KENNESAW STATE UNIVERSITY

2020


# Product/Market Design Report

Product: Owl-Eye: Espy-Drone
Document #: CPE4800-MDR-Ver5.4
Creation Date:2/17/2020
File: MDR-PDR.DOCX
Authors: Erick Cueva, Joel J. Morel, Tyler Davison, Marquis Guy

[CPE 4800 Kennesaw State University]

# Table of Contents

# REVISION HISTORY

| *WHO* | DATE | REASONS FOR CHANGES | VERSION |
|---|---|---|---|
| *Joel* | 02/17/20 | Initial Release | Ver. 1.0 |
| *Erick* | 02/20/20 | Added use case scenarios | Ver. 1.2 |
| *Erick* | 03/03/20 | Project management | Ver. 2.0 |
| *Tyler/Marquis* | 03/04/20 | Building block 2 & 3 | Ver. 2.1 |
| *Everyone* | 03/12/20 | A lot of Formatting | Ver. 2.2 |
| *Everyone* | 03/12/20 | PDR revisions | Ver. 3.0 |
| *Everyone* | 03/19/20 | Updated Risk analysis, block description, blocks | Ver. 3.1 |
| *Erick* | 03/19/20 | Updated Project Professional Document | Ver. 3.2 |
| *Joel* | 03/19/20 | Updated Use case Scenarios and added phases | Ver. 3.3 |
| *Everyone* | 03/19/20 | Updated and Finalized Building Block | Ver. 3.4 |
| *Joel* | 03/26/20 | Edits from meeting with dr. ho | Ver. 4.0 |
| *Joel* | 04/06/20 | Reorganization | Ver. 4.1 |
| *Everyone* | 04/07/20 | Feedback adjustments | Ver. 4.2 |
| *Joel* | 04/09/20 | System Ecosystem | Ver. 4.3 |
| *Erick* | 04/09/20 | Building Blocks | Ver. 4.4 |
| *Everyone* | 04/10/20 | Project professional and block description | Ver. 4.5 |
| *Joel* | 05/01/20 | Risk analysis edits | Ver. 5.0 |
| *Erick and Tyler* | 05/02/20 | Executive Summery | Ver. 5.1 |
| *Erick* | 05/03/20 | Financial Data | Ver. 5.2 |
| *Joel* | 05/03/20 | Requirement Mapping | Ver. 5.3 |
| *Joel* | 05/03/20 | Captioning | Ver. 5.4 |
| *Everyone* | 11/03/20 | Added new sections | Ver. 6.0 |
| *Joel* | 12/06/20 | Milestones section | Ver. 6.1 |
| *Everyone* | 12/06/20 | Challenges section | Ver. 6.2 |

# 1. Executive Summary

## *1.1 Business Objective*

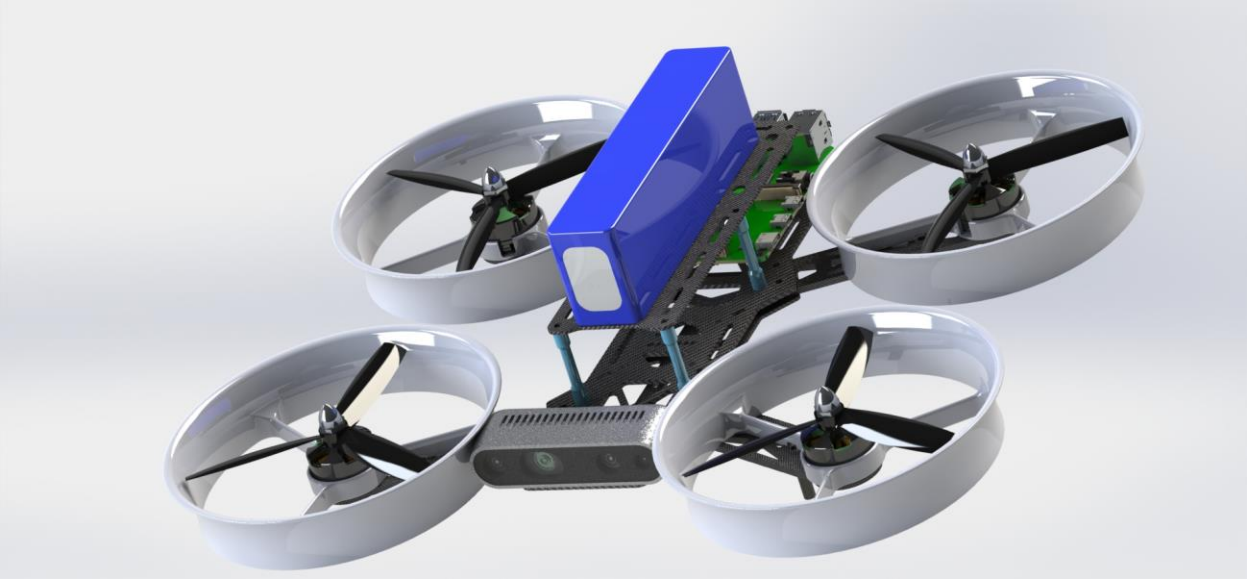


*Figure 1: Prototype Drone V1*

When we first set out on this project, our goal was to develop a durable, cost-efficient drone for use in mapping hazardous areas. We feel like we have achieved these goals in our design of the Owl-Eye: Espy-Drone. In order to keep the cost of our drone to a minimum, the KISS methodology was integrated within our development process and COTS were chosen for most of our components. Any required specialized components are designed to be 3d printed from affordable plastic materials and printers.

The main feature of the Owl-Eye is its ability to stream real-time camera data to a handheld device. A RGBD camera will allow or drone to wirelessly stream RGB and depth video data to a handheld device. Once it is received, the camera data, along with other telemetry, will be used to build a 3d representation of the drone's surroundings. This data will be viewable in real time, allowing for the operators to quickly respond to a recorded incident. If desired, the recorded data can also be uploaded to a cloud platform like AWS or Azure.

Two flight settings will be available for our drone in order to allow for a high level of compatibility with multiple use cases. One option will be manual flight with the use of a handheld terminal, and the other will be "hands free" autonomous flight. In order to allow for autonomous flight, multiple sensors and a Simultaneous Localization and Mapping (SLAM) algorithm will be used in conjunction with the main camera. This will allow the device to keep itself oriented in space and avoid any potential obstacles. As an additional safety feature, a temperature sensor will be used to keep track of ambient temperature. The data from this sensor will be used to warn the operator if the drone is in an area too warm to be used effectively.

## *1.2 Key Features and Attributes*

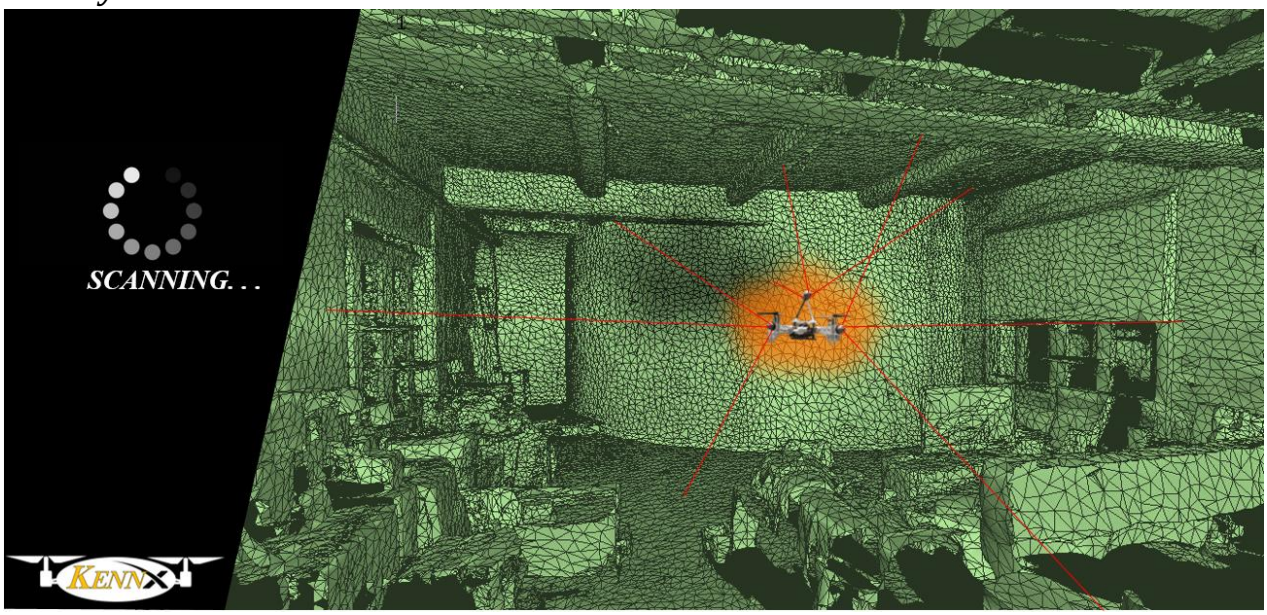



*Figure 2: Billboard Advertisement*

The Owl Eye: Intel Drone will come with the following features and attributes:

- RGBD 3D Camera for room mapping and model generation
- GPS and gyroscope modules that help provide drone location in relation to the ground station.
- Real-Time data transmission from the Raspberry Pi to the ground station.
- In the event of physical environmental damage, there is an autosave feature that will save all recent data collected, allowing the user to retrieve stored data.
- The drone has both manual and autonomous control modes that provided the user with the option for hands free flight.
- Object detection to avoid crashing.
- Safety landing procedure in case of low power.

## *1.3 Value and Benefits to Customers*

*Figure 3: Target Audience*

When in high-risk situations, like responding to a disaster or a military operation, protecting human life is the primary objective. Sadly, events like 9/11 led to 343 firefighter deaths when responding to the disaster. To save lives, firefighters must put themselves at risk within this unstable environment. When in other situations, like a military operation, the objective is to acquire intel within enemy lines. Soldiers must also put themselves at risk to acquire such information.

Our product provides a solution to these situations by significantly reducing certain risk factors. In the situation of firefighters responding to building wreckages, before they begin entering the building, they can deploy our drone that will begin maneuvering through building and mapping its surroundings. This allows firefighters to get a rough image of the inside before they enter to spot potential weaknesses. The drone could even facilitate in spotting hurt or trapped civilians. In the scenario of soldiers within enemy lines, soldiers could get an accurate depiction of the of the building they would be infiltrating before entering. This could provide crucial information about number of targets and whether is it worth entering. It could map out potential escape routes for hostages or even provide information for an extraction point. While these environments still pose a threat, our device significantly reduces the risk factors.

## *1.4 Team*

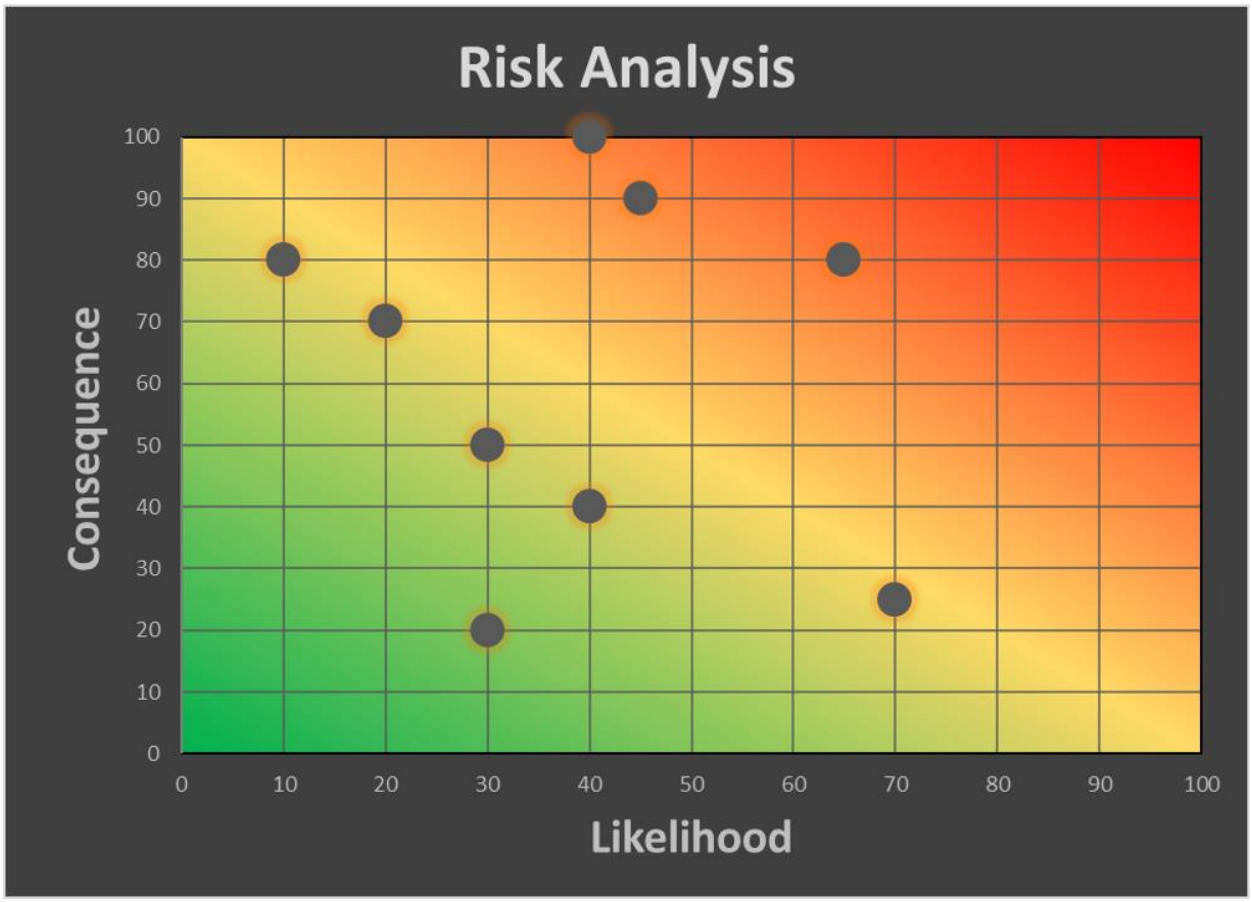
| *WHO* | *Field of Study* | *SKILL SETS* | *ROLE* |
|---|---|---|---|
| **Erick Cueva** | Computer Engineering | Managing/Organization, Hardware, and minor Programming. | Project Leader and Documentation |
| **Joel J Morel** | Computer and Mechatronics Engineering | Programming, Drones, 3D CAD, and Concept Manufacturing and Testing | Flight Engineer and Programmer |
| **Tyler Davison** | Computer Engineering | Machine vision, robotics programming, Embedded Linux | Software Architect |
| **Marquis** | Computer Engineering | Solidworks, Programming, Manufacturing | CAD Design and Physical Testing |

