from the sensor is not a valid float, a value of 0 will be published to the topic and a warning will print out stating that the sensor value is not a float. If the port disconnects so that it can no longer can be read from, the program will try to open the next available port, which for us happened to be the port 4 spaces ahead of the initial port (ACM0 would be ACM4, ACM1 would be ACM5, etc.). If this also failed, the program would print out an error message stating that the port was disconnected. In either case, 0 would be published to the topic since it is important to have some data constantly published to the topic.

The programs for the left, right, and back sensors are very similar to the front sensor program. The only difference comes down to the specific port names (ACM0, ACM1, ACM2, and ACM3 for the front, left, back, and right sensors respectively) and the publisher nodes that

each of the programs output to (tera_readingFront, tera_readingLeft, tera_readingBack, and tera_readingRight).

The last program that we made was uav_quadJoy20.py, which took in the sensor readings published by the sensor programs and utilized those to autonomously control a drone. The first part of the program involved making a class, ROSVars, that would contain the published values from all four sensors and the joystick input. Figure 13 displays a code snippet from this part of a program below.

```
class ROSVars:
    def __init__(self):
        self.velx=0 #forward/backward velocity
        self.vely=0 #left/right velocity
        self.xButton=0 #button input (used to overr
        self.sensors=[0,0,0,0] #array for sensor da

    #get joystick data
    def joyCallback(self,data):
        self.velx = data.axes[0] #+1 = right, -1 =
        self.vely = data.axes[1] #+1 = forward, -1
        self.xButton = data.buttons[0] #1 = pressed

    #get front sensor data
    def sensorCallbackFront(self,data):
        self.sensors[0] = str(data).split()[1] #ext


    #get left sensor data
    def sensorCallbackLeft(self,data):
        #rospy.loginfo(data)
        self.sensors[1] = str(data).split()[1]

    #get back sensor data
    def sensorCallbackBack(self,data):
        self.sensors[2] = str(data).split()[1]
```

Figure 13. ROSVars class which contains sensor readings and joystick inputs

All four sensor readings are stored in one array, sensors, that is continuously updating, even if the main while loop for this program is interrupted. In addition the readings from the left joystick and a button from the controller are read in and updated continuously.

In the main loop all of the subscriber nodes that the program is reading from are initialized along with the node for the program itself, the ROSVars object, the node to publish the drone velocity to (Twist), and the rate of the main while loop, which was set to 1000 since that would allow it to operate as fast as possible. As for the subscriber nodes that the joy program drew from, they included each of the sensors used along with the node that took in the joystick readings. The Twist [19] object in ROS is also initialized. This can be seen in Figure 14 below.

```
def main():
    #set up node to publish to quadcopter
    vel_pub = rospy.Publisher('/mavros/setpoint_velocity/cmd_vel_unstamped', Twist, queue_size=1)
    RosVars = ROSVars()
    #Subscriber nodes to supply joystick input/sensor data
    rospy.Subscriber("/joy", Joy, RosVars.joyCallback)
    rospy.Subscriber("/tera_readingFront", Float32, RosVars.sensorCallbackFront, queue_size=100)
    rospy.Subscriber("/tera_readingLeft", Float32, RosVars.sensorCallbackLeft, queue_size=100)
    rospy.Subscriber("/tera_readingBack", Float32, RosVars.sensorCallbackBack, queue_size=100)
    rospy.Subscriber("/tera_readingRight", Float32, RosVars.sensorCallbackRight, queue_size=100)

    #Initialize this node
    rospy.init_node('uav_joy', anonymous=True)
    #Less rate = slower, more rate = more noise
    rate = rospy.Rate(1000)

    #Twist object to publish to quadcopter
    vel = Twist()
```

Figure 14. Start of the main loop in uav_quadJoy20.py that contains the initialization of the nodes, the rate, the ROSVars object, and the Twist object that will publish the drone velocity

After this, the main while loop begins. Initially, the loop starts with setting the x and y velocity equal to the value of the joystick input from the controller multiplied by a constant variable representing the speed multiplier (this was initialized before the while loop). Then if the button input from the joystick isn't pressed, then the program should loop through all four sensor readings. This button input for overriding the sensors was included as a fail-safe in case one of the sensors had unexpected behavior or there was a software bug that would otherwise cause the drone to crash. This code is shown in figure 15 below.

```
vel.linear.x = -speed*RosVars.velx #speed * left joystick input (horizontal) m/s
vel.linear.y = speed*RosVars.vely #speed * left joystick input (vertical) m/s
vel.linear.z = 0 #shouldn't move in z direction

#if button not pressed, drone movement based on sensor data
if(RosVars.xButton == 0):

    #get sensor values
    for i in range(len(RosVars.sensors)):

        ################################################################################
        '''
        Legend


        [1.0, 0.0] : Moving Right
        [-1.0, 0.0]: Moving Left
        [0.0, 1.0]: Moving Forwards
        [0.0, -1.0]: Moving Backwards

        Sensor 0 = Front
        Sensor 1 = Left
        Sensor 2 = Back
        Sensor 3 = Right

        All velocities in meters/second

        '''
```