# 2. Risks and Mitigations

## *2.1 Risk Analysis*

| *RISK ID#* | *DESCRIPTION* | *LIKELIHOOD* | *SEVERITY* |
|---|---|---|---|
| RSK-001 | Failure of 3D Mapping software/hardware | 40% | 100% |
| RSK-002 | Autonomous Flight | 70% | 25% |
| RSK-003 | Flight failure | 65% | 80% |
| RSK-004 | Navigation Failure | 30% | 20% |
| RSK-005 | Realtime Communications | 45% | 90% |
| RSK-006 | Drone Structure | 10% | 80% |
| RSK-007 | Control Unit Integration | 30% | 50% |
| RSK-008 | Sensory System | 20% | 70% |
| RSK-009 | ROS Docker processing | 40% | 40% |

## *2.2 Risk Chart*



## *2.3 Mitigation Plan*

| *RISK ID#* | *Mitigation Plan* | *Action Plan* |
|---|---|---|
| **RSK-001** | This will be one of the first task to accomplish as acquiring the 3D mapping will be the proof of concept for the project | If 3D mapping is not acquirable than we will fall back to 2D or consider project failure and immediately jump to a backup project. |
| **RSK-002** | Manual flight will need to be accomplished first then the integration of autonomous would utilize AI Programs that could be utilize from online and reworked for our needs | If we do not acquire autonomous movement, we will resort to manual flight. |
| **RSK-003** | Obtain a working platform capable of flight and able to withstand double its weight | Remove least crucial parts, if not we resort to a terrestrial drone. |
| **RSK-004** | With the autonomous working we will utilize it from a rudimentary waypoint navigation. | If we cannot get the drone to return to home base after low battery (5%), we will aim to safely land the drone and wait for rescue |
| **RSK-005** | Ensure the connection to the Wi-Fi of both control system and control unit once accomplish establish a stable socket. | The is one of our biggest advertisements, not being able to do this will be close to product failure and we will resort to saving the information on the drone itself and assure a program to return to home base after satisfied mapping. |
| **RSK-006** | For the beginning stages we will utilize a carbon fiber frame to ensure that we will have a sturdy frame that can hold all the components | If a flight platform that can hold the components is not found, we will default to a terrestrial drone |
| **RSK-007** | We will conduct initial testing to first prove that the PixHawk and raspberry pi4 can communicate with each other | Should the PixHawk and pi be unable to communicate we will utilize a telemetry module directly to a laptop. |
| **RSK-008** | Once RSK-007 is resolved we will focus on being able to pull information from the PixHawk | We will utilize a telemetry module on the side connected to the ground station to obtain the sensory information |
| **RSK-009** | Ros will first be tested on the raspberry pi to ensure it can be implemented in the early stages | If RoS proves inconclusive we will use python threading to build the coding and communication for the drone. |

# 3. Financial Data

## *3.1 Preliminary Bill of Materials*

| Part | Part number | Quantity | Price |
|---|---|---|---|
| **Flight Platform** | | | |
| **4GB Raspberry Pi 4** | SB-PRT-008 | 1 | $60.00 |
| **Intel RealSense B415** | SB-PRT-010 | 1 | $150.00 |
| **PixHawk Mini Package** | | 1 | $125.00 |
| ---*3DR Pixhawk Mini* | SB-PRT-006 | 1 | --- |
| ---*Power Module* | SB-PRT-004 | 1 | --- |
| ---*GPS Module* | SB-PRT-007 | 1 | --- |
| **4S Lipo Battery** | SB-PRT-005 | 1 | $50.00 |
| **Inland PLA+** | SB-PRT-013 | 1 | $16.00 |
| **3 SPOT Lidar** | SB-PRT-009 | 1 | $42.00 |
| **4 Brushless Motor** | SB-PRT-002 | 1 | $34.00 |
| **12 5inch Props** | SB-PRT-014 | 1 | $15.00 |
| **Carbon Fiber Platform** | SB-PRT-001 | 1 | $25.00 |
| **4 Kiss ESC** | SB-PRT-003 | 1 | $100.00 |
| **Ground Station** | | | |
| Jetson Nano Tablet | SB-PRT-012 | 1 | $200.00 |
| Joy Con pair | SB-PRT-012 | 1 | $80.00 |
| Wi-Fi repeater | SB-PRT-011 | 1 | $40.00 |
| | | **Total** | $937.00 |

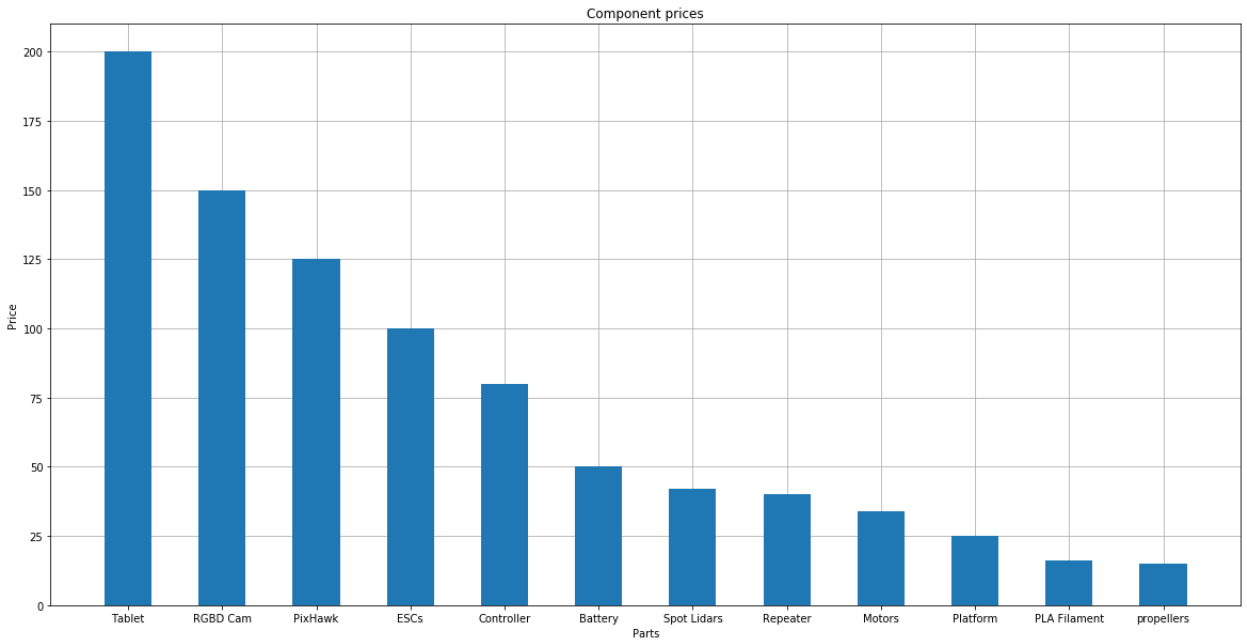



*Figure 4: Price Flow*

## *3.2 Cost of Goods Sold*

| Base + Play Area | Year 1 | Year 2 | Year 3 |
|---|---|---|---|
| Beginning Inventory | $1,000,000 | $1,500,000 | $2,000,000 |
| Ending Inventory | $937,000 | $1,405,500 | $1,874,000 |
| **COGS** | $63,000 | $94,500 | $126,000 |

## *3.3 Unit Price*

| | Consumer Price | Cost to Build |
|---|---|---|
| **Price per unit** | $1200.00 | $937 |

## *3.4 3-year ATAR Sales and Revenue Forecast*

| BASE + Play Area | Year 1 | Year 2 | Year 3 |
|---|---|---|---|
| Size of Target Market | 2,500 | 3,500 | 5,000 |
| Annual Growth Rate | 20% | 20% | 20% |
| Awareness | 40% | 45% | 50% |
| Trial Rate | 70% | 65% | 60% |
| Availability | 30% | 45% | 60% |
| Repeat | 60% | 70% | 80% |
| Total Sales - units | 1000 | 1500 | 2500 |
| Total Sales Revenue | $1,200,000 | $1,800,000 | $3,000,000 |

## *3.5 Profit Margin Projection*

| BASE + Play Area | Year 1 | Year 2 | Year 3 |
|---|---|---|---|
| Total Sales Revenue | $1,200,000 | $1,800,000 | $3,000,000 |
| Net Income | $937,000 | $1,405,500 | $2,342,500 |
| Gross Profit Margin | 21.9% | 21.9% | 21.9% |

# 4. System Architecture

## *4.1 Project Ecosystem*

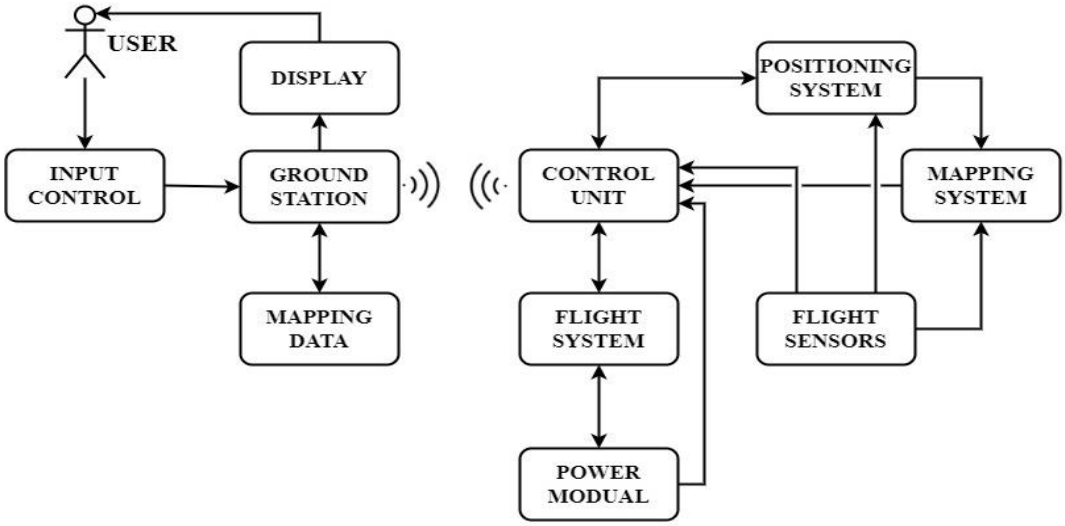



*Figure 5: Product Ecosystem*

## *4.2 Product Diagram*

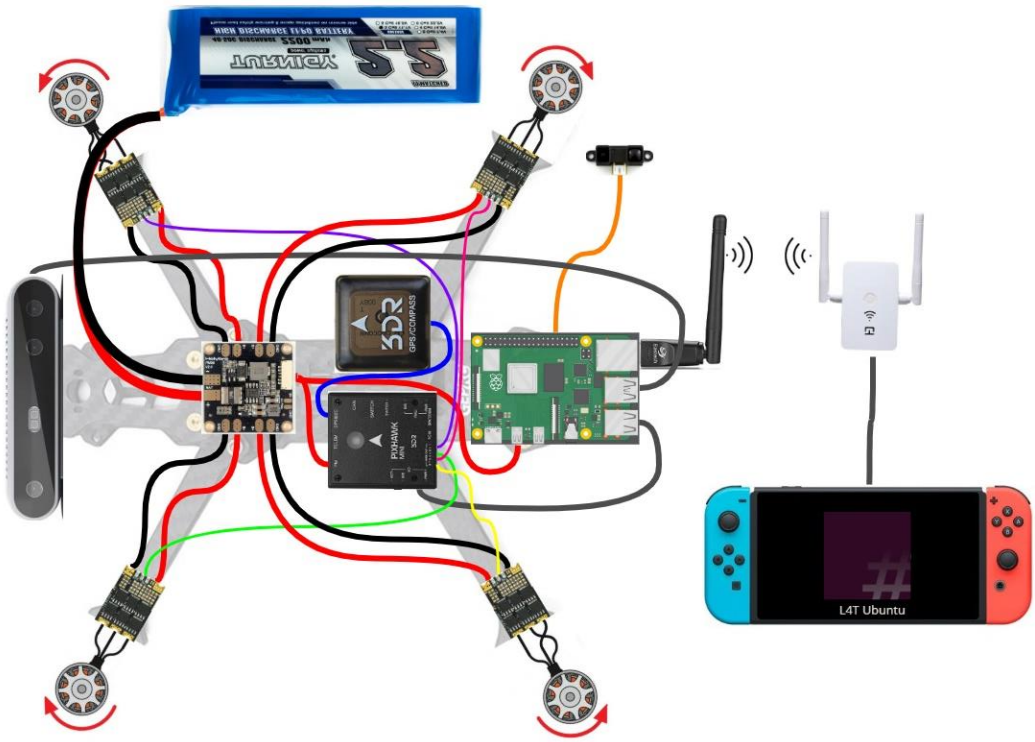



*Figure 6: Product Diagram*

## *4.3 System Breakdown*

| *Part ID#* | *Description* | *Functionality* | *What Success looks like?* |
|---|---|---|---|
| **SB-PRT-001** | Drone Platform | The platform on which the flight electronic will be attached too. Limited to a size of 1ft by 1ft by 0.5ft. | All electronics fit snugly with not wasted space and/or wire hanging near the props. |
| **SB-PRT-002** | Brushless Motors | Drone motors that provided the needed thrust for flight. | 4 motors provide a thrust to weight ratio of at least 1.2. |
| **SB-PRT-003** | Motor ESC's | Motor controllers for precise control of the Brushless motors | Precise control of all 4 motor allow for controlled flight |
| **SB-PRT-004** | Power Module | Modulate and provides power to electrical components from the 4-cell battery. | All electrical components are adequately powered without burning anything. |
| **SB-PRT-005** | 4s Battery | The main power source for the drone. | Has a large enough battery capacity to last at least 10minutes of flight. |
| **SB-PRT-006** | 3dr Pixhawk mini | The main flight controller for the mapping drone. | Must precisely maintain flight with a clear response to flight commands. |
| **SB-PRT-007** | GPS Module | Global Positioning System to localization of the drone. | Connect to at least 4 satellites. |
| **SB-PRT-008** | Raspberry Pi 4 | Main on board mapping control system. | Receive sensor data from all connected system and successfully transmit to the ground station. |
| **SB-PRT-009** | Spot Lidar | Uses light pulses to measure a singular distance | Accurately read a distance from 5in to 24in |
| **SB-PRT-010** | Intel RealSense d415 | 3d camera that generates a point cloud in 3d space | Slam success fully connects to the real sense camera and generate a rough map |
| **SB-PRT-011** | Wi-Fi Booster | Boost the Wi-Fi signal of the ground station | Ground station and flight system maintain connection to a minimum of 20ft |
| **SB-PRT-012** | Ground Station | Controller running slam and mapping the information from the drone | Receives all information from flight system and successfully processes and generate map |
| **SB-PRT-013** | PLA | Light and easy to 3d print that would hold the added components together | Does not add too much weight and can withstand the vibrations and light impacts |
| **SB-PRT-014** | 5inch Props | 5-inch propeller attach to the drone to produce thrust | Produces enough thrust for the drone to achieve lift |

## *4.4 Building Blocks*

### *4.4.1 Flight Platform*

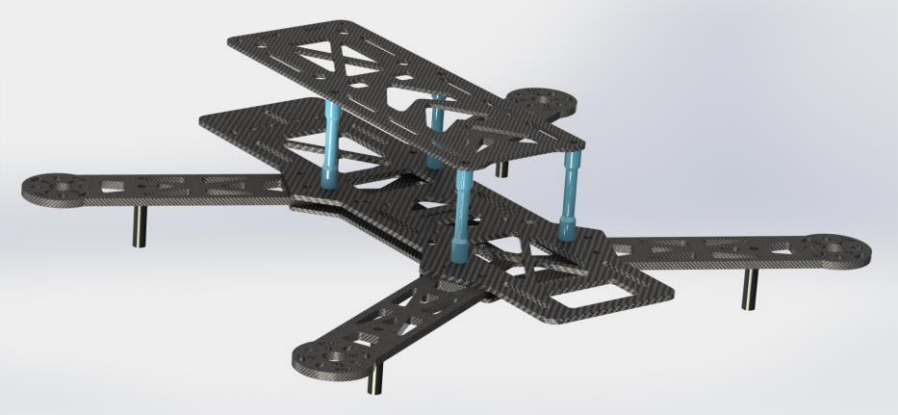


*Figure 7: Initial Flight Platform*

This building block is directed at achieving successful flight with all the components of the flight platform. The components consist of parts like the flight controller (PixHawk), brushless motors, motor ESC, and the power module. Once all parts have been collected, they will be assembled is as shown in the Product Diagram. They will all be placed on the carbon fiber platform that is depicted in the image above (CAD). If flight is attainable, we begin measuring the lift of the drone and begin considering the weight of other devices like the RealSense camera, Raspberry, and SPOT Lidar

### *4.4.2 3D Mapping Camera*

*Figure 8: Intel Realsense D415*

This building block's aim is to interface the RealSense with the Raspberry Pi. There are several modules and libraries required in order to get the two to interface. Once a stable connection, we begin testing source code to successfully transmit data points both accurately and efficiently. This data will be transmitted in real time to our Ground Station, the Nintendo Switch running Ubuntu. The Ground Station will receive these images via WIFI, download the data, and begin forming a preliminary map of the room.

### *4.4.3* ***Control Unit integration***


*Figure 9: Pixhawk and Raspberry pi Integration*

This building block is located on the flight platform. Once we have successfully established stable flight, we aim to interface the Pixhawk with the Raspberry Pi. This is necessary to control the Drone from the Ground Station and receive accurate quick flight instructions when in manual mode. The interfacing of these two devices will be tested by attempting manual flight control from a standard computer. Once this has been done successfully, manual flight will be tested from our handheld device.

### *4.4.4* ***Sensory Systems***


*Figure 10: Components of Sensory System*

The focus here is data analysis. The goal is to successfully interface the Lidar, RealSense, and GPS sensors with the Raspberry Pi on the Flight Platform. Each sensor provides a crucial piece to mapping the surrounding terrain. The data sourced from these components will be used with a SLAM algorithm for autonomous movement. Once the data is collected on the raspberry pi, the next goal is transmitting this information in real time to Ground Station.

### *4.4.5* ***Ground Station and Communication***


*Figure 11: Ground Station Communications*

Once we can receive data from the various sensors to the Raspberry Pi, the goal is to transmit this data in real time to the Ground Station. The Ground Station consists of a modified Nintendo Switch running Nvidia's "Linux 4 Tegra" kernel and an external Wi-Fi repeater. A Nintendo switch was chosen to be our ground station because it has a built-in display and controller, along with having the same SoC as the one used in the Nvidia Jetson nano. Having hardware comparable to a Jetson nano will allow us to make use of the Jetpack api and specialized GPU hardware for image processing. The Wi-Fi booster will improve the Wi-Fi connectivity between the drone and Nintendo Switch

### *4.4.6* ***Preliminary 3D Mapping***

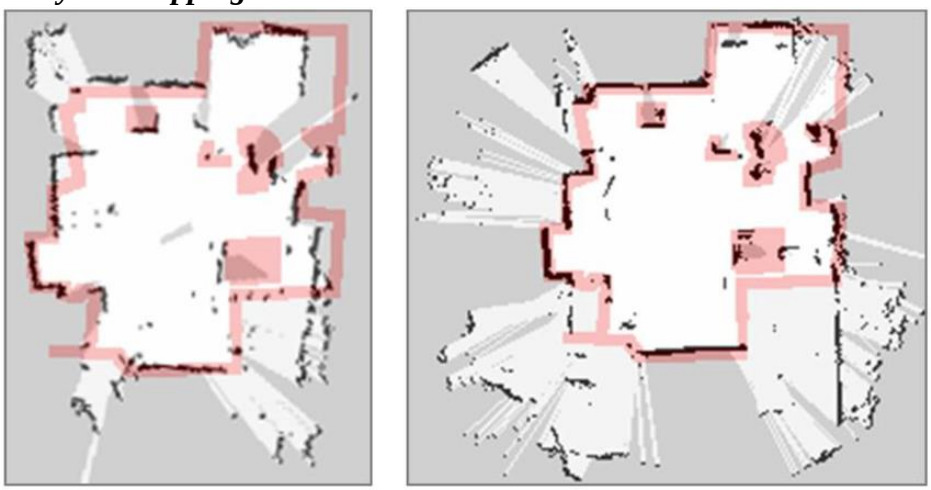


*Figure 12: Initial ROS RVIZ Mapping*

Mapping data will be transmitted from the drone to the ground station in real time. The main source of this data will come from the stereo IR camera on the RealSense D415. 3D mapping will be handled by RTAB Map, a program used within ROS. 2D mapping will be done by RVIZ using the data from the spot lidars. The mapping capabilities will be tested in Gazebo, a simulation program included with ROS.

### *4.4.7* ***Controlled Flight***

*Figure 13: First Test of Controlled Flight*

The aim here is to take a step back from the handheld controller and be able to control the drone via a home desktop. This is achievable once the PixHawk and Raspberry Pi have been interfaced. This is a crucial step that allows control from any location if the Raspberry Pi is connected via WIFI.

### *4.4.8* ***Platform and Program Optimizations***


*Figure 14: ROS System*

Robot Operating System (ROS) is an open-source programming framework used in writing software for robotic systems. It includes a suite of tools and libraries designed for simplifying the task of developing code for automated robots. In order to improve stability and performance, ROS will be run from within docker. Docker is a very popular project software that uses Linux Kernel features such as namespaces and control groups for containers to be created.

# 5. Market Requirements

## *5.1 - Key Product Features (Functional Requirements)*

| *Market Requirement ID* | *Functional Requirement* | *Verification Method* | *Qualification* |
|---|---|---|---|
| **MR-FNC-001** | When in manual mode the drone must be in full control of the ground station operator. | Design | *Must have* |
| **MR-FNC-002** | Drone must be able to carry the added weight of the sensory system. | Design | *Must have* |
| **MR-FNC-003** | Ground station should keep a save scanned in case of drone failure. | Design | *Should have* |
| **MR-FNC-004** | Drone should have an emergency shutoff procedure. | Design | *Should have* |
| **MR-FNC-005** | Drone must be able to autonomously maintain a distance from nearby objects. | Design and Testing | *Must have* |
| **MR-FNC-006** | Drone must be able to maintain connection to ground station. | Program | *Must have* |
| **MR-FNC-007** | Drone should safely return to ground station if connection is lost. | Design | *Should have* |
| **MR-FNC-008** | Sensory system must accurately track altitude in relation to ground station. | Program | *Must have* |
| **MR-FNC-009** | Sensory system must accurately track pitch, yaw, and roll. | Program | *Must have* |
| **MR-FNC-010** | When in auto mode the drone must be able to proceed to the mapping protocol. | Design | *Must have* |
| **MR-FNC-011** | The drone should be able to autonomously navigate to the waypoint presented. | Program | *Should have* |
| **MR-FNC-012** | The drone must transmit scanned surface back to ground station in real time. | Design | *Must have* |
| **MR-FNC-013** | Drone must be able to track its x, y, and z position in relation to ground station. | Design | *Must have* |

## 5.2 - Key Performance Requirements

| Market Requirement ID | Performance Requirement | Verification Method | Qualification |
|---|---|---|---|
| **MR-PER-001** | Drone shall have a thrust to weight ratio of at least 1.2. | Design | Must have |
| **MR-PER-002** | The Drone shall not exceed the dimensions 1ft*1ft*0.5ft (L*W*H). | Design | Must have |
| **MR-PER-003** | Drone shall be able to maintain flight for a minimum of 10mins. | Design | Must have |
| **MR-PER-004** | Propeller shall be safeguarded for safety. | Design | Should have |
| **MR-PER-005** | Ground station must maintain communication at a minimum distance of 40ft. | Design | Must Have |
| **MR-PER-006** | Drone shall be able to map the space within 3.5 meters of itself. | Design | Must Have |
| **MR-PER-007** | Drone must keep track of itself in relation to initial takeoff point. | Design | Must Have |
| **MR-PER-008** | Drone should be able to return to takeoff point at a moment's notice. | Program | Should Have |
| **MR-PER-009** | Drone must be capable of real-time mapping and transmission back to the ground station. | Design | Must have |

## 5.3 Product Documentation Requirements

| Market Requirement ID | Documentation Requirement | Verification Method | Qualification |
|---|---|---|---|
| **MR-DOC-001** | An instruction sheet that will explain the various sensors and their locations. | Focus Group | Must Have |
| **MR-DOC-002** | A video demonstrating the use of the drone in manual and autonomous mode. | Focus Group | Must Have |
| **MR-DOC-003** | A safety/warning document. | Focus Group | Must Have |
| **MR-DOC-004** | System diagram of drone. | Focus Group | Must Have |
| **MR-DOC-005** | Orthographic designs. | Focus Group | Must Have |
| **MR-DOC-006** | Detailed instruction of code operations. | Focus Group | Must Have |
| **MR-DOC-007** | Addendum of important equations and functions | Focus Group | Must Have |