Figure 15. Start of the while loop in uav_quadJoy20.py that contains the initialization of the drone velocities and a table illustrating joystick directions

Within the for loop, a sensor value is read in from the sensor array and if it is less than 1 meter, the program determines which sensor has the reading. This is done by determining which index in the sensor array the value is located since the array contains readings for the front, left, back, and right sensors respectively. Depending on which sensor the <1.0 meter reading is for, the drone may either stop completely (for the front sensor), be forced to move in the direction opposite of which sensor detected the object (if the left sensor detected the object, the drone would be forced to move right), or may be only have limited movement in 1 direction (as is the case with the back sensor). All of these actions are done by modifying the x and y velocities to either be close to 0 (for it to stop) or 0.5 times the speed variable in the opposite direction for the x velocity for the right and left sensors. In the case where the drone is forced to move in the direction opposite of the object that is detected (for the left and right), this represents the drone autonomously avoiding an object. This code is shown in Figure 16 below.

```
sen0 = float(RosVars.sensors[i])
if(sen0 < 1.0):
    #switched x and ys from what they originally were
    if(i == 0):
        #don't move forward
        vel.linear.x = 0.001
        vel.linear.y = 0.001
        break

    elif(i == 1):
        #don't move left
        vel.linear.y = 0.001
        vel.linear.x = speed*0.5

    elif(i == 2):
        if(vel.linear.y > 0.001):
            vel.linear.y = 0.001

    elif(i == 3):
        #don't move right
        #check if an object is also detected to the left
        if(vel.linear.x == speed*abs(0.5)):
            #if so, stop
            vel.linear.x = 0.001
        else:
            vel.linear.x = -speed*abs(0.5)
        vel.linear.y = 0.001
```

Figure 16. Code snippet from uav_quadJoy20.py illustrating how object detection from a sensor affects drone velocity

Once out of the for loop for the sensors and the if statement for the buttons, the program outputs messages based on its current operation. For example, if the drone has its x and y velocities set to 0.001 (close to 0), the program will print out a message stating “stop”, meaning that the drone has been completely stopped. Another example is that if the drone is moving at a speed equal to half the speed constant in the positive direction, the program will print “moving

right" since it is forced to be moving right to move out of the way of an obstacle. If the movement of the drone isn't forced to the left/right, isn't stopped, and isn't being overridden by the button on the controller being pressed, the program will print out the message "keep going", meaning that the drone movement is largely unhindered by the sensors. At the end, the velocities are published to the drone and rate.sleep is called so that the main while loop can be run again. Figure 17 has this code snippet below.

```
    #if button pressed
    if(RosVars.xButton == 1):
        rospy.loginfo("Overridden")
    #If obstable detected in front or (left and right)
    elif(vel.linear.x == 0.001 and vel.linear.y == 0.001):
        rospy.loginfo("stop")
    #If obstacle detected on left
    elif(vel.linear.y == speed*abs(0.5)):
        rospy.loginfo("moving right")
    #If obstacle detected on right
    elif(vel.linear.y == -speed*abs(0.5)):
        rospy.loginfo("moving left")
    #No obstacle detected, override button not pressed
    else:
        rospy.loginfo("keep going")

    #publish velocities to quadcopter
    vel_pub.publish(vel)
    #allows for infinite loop (as long as ROS is active)
    rate.sleep()
```

Figure 17. Code snippet from uav_quadJoy20.py illustrating code output before drone velocities are published.

The entire software process is illustrated by an FSM shown in Figure 18.