### *5.4 Miscellaneous Market Requirements*

| *Market Requirement ID* | *Miscellaneous Requirement* | *Verification Method* | *Qualification* |
|---|---|---|---|
| **MR-MISC-001** | Drone shall be compliant with FAA guidelines. | Design | Must Have |
| **MR-MISC-002** | The system shall meet FCC regulatory compliance | Design | Must Have |
| **MR-MISC-003** | Must follow Asimov's laws of Robotics | Design | Must have |
| **MR-MISC-004** | The system shall meet CE regulatory compliance | Design | Must have |
| **MR-MISC-005** | The system shall meet UL regulatory compliance | Design | Must have |

# 6. Product Requirements

## *6.1 Performance Requirement Mapping*

These Performance Requirement Depend on The Mitigation of the following Risks:

| *Building Block* | *RISK ID#* | *Description* | *MR-PER-001* | *MR-PER-002* | *MR-PER-003* | *MR-PER-004* | *MR-PER-005* | *MR-PER-006* | *MR-PER-007* | *MR-PER-008* | *MR-PER-009* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *3D Mapping Camera* | RSK-001 | 3D Mapping software/hardware | | | | | | **X** | | | **X** |
| *Controlled Flight* | RSK-002 | Autonomous Flight | **X** | | | | **X** | | **X** | **X** | |
| | RSK-003 | Flight failure | **X** | | **X** | | | | **X** | | |
| *Sensory System* | RSK-004 | Navigation Failure | | | | | **X** | **X** | **X** | **X** | **X** |
| *Ground Station and Comms.* | RSK-005 | Realtime Communications | | | | | **X** | **X** | | | **X** |
| *Flight Platform* | RSK-006 | Drone Structure | | **X** | **X** | **X** | | | | | |
| *Control Unit integration* | RSK-007 | Control Unit Integration | | | | | | | **X** | **X** | **X** |
| *Preliminary 3D Mapping* | RSK-008 | Sensory System | | | | | **X** | **X** | **X** | **X** | **X** |
| *Program Optimization* | RSK-009 | ROS Docker processing | | | | | | | | | **X** |

## *6.2 Functional Requirement Mapping*

These Functional Requirement Depend on The Mitigation of the following Risks:

| *Building Block* | *RISK ID#* | *Description* | *MR-FNC-001* | *MR-FNC-002* | *MR-FNC-003* | *MR-FNC-004* | *MR-FNC-005* | *MR-FNC-006* | *MR-FNC-007* | *MR-FNC-008* | *MR-FNC-009* | *MR-FNC-010* | *MR-FNC-011* | *MR-FNC-012* | *MR-FNC-013* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *3D Mapping Camera* | RSK-001 | 3D Mapping software/hardware | | | X | | | | | | | | | X | |
| *Controlled Flight* | RSK-002 | Autonomous Flight | | X | | | X | X | | X | X | X | X | X | X |
| | RSK-003 | Flight failure | | X | | | | | | | | | | | |
| *Sensory System* | RSK-004 | Navigation Failure | | | | | X | X | X | X | X | | X | | |
| *Ground Station and Comms.* | RSK-005 | Realtime Communications | X | | X | X | | X | | | | | | X | |
| *Flight Platform* | RSK-006 | Drone Structure | | X | | | X | | X | | | | X | | |
| *Control Unit integration* | RSK-007 | Control Unit Integration | X | | X | X | X | X | X | | | X | X | X | X |
| *Preliminary 3D Mapping* | RSK-008 | Sensory System | | | | X | X | | | X | X | | X | | X |
| *Program Optimization* | RSK-009 | ROS Docker processing | | | X | | | | | | | X | | X | |

# 7. Project Implementation

## *7.1 Milestone1 – Pipelines*

### *7.1.1 Task 1: Send PixHawk information through the pi to the base station*

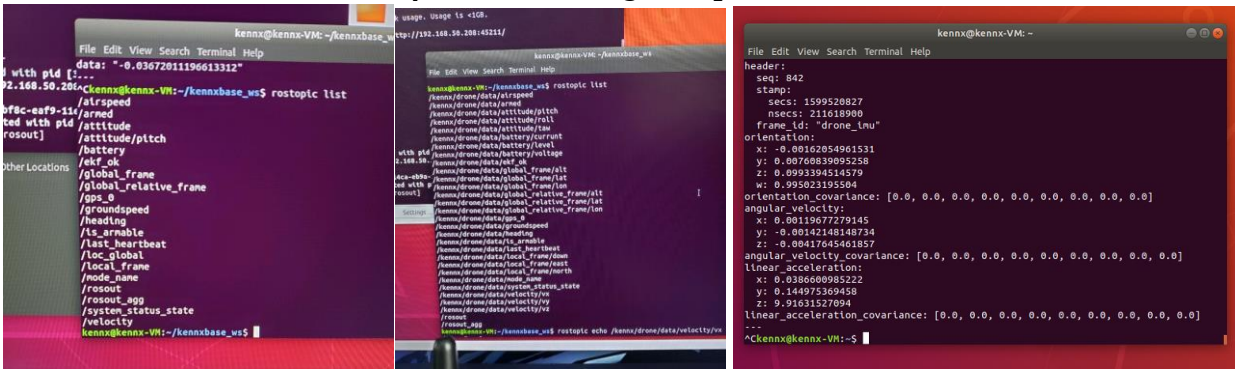



*Figure 15: PixHawk data*

The focus is the establishment of communications from the PixHawk to the base station. The sequence to steps required the connection to the raspberry pi first where we utilize the py drone kit library to access all parameter and sensory information provided from the PixHawk via a usb connection. The data collected would be Attitude (Gyro), Heading (Magnetometer), Global location (altitude), Battery, Velocity, and System information (Status, Mode, Armable).

The following sensory information would be comprised into a custom RoS msg that the base would be capable of subscribing to. This information would be used in the tracking and status checks of the drone during its flight plan.

### *7.1.2 Task 2: Send video feed from Realsense to the pi to the base station*

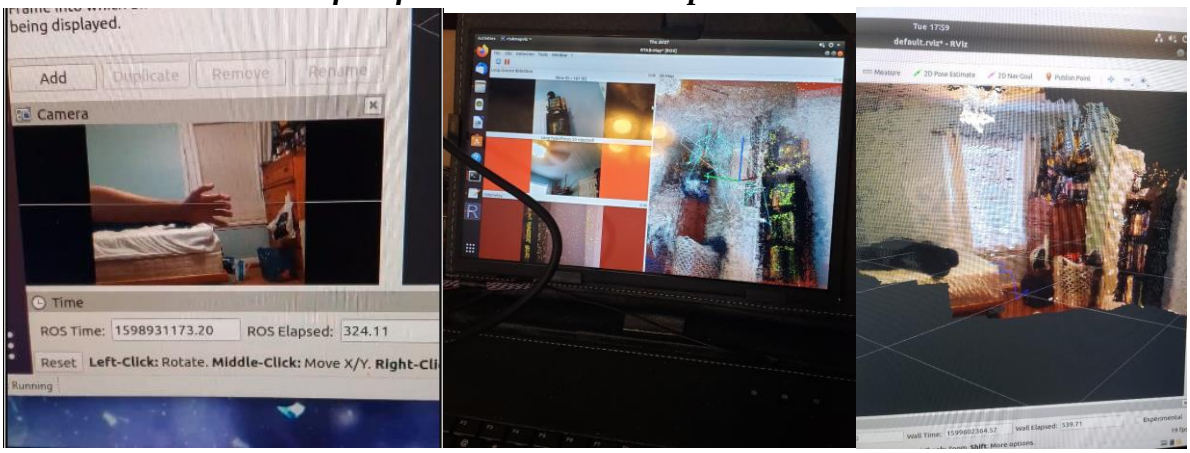



*Figure 16: RGB and Depth msg*

The raspberry pi will publish RGB video data from the Realsense camera to the base station. The Realsense RGB feed published from the drone is viewable from the base station's display as either a series of images or a live video stream. During the setup of the connection the stream was only able to achieve a constant FPS of 3HZ for the video and 1HZ from the depth field. A better compression system will have to be utilized and possible use a Wi-Fi dongle to better boost the Wi-Fi signal.

### *7.1.3* ***Task 3: Publish commands to the pi from the base station***

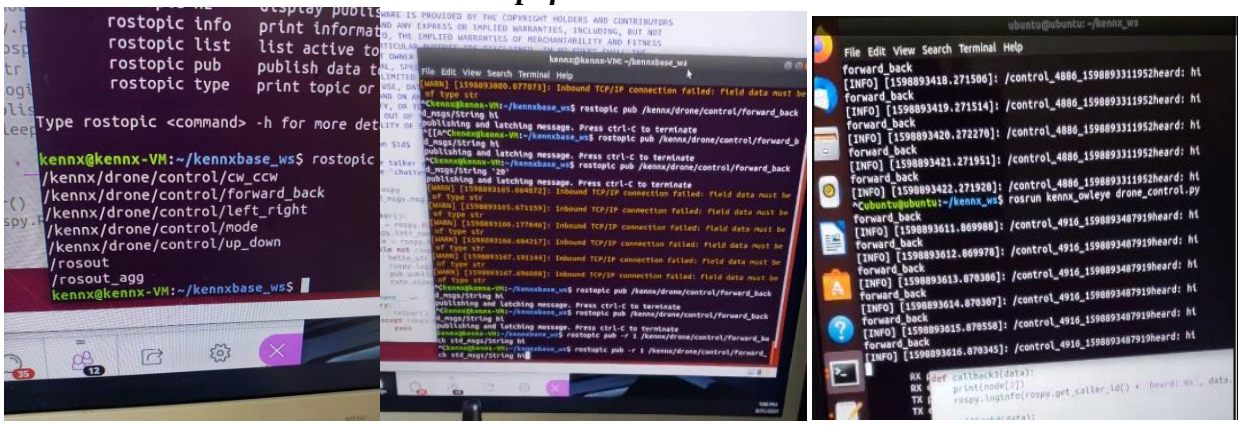



*Figure 17: Controller node*

The base station will publish topics used for controlling the drone. Published data values will be viewable in the terminal. The Raspberry Pi will subscribe to the topics related to drone control. The data received by the Pi will be viewable from the device's terminal. With the data now available to the base station the process of integrating the values for controlled flight and Rtabmap can procced.

### *7.1.4* ***Task 4: Controlled flight from the base station / Controlled flight from pi***

*Figure 18: First Stable Flight*

Utilizing the RoS joy node the connected controller values will be adjust to a range of 1100 to 1900, the range of value the PixHawk uses to drive the brushless motors. Once set the controller can direct the drone forwards/backwards, left/right, clockwise/counterclockwise, and fly up/down.

During test it became apparent that the transmission delay clocked at 20ms, due to the system being real time more time needs to be set into reducing the delay as it was difficult to properly fly the drone. In addition, much better tuning needs to take place since the drone was far too sensitive. Lastly, Due to the COVID-19 pandemic this task experienced delays due to lack of in person coordination and suitable space for test flights.

### *7.1.5* ***Task 5: Have all sensors accounted for and a python script written for each sensor***

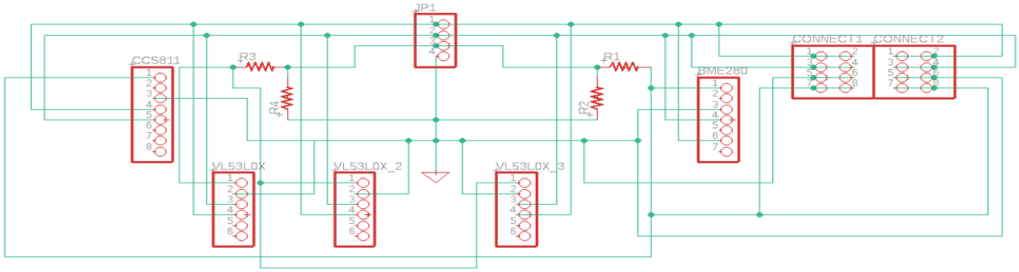



*Figure 19: PCB Schematic v1*

A bill of materials will be drafted for the sensors used on the PCB. The gas, particle, carbon monoxide, lidar, and humidity sensors will be individually inspected for their port type (I2C/SPI) and ordered. In addition, a unique python script will be written for each corresponding sensor and contain the necessary libraries to properly read the raw data. Once able to print data from all sensors, ROS topics will be created for each sensor that will then be requested by the base station and display data readings.