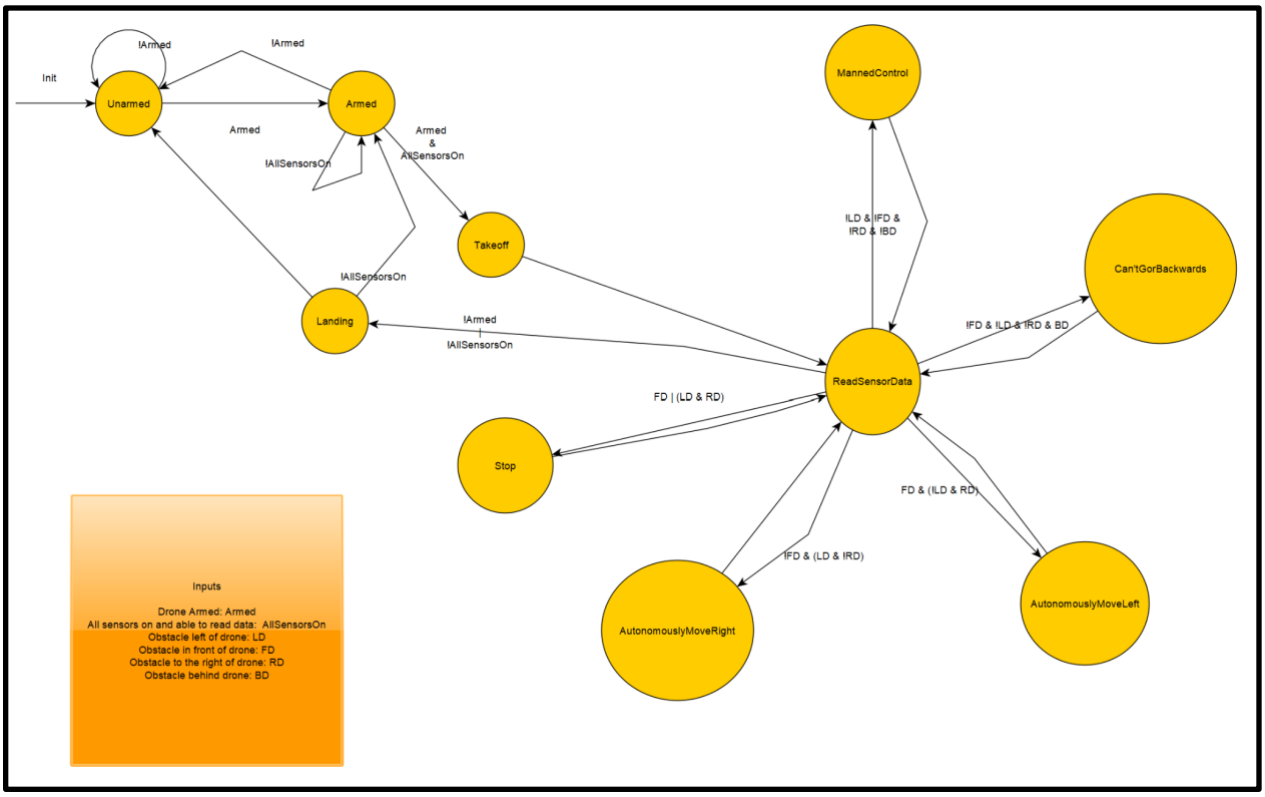


Figure 18: Finite State Machine illustrating software behavior

The physical implementation of our design was brought together using a 3D printed figure. This 3D design was prepared using FreeCAD, then sliced and printed using Cura slicing software and the Ultimaker S3 3D printer. The design held a hollow pentagonal prism structure, leaving space on the front to mount the LiDAR sensors facing forward, right , left and backwards. The shape allowed for the PCB to sit inside the prism, with the 40 pin ribbon cable feeding out a rectangular cut out in the back of the shape, and the micro-usb to I^2C cables feeding through a cylindrical cut out in the top level of the structure to connect to the sensors. This structure and components sat on top of a carbon fiber plate that is screwed into the second level of the drones top layer.

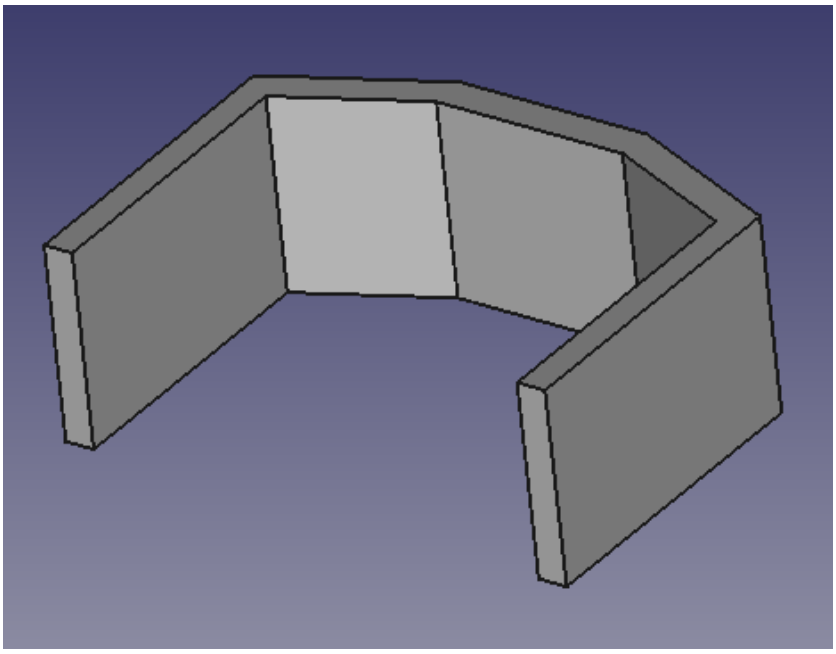


Figure 19. Front Wall of 3D design

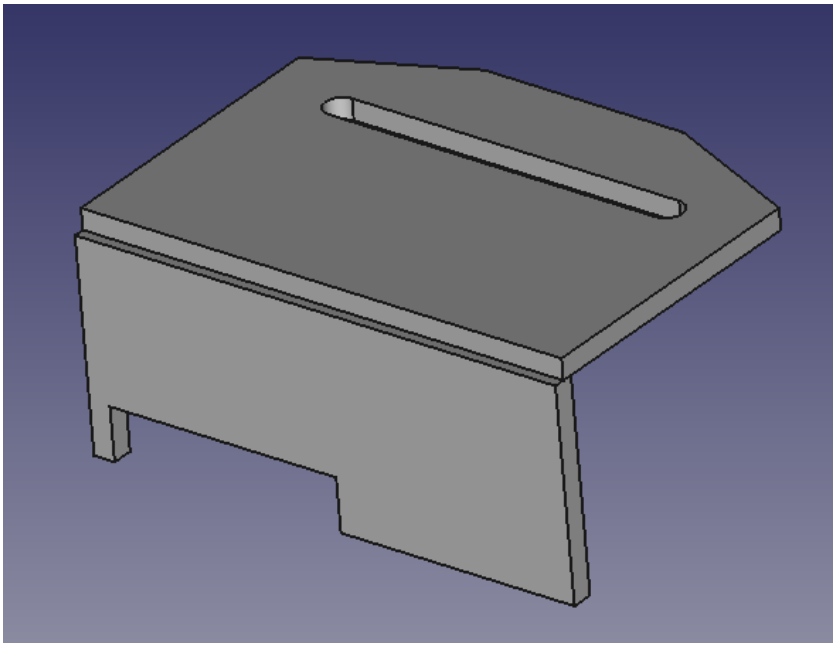


Figure 20. Back Wall and Cover of 3D Design

Project Time Line

Original Gantt Chart

In the original timeline for our project, we were very optimistic in how quickly we could complete each task and eventually come to a final deliverable. A large majority of our tasks were serial in terms of hardware and software. This allowed for us to progress very quickly in accomplishing tasks.