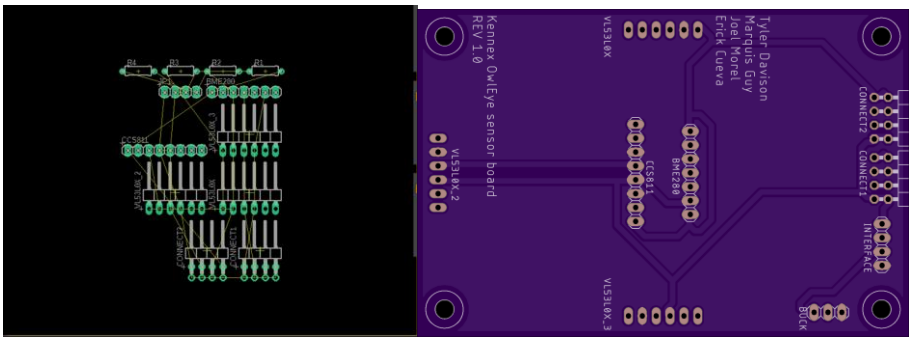



*Figure 20: PCB Design v1*

***BME280*** **(Required voltage 3.3$^{V}$)**

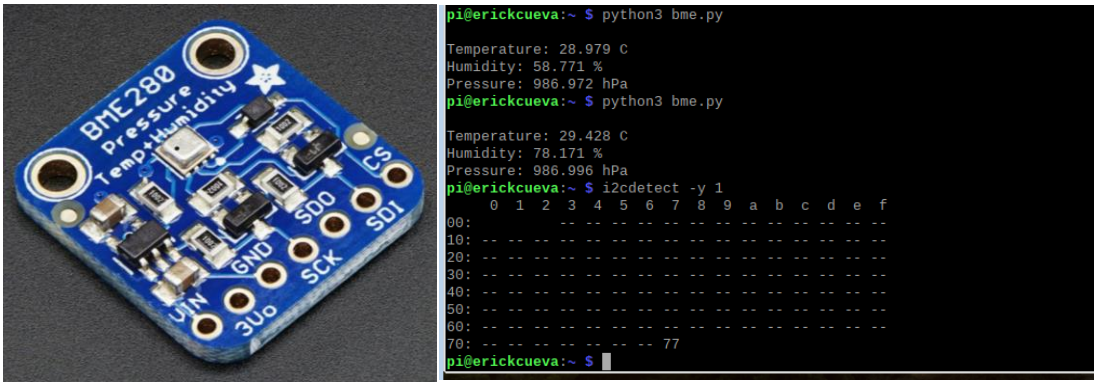



*Figure 21: BME280*

Test Code:

```
import board
import busio
import adafruit_bme280 i2c = busio.I2C(board.SCL, board.SDA)
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)
print("\nTemperature: %0.3f C" % bme280.temperature)
print("Humidity: %0.3f %%" % bme280.humidity)
print("Pressure: %0.3f hPa" % bme280.pressure)
```

***CCS811 Air Quality Sensor Breakout*** **(Required Voltage 3. 3V)**

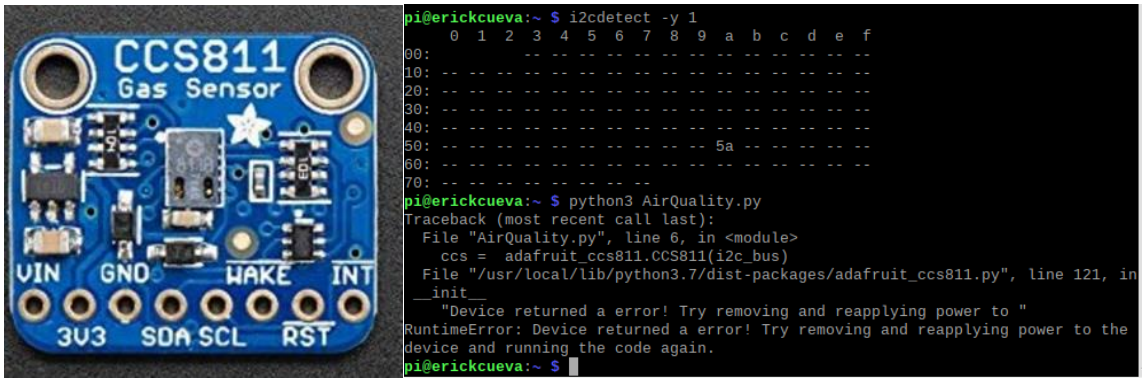



*Figure 22: CCS811*

Test Code:

```
import time
import board
import busio
import adafruit_ccs811 i2c = busio.I2C(board.SCL, board.SDA)
ccs811 = adafruit_ccs811.CCS811(i2c)

while not ccs811.data_ready:
   pass while True:
print("CO2: {} PPM, TVOC: {} PPB".format(ccs811.eco2, ccs811.tvoc))    time.sleep(0.5)
```

***HiLetgo VL53L0X Time-of-Flight Flight Distance Measurement Sensor*** **(Required Voltage 2.8V)**

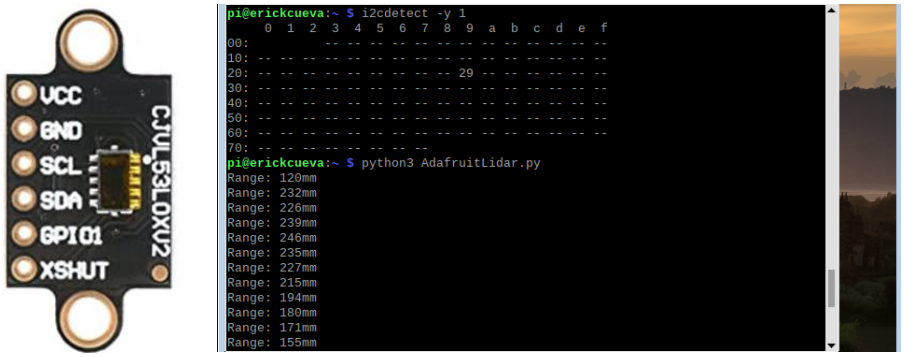



*Figure 23: VL53L0X*

Test Code:

```
import time
import board
import busio
import adafruit_vl53l0x
i2c = busio.I2C(board.SCL, board.SDA)
vl53 = adafruit_vl53l0x.VL53L0X(i2c)
while True:
   print("Range: {0}mm".format(vl53.range))
   time.sleep(1.0)
```

## 7.2 Milestone2 – Sensory Development

### 7.2.1 Task 1: Sensor Addressing

```
pi@erickcueva:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- 29 -- -- -- -- -- --
30: 30 31 32 33 -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- 5a -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- 77
```

```
pi@erickcueva:~ $ python3 AdafruitLidar.py
Range of sensor 1: 147mm
Range of sensor 2: 199mm
Range of sensor 3: 174mm
Range of sensor 4: 202mm
Range of sensor 5: 8190mm

Temperature: 26.566 C
Humidity: 62.423 %
Pressure: 984.267 hPa
```

*Figure 24: Sensor I2C Address*

The libraries for each of the sensors were found and downloaded to interface them properly. Four python scripts were written to change the addresses of the VL53 lidars. Each script powers on the next GPIO and sets its address. We will run all four every time we power on due to the fact the lidars lose their address once powered off. The code provided bellow demonstrates the first python script which sets four GPIO pins too LOW to address the first VL53 lidar.

Once all the lidar sensors are properly addressed, a test will be conducted using all seven sensors. This test is part one of two videos created for milestone one. The test will be done using the breadboard before soldering the sensors to the PCB. Here is an image of the breadboard set up and an image of the i2cdetect output of all the addresses. Addresses 0x29-0x33 are the VL53 lidars, 0x5a is the CCS811 Air Quality sensor, and the 0x77 is the BME280. A script was then run to pull values from all the addresses and print them in the terminal.

### 7.2.2 Task 2: PCB Implementation with Sensors

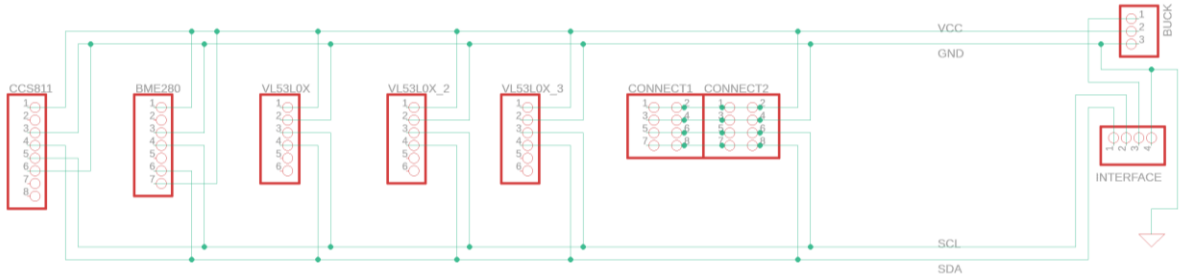



*Figure 25: PCB Schematic V1.1*

Once a finalized schematic of our PCB was designed in milestone one, the eagle directory files were sent to a PCB manufacturer where they custom built our board. The following image is version one of our PCB. I, Erick, made the mistake of not including the GPIO pins within the PCB requirements when given to Tyler. Version One works flawlessly with jumper cables added from the GPIO ports of the raspberry pi to the XSHUT of each VL53 lidar. Version Two of the PCB will have the extra GPIO pins that were missed the first time around.

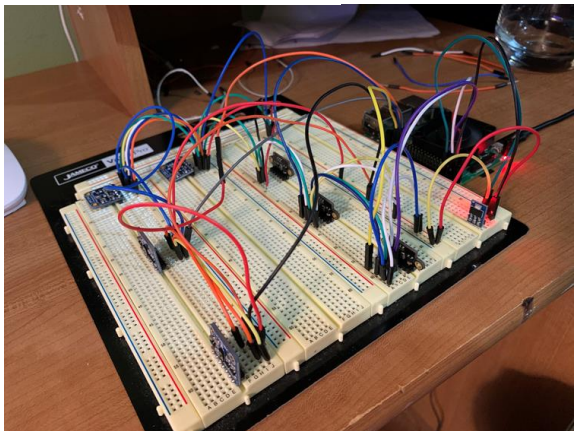


*Figure 26: Initial Sensor test*

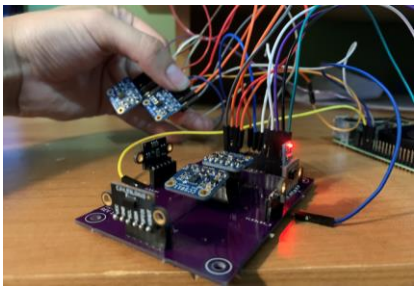


Figure 27: PCB V1

Once the PCB arrived and the breadboard test was complete, all the sensors were soldered to the PCB. A second test was conducted which is part two of my video. This test demonstrates all the sensors connected to the PCB and their addresses being individually set. Below are images of the PCB with the sensors soldered, the i2c detect output, and terminal sensor data.

### *7.2.3 Task 3: Preliminary Scan of Obstacle Course*

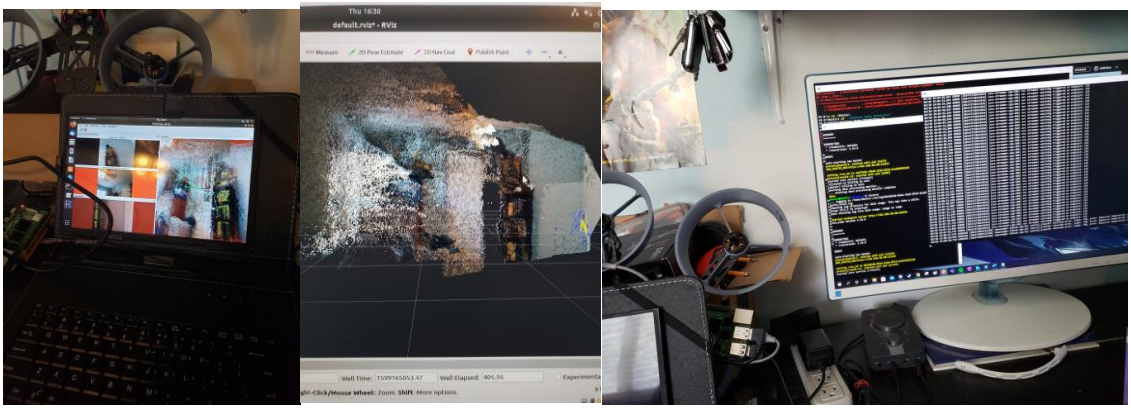


Figure 28: Preliminary scan

Using the demo code provided by the Realsense camera, the drone will do a preliminary scan of an obstacle course created. The drone will not be in flight mode, instead a walkthrough with the drone in hand will be conducted. The purpose is to obtain accurate RGB, IR, and dot point could data on the base station. Only the camera data topic will be requested, tested, and displayed on the base station to prepare for full interfacing in milestone three.

## *7.3 Milestone3 – Device Synthesis*

### *7.3.1 Task 1: Interfacing the Drone with the PCB*

In order to properly address each lidar by milestone two, a python script was created for each individual lidar. By milestone three, we have reduced these five scripts into a single script that addresses each of the lidars using the GPIO and XSHUT pins. Their addresses start at 0x29 and end at 0x2D. Now, there are only two scripts, one that addresses the lidars and the other that prints all the sensor data. Now that the code is simplified as much as possible, these files were transferred from a Raspien environment onto a Raspberry Pi 4 running Ubuntu 18.04. This is done to begin working in the same environment the drone is in. Once all the packages were downloaded, both scripts were run and worked fine. An issue we ran into is the execution of the script that addresses the lidars. This script can only be run once at startup, and if there is an error with addressing, the pi4 needs to be rebooted.