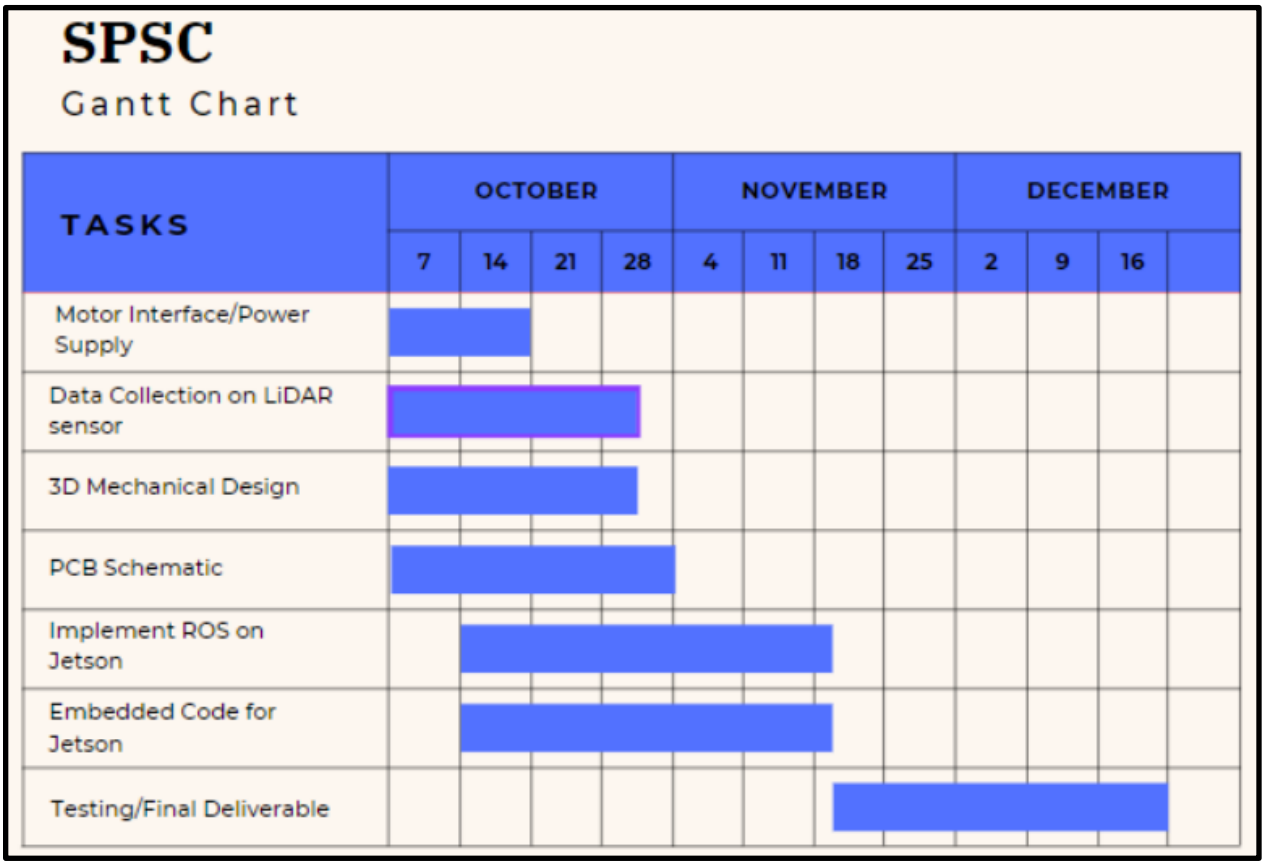


Figure 21. Original Gantt Chart

Revised Gantt Chart

Our revised Gantt chart below shows a more realistic timeline of the project. Some of the dates have been shifted back and there is also a more detailed list of tasks provided. The previous timeline did not accurately reflect how difficult some tasks would be and the time needed to complete them.

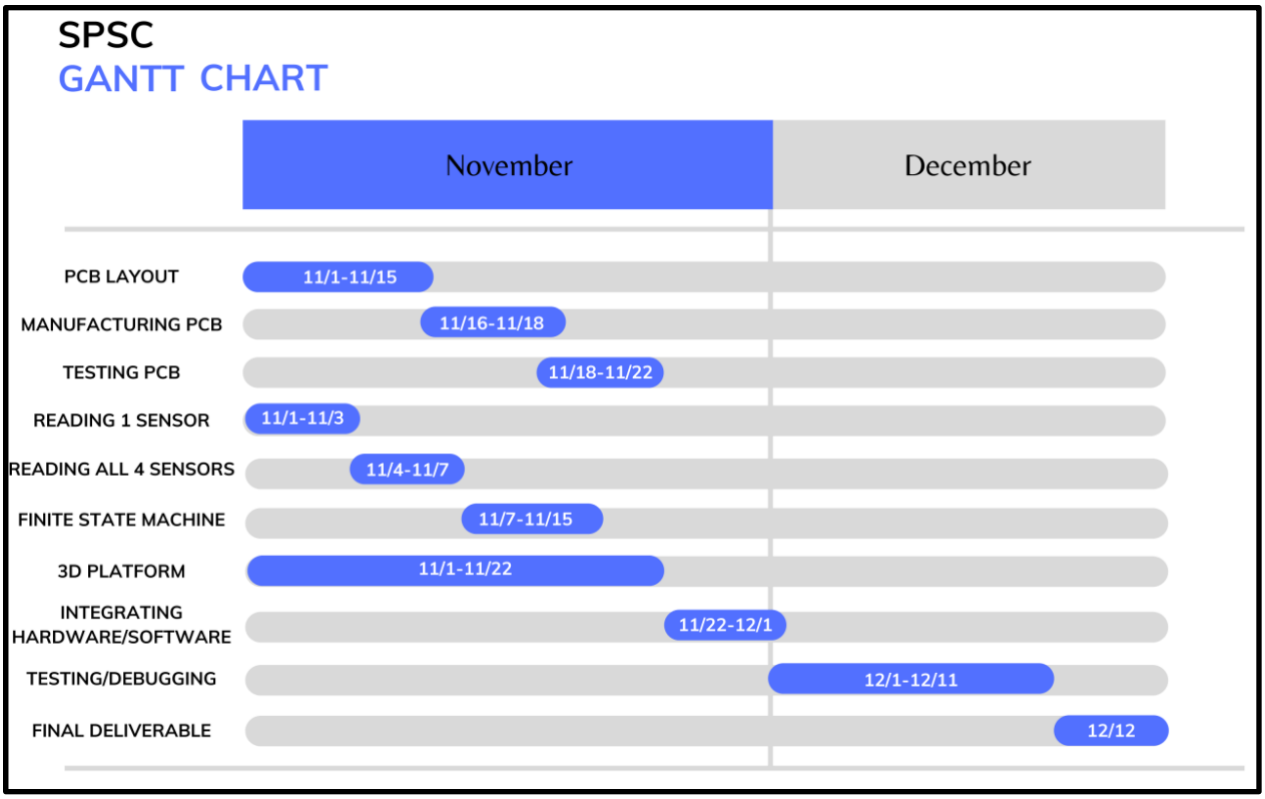


Figure 22. Revised Gantt Chart

Test Plan

The main purpose for our test plan is to provide a systematic and organized approach to ensure that it is functioning correctly and meets the specified design requirements. Our test plan outlines the specific tests that were performed, the equipment and tools that were used, the criteria for passing or failing each test, and the procedures for recording and reporting the test results. This is an important part of the overall design and manufacturing process, as it helps to ensure that the project is of high quality and meets the desired specifications.