Due to some mistakes made the first time around on version 1.0, version 1.1 of our PCB was sent off. The size of version 1.0 was slightly too large and would not fit on the underside of the drone. For this reason, we were not able to hook the PCB on the drone, but we were able to demo it properly working in the milestone video. Despite not having the PCB attached to the drone, we have demonstrated it functioning properly within the Ubuntu environment by addressing and printing the sensor values.

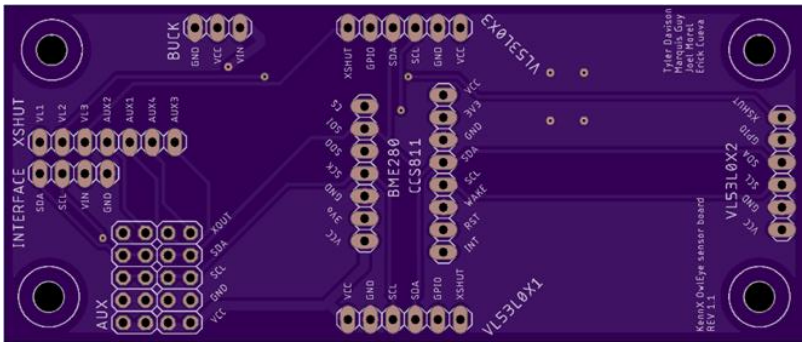



*Figure 29: PCB Design v1.1*

The new PCB will have the adequate XSHUT pins needed for the lidars and two auxiliary ports if we would like to add another sensor.

### 7.3.2 **Task 2: 3d scan with a controlled flight**

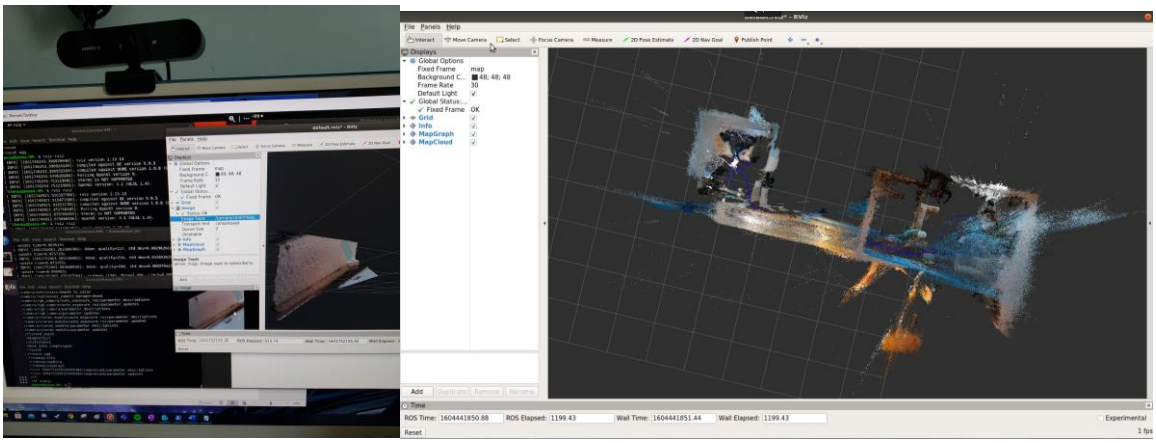


*Figure 30: First full scan*

The drone will Create a 3d scan while in flight or by handheld to test and prove the sensory connection, process, and accuracy. The 3d image should be recognizable. We were able to get full control from the base station to the drone and accurate 3d mapping on a virtual machine. Sadly, due to a major project oversight we failed to do an inflight 3d scan as the Nintendo could not handle the 3d software as such we will be switching to a laptop with ubuntu installed.

### *7.3.3 Task 3: Preliminary Navigation and Redundancy features*

**Failsafe code implementation**

A node will be dedicated towards handling the failsafe procedures for our drone. The node will subscribe to controller information from the base station, along with IMU and battery voltage data from the master drone node. While the drone is in an ideal position or state, the failsafe node will publish the controller data to the drone's master node. If the laser sensors, IMU, or battery voltage data indicate that the drone is in an unfavorable position or state, the failsafe node will publish directional data for placing the drone into a more favorable state instead of the controller data.

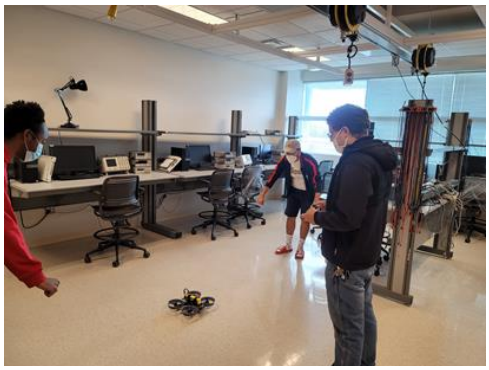


*Figure 31: Controlled flight test*

**The redundancy feature to test will be crash avoidance, low power scenario, and crash detection.**

- In crash avoidance they will be placed to hover, and a cardboard sheet will approach the drone from the back, sides, and front. The drone should move away to maintain a minimum distance of 5 cm. Lastly the drone will do a slow fly forward to a box and it should stop without hitting the box so hard that it falls.
- For low power, the battery will be drained up to at least 5%. Once the threshold is reached the drone should forcefully land ignoring pilot input.
- For crash detection when the Z-axis of the IMU reads 9.8 meters per sec, the failsafe code will disable the drone's motors, indicating crash detection.

# 8. Challenges and Solutions

## *8.1 Rtabmap*

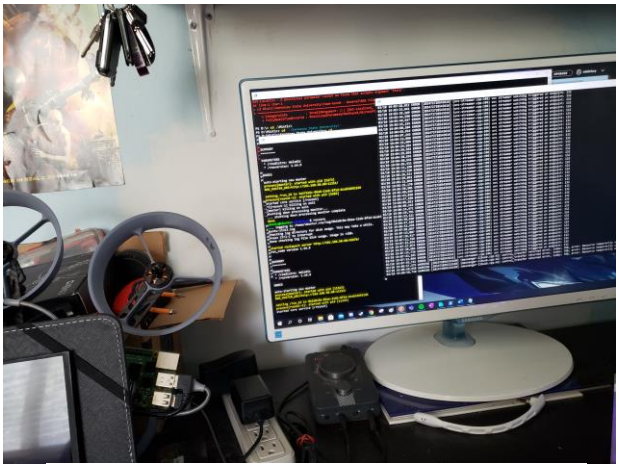


*Figure 32: Rtabmap Optimization*

During the project, our main choice for the 3d map was Rtabmap due to it being open source and integrated with RoS the main operation system for controlling and communication with the drone. The biggest issue we faced with Rtabmap was the sever lack of documentation. Since Rtabmap map is mainly utilized in a 2d configuration not many people worked on in in a 3d configuration thus a lot of the setup and optimization was left to trial and error. Moreover, due to COVID-19 testing was limited since the main base station at the time was a Nintendo switch and many parts of the system was divided between each member.

Later in the game stage it was discovered that the Nintendo switch we intended to use as the base station was unable to fully support Rtabmap as the processing power was not enough. Luckily, all files were easily transferred to a laptop with ubuntu 18.04.

## *8.2 Communication and Latency*

The greatest hurdle to the project was the excessive delays experienced, the rgb and depth field data at worst came in at less than 1HZ making it unsuitable to feed to the Rtabmap. This issue was resolved by using the RoS msg throttle package to compress the rgb and depth filed data. In the end we were able to optimize for a stable rate of 8HZ, for future sake it would be better to boost the signal or switch to using udp instead of tcp for the image broadcasting.

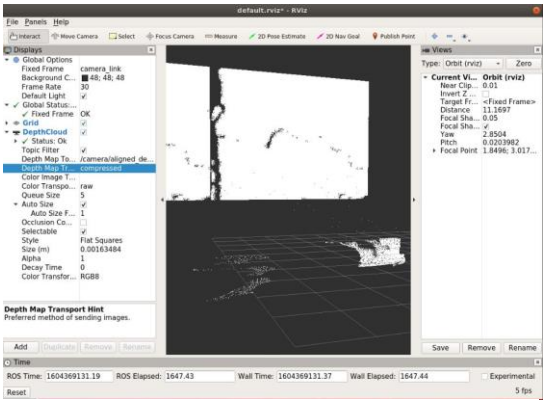


*Figure 33: Corrupted Depth Compression*

## 8.3 PCB Design

One of the biggest challenges when making our board was converting the project in eagle from a schematic to a PCB design. When the PCB design was first loaded, we saw a web of yellow lines between the connections and thought those were the autogenerated traces. It was later discovered that the lines were not traces but instead representations of the connections between the components on the PCB. The traces were eventually added into the design with the auto routing tool.

Our first PCB design that came in was made a bit too large for the drone and was not able to be latched to the belly of the drone. We needed it to be a certain size so it would fit properly, but we overlooked this in the first version. The first version was also missing XSHUT connections for each of the VL53 and instead they were all tied to ground.

One the second version of the PCB, there was a small error where the ground for the interface and buck converter were not tied, but this was easily fixed with a small jumper cable. Another error we had with the second version is one of our voltage traces was short circuiting one of the XSHUT pins. This was a hard catch to find and was causing complications when trying to change the address of each lidar.

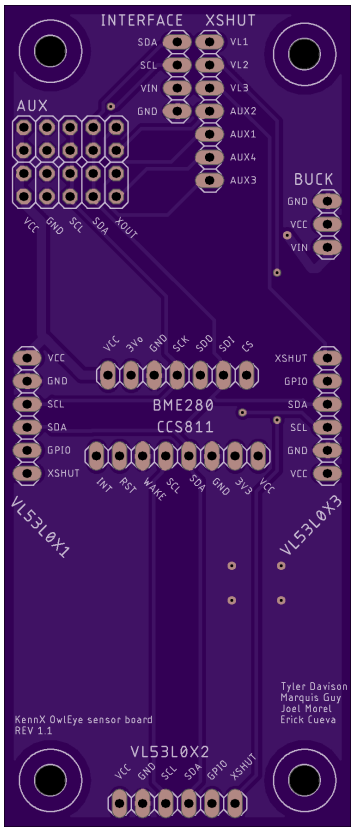



*Figure 34: PCB Design v1.1*

## 8.4 Sensor Connection Issues

Our PCB was designed for seven sensors and a buck converter. Five of those seven are the VL53L0X lidars. Three lidars are directly soldered to the left, right, and bottom pin connections on the PCB while the other two are interfaced though the auxiliary ports. This is due to the fact we wanted to point lidars in different directions. A challenge we faced with the VL53 is being able to address each one. Within the first version of the PCB, we did not create connections for the XSHUT pins of the VL53 lidars which are necessary to address each lidar. We were able to fix this by soldering some jumper cables to the XSHUT pins of each lidar which then went to the GPIO pins on the Pi. All the sensors shared the same power, ground, SDA, and SCL bus which made creating the schematic a bit easier. The interface connections from the PCB to the Pi include power, ground, SDA, SCL, and five GPIO leads for each of the VL53 lidars.

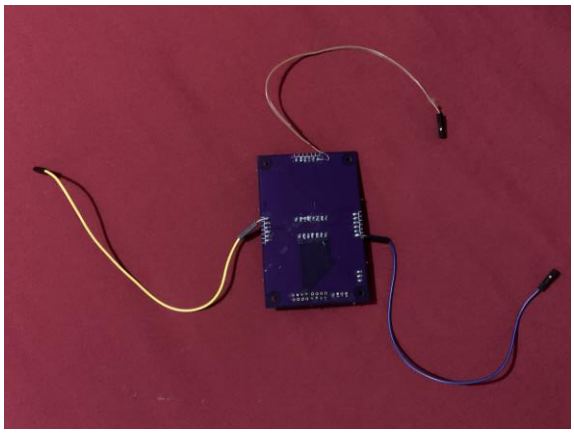


*Figure 35: PCB Connection Faults*

The final two sensors are the BME280 and the CCS811 which are located in the center of the board. The BME280 measures temperature, pressure, and humidity while the CCS811 measure carbon dioxide levels and total volatile organic compounds. We also used a buck converter that took in 5 volts from the Pi and outputted 3.3 volts which is used to power the whole board. A challenge we faced with the CCS811 is lack of documentation that explained how to interface the sensor. Every time I would run a test script to print CO2 and TVOC, I would get an error message that said to check the connection to the board. I thought this meant we had a faulty CCS811, so I ordered another. I had the same issues with the new CCS811 I ordered, so I decided to purchase a CCS811 from a different manufacture, SparkFun. This sensor came with documentation that stated we needed to enable clock stretching for the sensor to work. After editing the /boot/config.txt file and enabling clock stretching, the test script began printing the CO2 and TVOC values. Now when I tested the Adafruit CCS811, it no longer displayed and error message and began printing the sensor values.