In terms of testing the PCB, the following methodology was utilized:

- Visual inspection: We examined the PCB by eye to look for any obvious defects, such as broken traces, missing components, or other issues.
- Continuity testing: For this aspect a multimeter was utilized to check for electrical continuity between different points on the PCB. This was crucial in helping identify any broken traces in the circuit.
- Power-on testing: Here we applied power to the PCB and used a multimeter to measure the voltage level at the power and ground test points. We successfully measured 5V relative to GND.

- Functional testing: We tested the PCB as part of the final product to ensure that it is functioning correctly for our application. We tested the PCB under a variety of different conditions and environments to ensure that it performs as expected, including with the drone off, drone on and not flying, and the drone on and flying. The software received the I2C signals from the PCB with delays within our allowed specified range.

In terms of testing the sensors and obstacle detection and avoidance, the following methodology was utilized:

- Sensor readings: We simulated sensor readings with 1 sensor first, ensuring the data is read in correctly and the Gazebo simulator visualizing the quadrotor halt when an object is detected within 2-5 meters in front of the sensor. We then expanded the previous criteria to all 4 sensors and connected the indicated sensors to the servo motors, testing again for functionality of the sensor data.
- Drone obstacle avoidance: In order to test this we ran the simulator with someone covering one of the sensors and observing what direction the drone would move when flying with this action.
    1. We first texted that we could get the drone in the simulator to be controlled with a joystick
    2. We then tested if the drone would stop based on the reading from 1 sensor regardless of joystick input
    3. We then expanded this to 4 sensors, with each sensor representing a different direction
    4. Next we tested the drone and four sensors to see if we could eliminate any noticeable delay between the sensor readings and when the drone stops based on those readings
    5. We then changed what the drone would do when the sensors detected an object. Instead of not being able to move in one direction, the drone would either stop completely (for the front sensor) or move away from the detected object (for the left and right sensors)
    6. Once the above worked on a simulator, we connected the Jetson with our computer and made sure 1 sensor could be hooked up to the Jetson and transmit data to the computer
    7. We expanded this to include 4 sensors that could be connected to the Jetson
    8. Once we got 4 sensors from the Jetson to transmit data to our computer, we ran the software on a real drone and tested it to see if it could detect objects and move based on that

Final Results

Tasks
Device accurately takes in field of view with LIDAR sensors
Device always starts up with battery supply
PCB sends correct data from I2C to embedded microcontroller (Jetson)
Data can be simulated with LIDAR Data Visualization (RViz) and Drone Simulator (Gazebo)
Drone will perform obstacle avoidance and not allow movement towards detected (i.e., if an object is detected in front of done, even if the human operator commands the drone to move forward, it will not move towards the object
3D printed board accommodates physical components

Table 3. Project Tasks

Letter Grade	Number of Tasks Completed
A+	6 Tasks
A	5 Tasks
B	4 Tasks
C	3 Tasks
D	0-2 Tasks

Table 4. Letter Grade Based Upon Number of Tasks Completed

For the task of “Data can be simulated with LIDAR Data Visualization (RViz) and Drone Simulator (Gazebo)”, we had one of the sensor programs running and we published the topic that the program was being published to. For instance, when we ran our script for the first sensor (uav_sensor0_ROS.py), if we ran rostopic echo /tera_readingFront, sensor values from the front sensor would continuously print out and would be responsive to an obstacle moving in front of it.

For the task of "Drone will perform obstacle avoidance and not allow movement towards detected (i.e., if an object is detected in front of done, even if the human operator commands the drone to move forward, it will not move towards the object", we accomplished this by having the simulator up and running and having someone cover the left sensor, which caused the drone to move to the right regardless of the joystick input provided. Similarly, when the right sensor was covered, the drone in the simulator would move left regardless of the joystick input. This can be shown in Figure 23 and 24 below.

Figure 23. Blocking the right sensor causes the drone to move to the left regardless of joystick input

Figure 24. Blocking the left sensor causes the drone to move to the right regardless of joystick input

For the task of "3D printed board accommodates physical components", we designed prototypes using paper and other materials at first, mapping out how we wanted the structure to

sit on the drone before printing. Following this we designed structures in FreeCAD to visualize the prototypes. There were several CAD designs printed throughout the project, each time we attempted to mount the sensors on the structure and found issues with field of view, we edited the design and printed again in order to get the most efficient design.