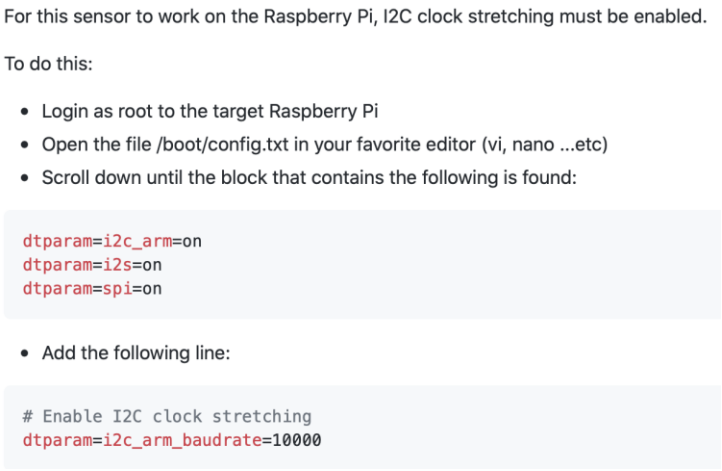



*Figure 36: CCS811 Debug*

## *8.5 Base Station and Compatibility*

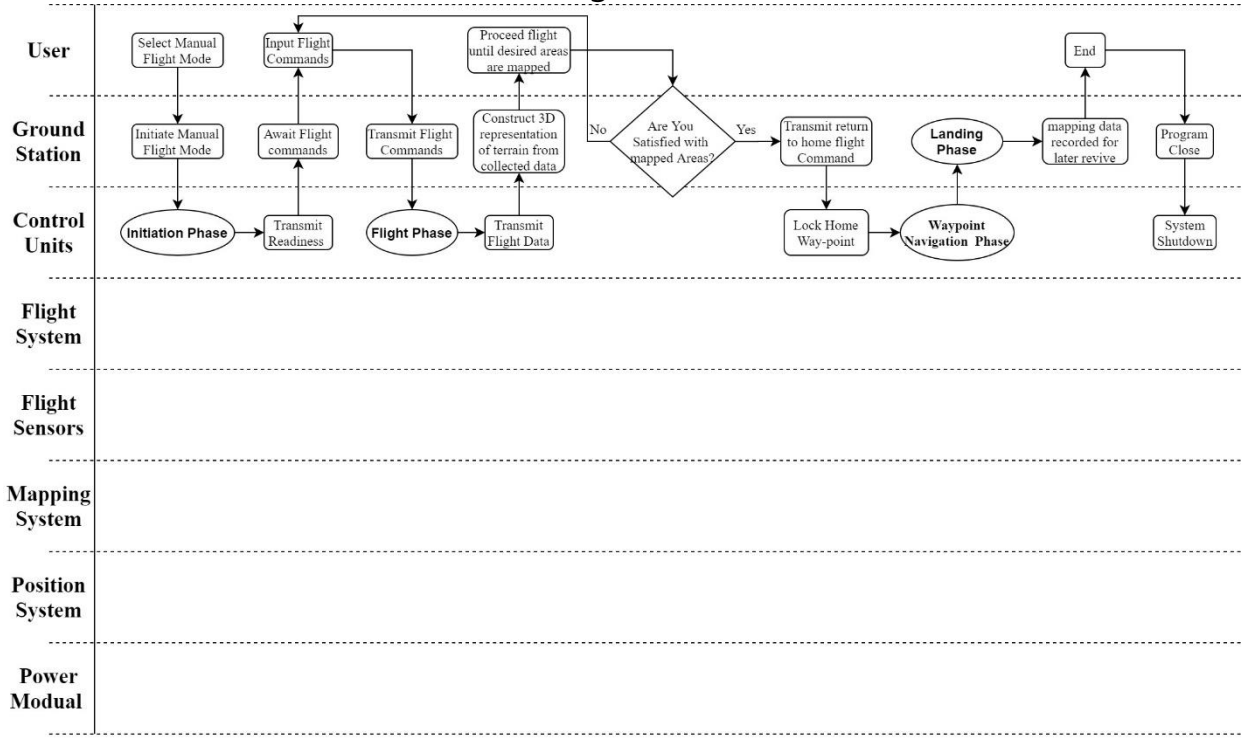
One of our primary goals was to develop a base station platform that was hardware agnostic and user adaptable. We planned to demonstrate these features by testing our drone with a handheld a device and a x86 laptop. Unfortunately, unforeseen compatibility issues on the ARM device prevented it from being used in this project.


*Figure 37: Nintendo switch*

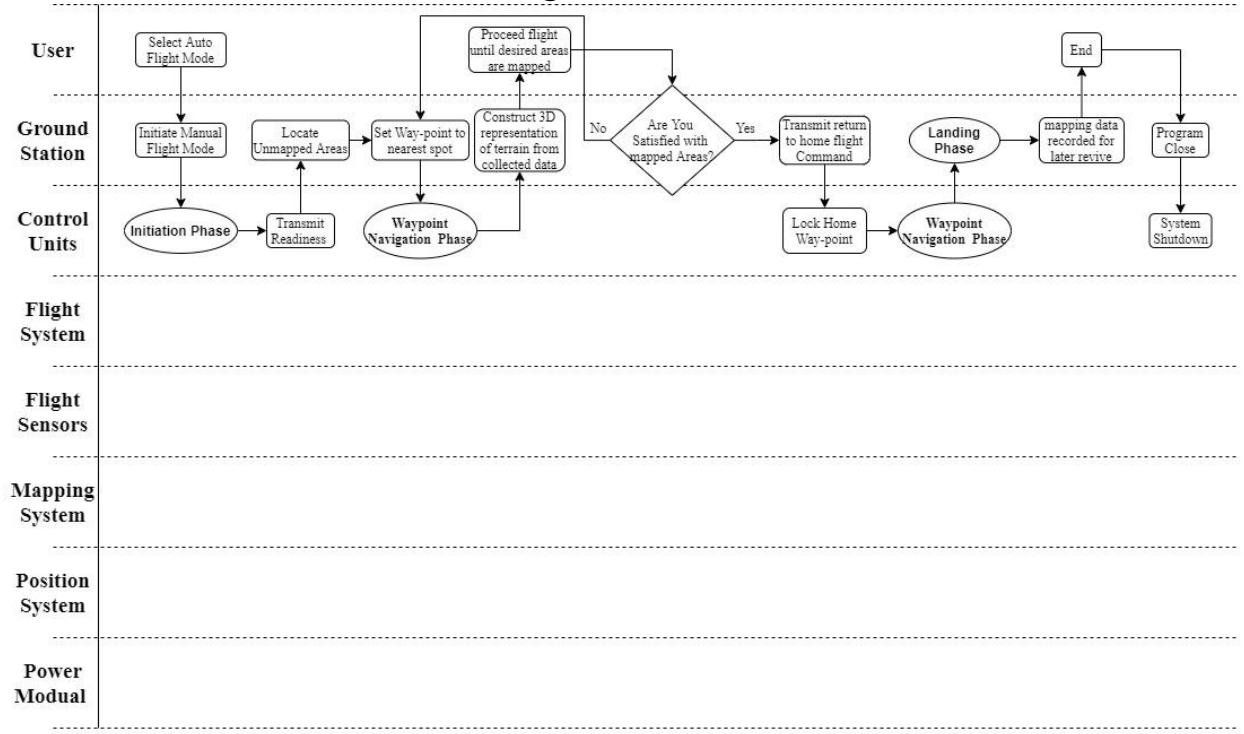
For our ARM handheld, we decided to use a Nintendo Switch modified to run the Linux 4 Tegra kernel. This device seemed like a suitable choice because it has the same SoC as the Jetson Nano while also having a built-in display and controller. Unfortunately, the Linux image used an outdated kernel and a non-standard release of the jetpack API. Both ended up causing compatibility issues with OpenCV and RTAB Map that were not noticed until milestone 4. Given the issues we were having and the short amount of available time, it was decided that only the laptop would be used as the base station.

# 9. Customer Usage Scenario

## *9.1 Use Case Scenario #1: Manual Flight Mode*

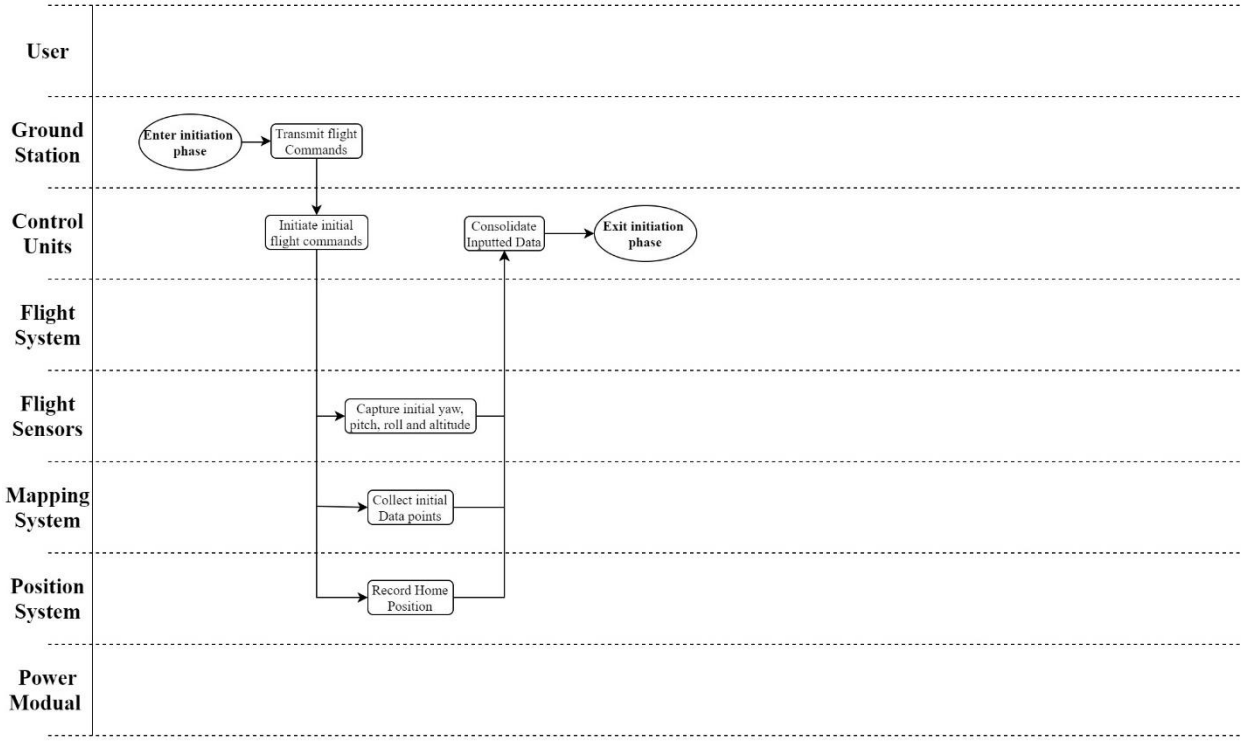



## *9.2 Use Case Scenario #2: Auto Flight Mode*

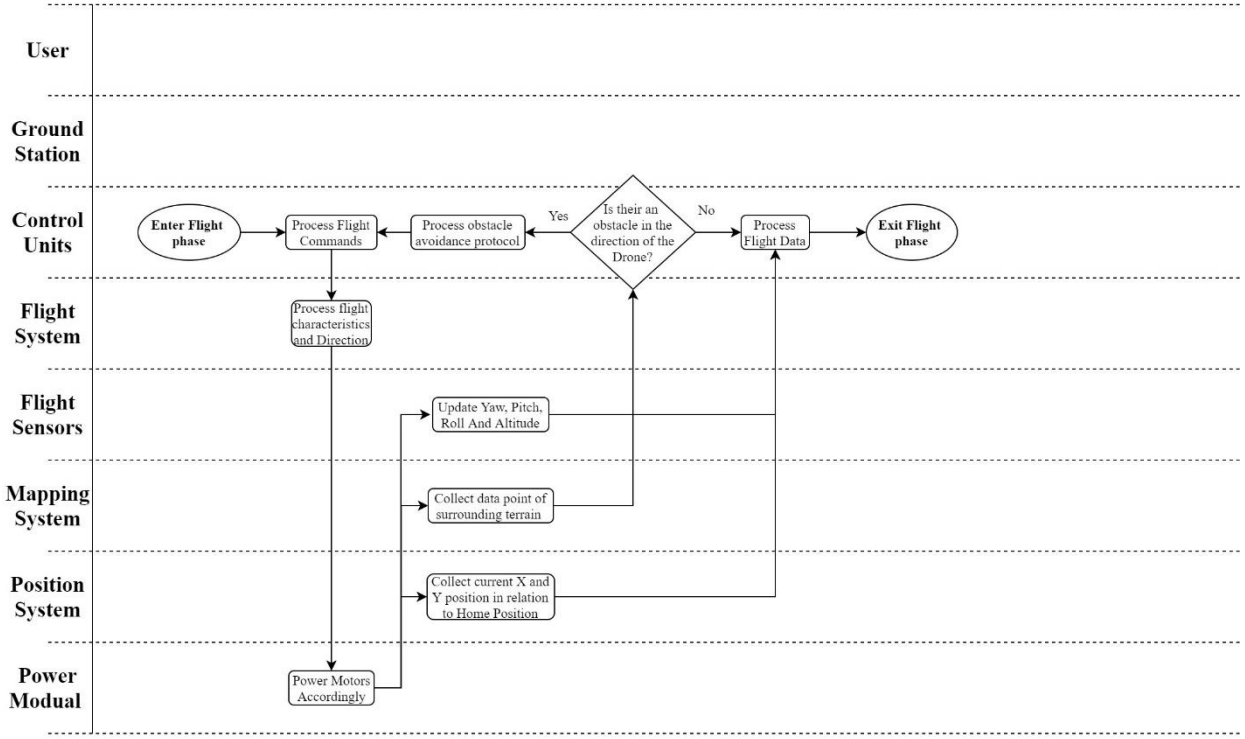


# 10. Scenario Case Phases

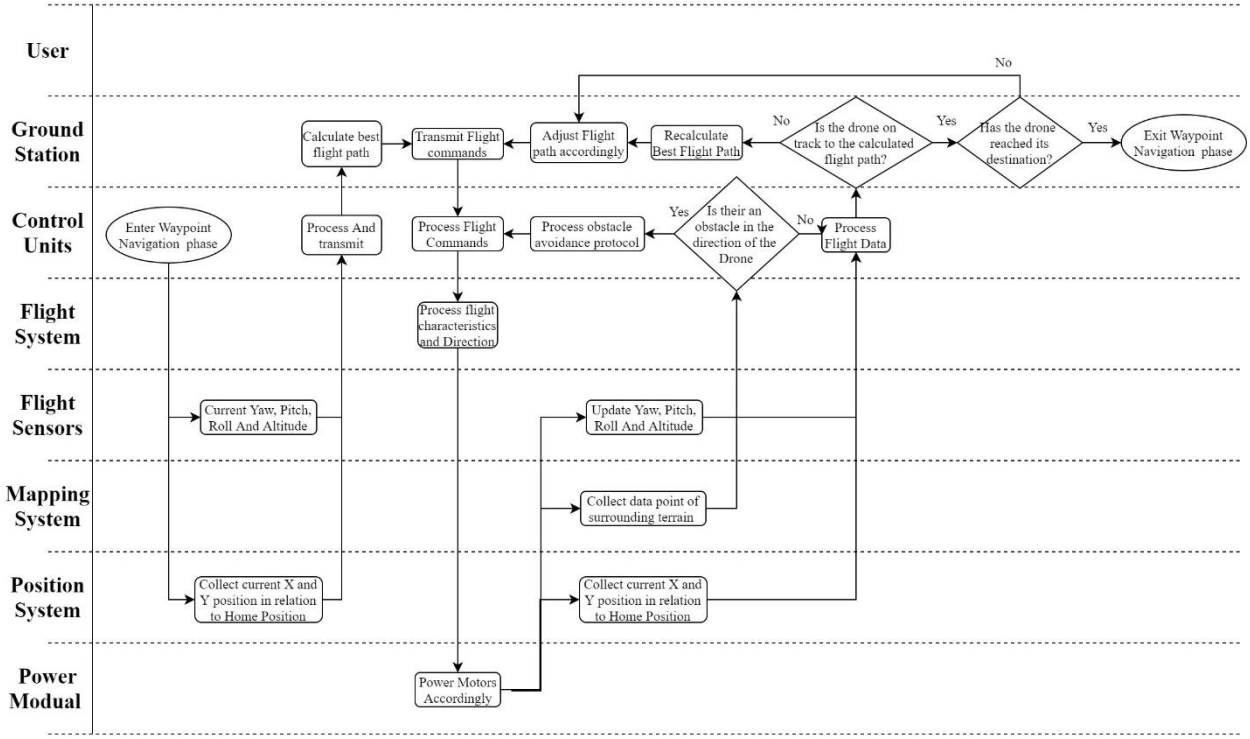
## *10.1 Use Case Phase: Initiation Phase*



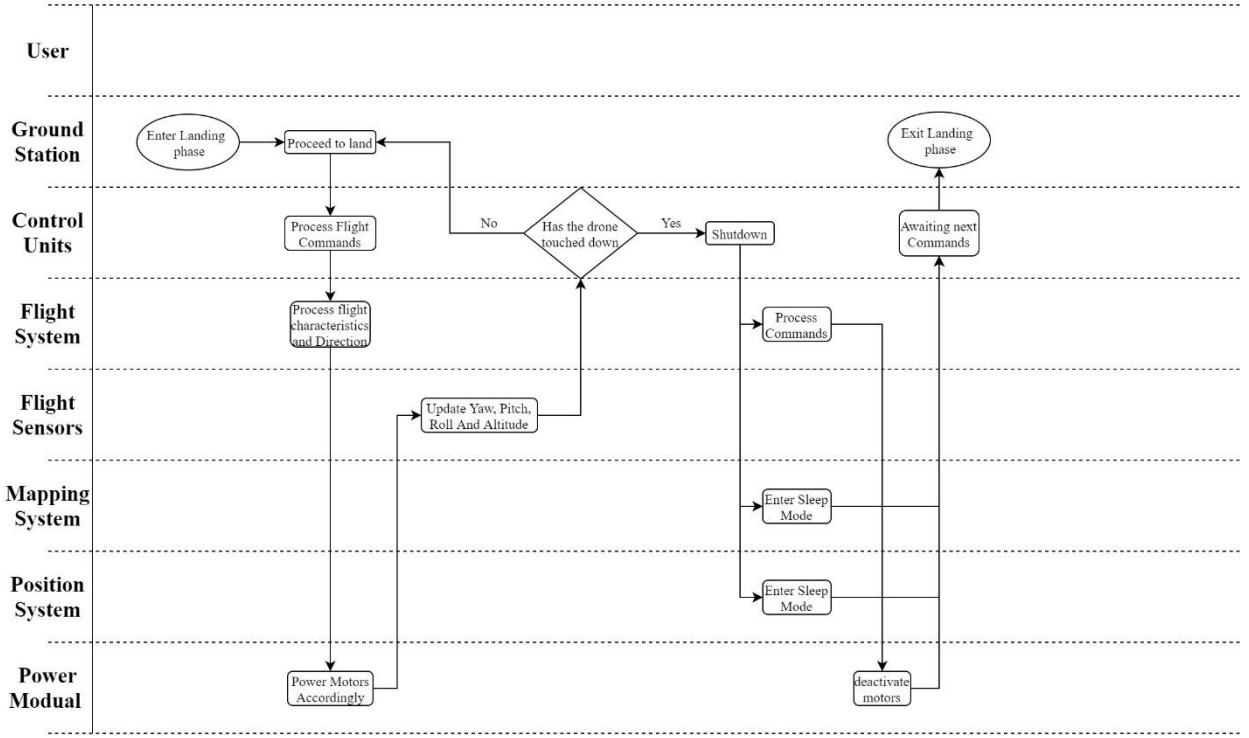

## *10.2 Use Case Phase: Flight Phase*



## *10.3 Use Case Phase: Waypoint Navigation Phase*




## *10.4 Use Case Phase: Landing Phase*



# 11. Project Conclusion

## *11.1 Results and Final Thoughts*


*Figure 38: Owl-EYE*

**In-Flight Scan**

Our final results of the 3d scan left a lot to still be worked out, although the accuracy of the scan came out fairly well the issue of the drone repeatedly losing itself made it difficult to produced reliable real-time scans. In addition, the tuning of the flight contained major instability making it hard to fly the drone in a stable manner. But despite all the issues the project resulted in a great success as a proof of concept, with better tuning and optimization as well as an upgrade to communications the drone would be a feasible device for not only disaster relief but many other industrial, and/or militaristic applications.

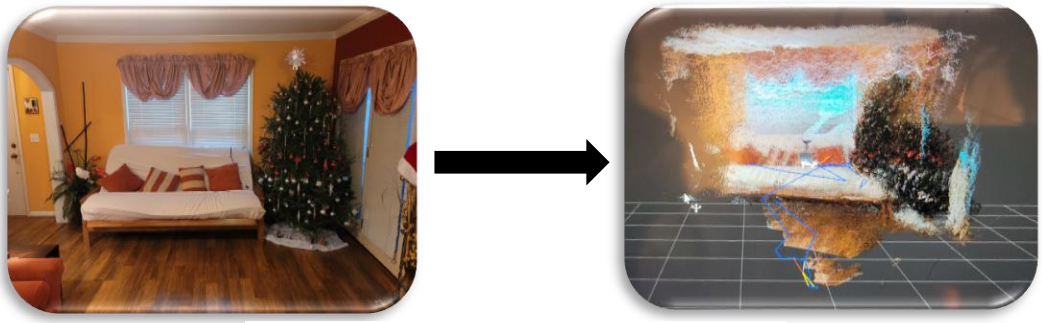


*Figure 39: Final Scan Results*

**Printed Circuit Board**

Our final version of the PCB had the corrections necessary to remove the XSHUT jumper cables. There was a small error where the voltage trace short circuited an XSHUT pin, but overall, the finished design was excellent. The board provided the ability to interface all these sensos in a small compact manner that allowed for placement at the belly of the drone. We are able to get sensor data from the five VL53 spot Lidars, CCS811 Air Quality, and BME280 Environmental Sensor. The buck converter takes in 5 volts from the Pi and outputs 3.3 volts which is used to power the board.

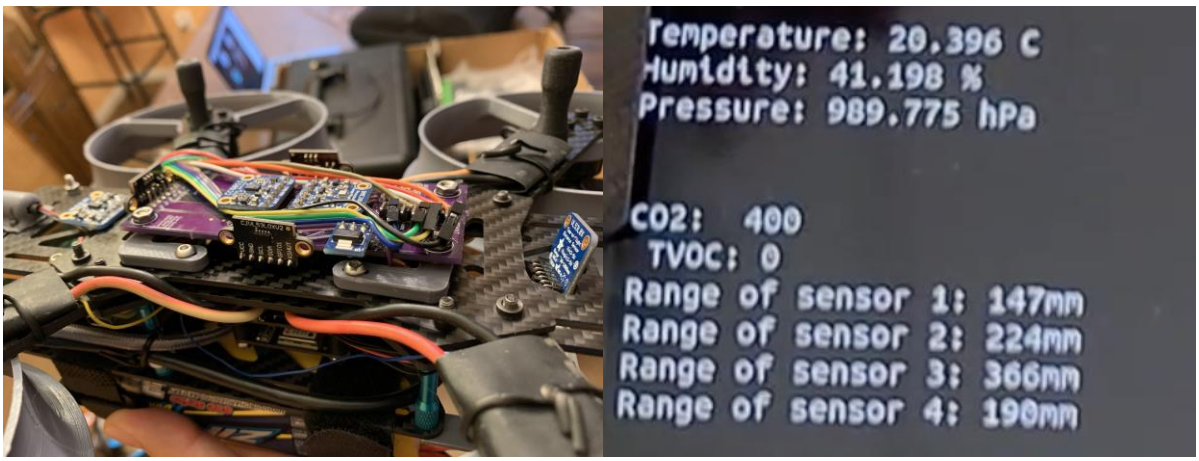



*Figure 40: Final PCB Results*

## *11.2 What we would have liked to do different?*

Something we would have wanted to do different is implement the VL53 spot lidars a lot earlier. We made the mistake of only interfacing a single VL53 and not all of them. This caused an issue when creating the PCB because we did not create an interface for the GPIO pins. This made us spend more money to order another PCB and also created some delay within the project.

Along with using different spot lidars, we would have liked to test Rtabmap and its dependencies earlier. Incompatibilities between packages and Rtabmap on the base station and the drone were the main reasons for not using our original base station in the final demo.

## *11.3 Future Goals/Aspirations*

We eventually plan to continue this project and turn it into an actual product that would be used by first responders. If we plan to do so, there are certain features we would like to improve. We would want a longer lasting battery for the drone with increased power inefficiency. By increasing the battery life, this provides first responders with longer flight times in-between battery changes. The longer the drone can stay in the air, the better but we do not want to increase the weigh too much. We also want better reliability of the data being sent to the drone while also improving communication. By improving the communication, we decrease latency and have clear stream of data from the drone module to the base station.

Another potential upgrade would be to replace the point lidars with a single spinning lidar. Having this one lidar would allow the drone to detect objects from all sides. We were unable to add this sensor to our project because it was not within our budget. Using the spinning lidar would have cost more in funds, weight, and development time than the spot lidars.

We would also like to add support for other wireless technologies. While scanning our test environment, we noticed the framerate for our camera feed was considerably low. Given the types of situations our drone would be used in, we would like to have our base station receive the most relevant data possible in order to allow the operator to act decisively. Adding a radio for private LTE would afford the operator more flexibility when he or she is trying to get the best wireless performance between the base station and the drone. Using a faster router and having a dedicated Wi-Fi adapter on both the base station and the drone.

Given the scale of our project, we tried to keep a low budget and purchase only what was necessary. If we were given an unlimited budget, there are several other features we would have liked to add. For starters, if the spinning lidar proved to be too heavy, we would like to have a total of six spot lidars, one for every direction. We would also probably go with a higher quality lidar as well to increase the accuracy and efficiency of our redundancy features. Also, we would change the layout of the PCB to incorporate more lidars and have less lidars connected to the auxiliary ports. By creating more space on the PCB, we would only need one lidar connected to the auxiliary ports and this creates less loose wire while also improving the overall look.


*Figure 41: added sensors*

With an increased budget, we would also like to include an oxygen sensor and an MQ2 which is able to detect LPG, I-butane, propane, methane, alcohol, Hydrogen, and smoke. All this extra information could paint a better picture of the environment and could notify Ground Station if there could possibly be an explosion. These extra modules would help responders plan a safer course of entry and exit to keep both civilians and responders as safe as possible.

# Index

## *Table of Figures*