Costs

The following shows the cost of each component of our PCB.

Line #	Designator	Quantity	Manufacturer	Manufacturer Part Number	Supplier	Supplier Part Number	Price/Unit	Total Price
1	C1	1	TDK Corporation	C2012X7R1A106K125AC	Digi-Key	445-6857-1-ND	0.33	0.33
2	J1, J2, J3, J4, J5	5	JST Sales America Inc.	B9B-PH-K-S(LF)(SN)	Digi-Key	455-1711-ND	0.48	2.4
3	J6	1	Amass	XT30UPB-M			0.48	0.48
4	J7	1	Amphenol ICC (FCI)	69167-103HLF	Digi-Key	609-2410-ND	0.99	0.99
5	J8, J9, J10, J11, J12	5	Samtec Inc.	TMM-102-03-G-S	Digi-Key	SAM10191-ND	0.65	3.25
6	J13	1	TE Connectivity AMP Connectors	3-1761608-3	Digi-Key	A107237-ND	11.62	11.62
7	R1, R2, R3, R4, R5, R6	6	Panasonic	ERA-6AEB103V	Digi-Key	P10KDACT-ND	0.36	2.16
8	R7	1			Digi-Key		0.1	0.1
9	TP1	1	Keystone	5010	Digi-Key	36-5010-ND	0.1	0.1
10	TP2	1	Keystone	5011	Digi-Key	36-5011-ND	0.42	0.42
11	TP3, TP6	2	Keystone	5012	Digi-Key	36-5012-ND	0.42	0.84
12	TP4	1	Keystone	5014	Digi-Key	36-5014-ND	0.42	0.42
13	TP5	1	Keystone	5013	Digi-Key	36-5013-ND	0.42	0.42
14	U1	1	Texas Instruments	TCA9548APWR	Digi-Key	296-34905-1-ND	2.09	2.09

							Total Cost	$ 25.62

Table 5: PCB Parts and Costs

The standard price to fabricate a 4-layer PCB with the number of holes, traces, pad, etc. that we utilized in our design is about $274.38. Thus, the total cost of one PCB is about $300.

Part	**Cost for 1 Unit**	**Cost for 10,000 Units**
Holybro X500 V2 ARF KIT	$117.00	$1,170,000
HRB 4S Lipo Battery 14.8V 5000mAh	$49.95	$499,500
PM07 12s Power Module	$42.00	$420,000
Terabee TeraRanger Evo 15m	$64.00	$640,000
Parallax 360 Degree High Speed Continuous Rotation Servo	$49.90	$499,000
NVIDIA Jetson AGX Orin Developer Kit	$1,999	$19,990,000
PCB Costs	$300	#3,000,000
Total	$2,621.85	$26,218,500

Table 6: Drone with Obstacle Avoidance Module Cost

Future Work

If a group were to expand upon our design, there are several concerns to account for in order to speed up implementation and avoid roadblocks. On the software side of the implementation, we had lingering issues with delays and timing for the intake and printing of sensor values. A way to avoid such setbacks would be to publish each sensor implementation as separate rostopics, rather than all being published to one rostopic. Also, in connection with the 3D CAD design, we would recommend accounting for 3D implementation when making changes to software, as we ran into issues with making small tweaks to fit our software, printing several 3D designs, each taking at least 3 hours to complete. The 3D design was also affected by changes in hardware, as we would recommend accounting for where to feed wires, place new components (sensors and PCB), and encase parts as needed. Each change to hardware requires several tweaks to the 3D design.

When developing and updating the PCB, it would be good practice, and avoid many setbacks, to think about how and where wires from sensors will be connected to the PCB, for example, our first iteration of the PCB had difficulties connecting to sensors because headers were in weird locations. Our second iteration took into account location of headers and wires, leaving our design in a better position to expand upon.

Finally, with our high budget, we ended up using many parts that are high quality but scarce in supply, making some difficult to order. Focusing on parts that are good quality and have a higher supply would help avoid issues with ordering and running into problems with replacing parts if some were to break.

References

[1] “ROS Documentation.” Open Robotics, May 20, 2022. Accessed: Sep. 12, 2022. [Online]. Available: http://wiki.ros.org/

[2] N. O. and A. A. US Department of Commerce, “What is LIDAR.” https://oceanservice.noaa.gov/facts/Lidar.html (accessed Sep. 27, 2022).

[3] “What is Inter-IC (I2C)? - Definition from Techopedia,” Techopedia.com. http://www.techopedia.com/definition/319/inter-ic-i2c (accessed Sep. 27, 2022).

[4] “User Manual for TeraRanger Evo single point distance sensors and backboards.” Terabee, 2021. Accessed: Sep. 12, 2022. [Online]. Available: https://terabee.b-cdn.net/wp-content

[5] “Product Description for Smraza 10 Pcs SG90 9G Micro Servo Metal Geared Motor Kit for RC Robot Arm/Hand/Control with Cable, Mini Servos for Arduino Project (10).” Maxmartt. Accessed: Sep. 12, 2022. [Online]. Available: https://www.amazon.com/Maxmartt-Stepper-Linear-Engraving-Machine/dp/B087Q8B38R/ref=asc_df_B087Q8B38R/?tag=hyprod-20&linkCode=df0&hvadid=532923056828&hvpos=&hvnetw=g&hvrand=9802371421801855586&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9008337&hvtargid=pla-1456359630889&psc=1

[6] “Product Description for DC 5V-9V 12V 2 Phase 4 Wire Stepper Motor Linear Rail 90mm Stroke Lead Screw Linear Stage Actuator with Nut Slider Step Angle 18 Degree for DIY Laser Engraving Machine.” QINZX. Accessed: Sep. 12, 2022. [Online]. Available: https://www.amazon.com/Stepper-Linear-Actuator-EngravingMachine/dp/B09BZDSY7V/ref=pd_lpo_3?pd_rd_i=B09BZDSY7V&psc=1

[7] “Understanding the IEEE 802.11 Standard for Wireless Networks,” Juniper Networks, Oct. 2018. Accessed: Sep. 12, 2022. [Online]. Available: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-80211.html

[8] “The Environmental Impact of Lithium Batteries,” IER, Nov. 12, 2020. https://www.instituteforenergyresearch.org/renewable/the-environmental-impact-oflithium-batteries/ (accessed Sep. 27, 2022).

[9] Carlton Reid, “Semi-Autonomous Cars Hit Cyclist In 5 Out Of 15 Test Runs, Finds AAA,” Forbes Magazine, May 2022, Accessed: Sep. 12, 2022. [Online]. Available: https://www.forbes.com/sites/carltonreid/2022/05/16/semi-autonomous-car-hitcyclist-in-5-out-of-15-test-runs-finds-aaa/?sh=5c991fdc3d03

[10] “Understanding the IEEE 802.11 Standard for Wireless Networks,” Juniper Networks, Oct. 2018. Accessed: Sep. 12, 2022. [Online]. Available: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director4.0/topics/concept/wireless-80211.html

[11] “Drones and Sustainability: Protecting The Environment.” https://www.unsustainablemagazine.com/drones-and-sustainability-5-ways/ (accessed Sep. 27, 2022).

[12] WilliamWoodall, “rviz - ROS Wiki,” ROS.org, May 16, 2018. https://wiki.ros.org/rviz (accessed Sep. 25, 2022).

[13] “Drones, the great ally of sustainability.” https://www.activesustainability.com/sustainable-development/drones-the-great-ally-ofsustainability/ (accessed Sep. 27, 2022). Page 13 of 13

[14] “14 CFR Part 107 -- Small Unmanned Aircraft Systems,” Code of Federal Regulations. https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-107 (accessed Sep. 25, 2022).

[15] P. M. Kanade, C. W. S. III, and J. B. GEHLHAAR, “Unmanned aerial vehicle obstacle detection and avoidance,” US10019907B2, Jul. 10, 2018 Accessed: Dec. 13, 2022. [Online]. Available: https://patents.google.com/patent/US10019907B2/en?q=obstacle+detection+drones&oq=obstacle+detection+for+drones

[16] 伊利亚·布雷瓦兹, “Method for training heterogeneous sensing system and heterogeneous sensing system,” CN108052097B, Apr. 09, 2021 Accessed: Dec. 13, 2022. [Online]. Available: https://patents.google.com/patent/CN108052097B/en?q=obstacle+detection+drones&oq=obstacle+detection+for+drones

[17] P. L. Guarnizo, G. Michalke, and T. Michalke, “Method and control device for identifying a potential collision between an unmanned aerial vehicle and an object,” WO2017097596A2, Jun. 15, 2017 Accessed: Dec. 13, 2022. [Online]. Available: https://patents.google.com/patent/WO2017097596A2/en?q=obstacle+avoidance+drones&oq=obstacle+avoidance+for+drones

[18] “geometry_msgs/PoseStamped Documentation,” *ROS.org*, Mar. 02, 2022. http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/PoseStamped.html (accessed Dec. 13, 2022).

[19] “geometry_msgs/Twist Documentation,” *ROS.org*, Mar. 02, 2022. http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Twist.html (accessed Dec. 13, 2022)