Spring 2024

Autonomous Damage and Structure Scanning Drone

Natasha Ninan  
*The University of Akron*, nrn21@uakron.edu

Amber Long  
all132@uakron.edu

Lee Nestor  
ljn13@uakron.edu

Emmanuel Jensen  
evj4@uakron.edu

Follow this and additional works at: [https://ideaexchange.uakron.edu/honors_research_projects](https://ideaexchange.uakron.edu/honors_research_projects)

Part of the Electrical and Electronics Commons, Robotics Commons, and the Systems and Communications Commons

Please take a moment to share how this work helps you through this survey. Your feedback will be important as we plan further development of our repository.

**Recommended Citation**

[https://ideaexchange.uakron.edu/honors_research_projects/1861](https://ideaexchange.uakron.edu/honors_research_projects/1861)

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.
Autonomous Damage and Structure Scanning Drone

Senior Design Project Final Report

Design Team 12:

Natasha Rachel Ninan
Amber Long
Lee James Nestor
Emmanuel Jensen

Department of Electrical and Computer Engineering
College of Engineering and Polymer Science

Honors Project Sponsor: Dr. Mohammed Ali
Honors Faculty Advisor: Dr. Nghí Tran
Honors Project Reader One: Dr. Nghí Tran
Honors Project Reader Two: Dr. Michael J. Lichter

4/26/2024
Contents
1. Abstract .............................................................................................................................. 5
2. Problem Statement ........................................................................................................ 5
2.1. Need .............................................................................................................................. 5
2.2. Objective .................................................................................................................... 6
2.3. Marketing requirements ............................................................................................ 6
2.4. Background ............................................................................................................... 7
3. Engineering Analysis ...................................................................................................... 13
3.1. Drone .......................................................................................................................... 13
3.1.1. Components ......................................................................................................... 14
3.1.2. Flight Controller: MATEKSYS F405-VTOL ......................................................... 14
3.1.3. Battery .................................................................................................................. 15
3.1.4. Weight calculations ............................................................................................. 16
3.1.5. Thrust calculations ............................................................................................... 19
3.2. Objective ................................................................................................................... 19
3.2.1. Camera Module Version 2 .................................................................................. 20
3.2.2. Capturing and Storing Images ............................................................................ 21
3.2.3. TF-Mini-S LiDARs ............................................................................................... 22
3.2.3.1. UART enabled LiDAR ..................................................................................... 22
3.2.3.2. USB TTL Connected LiDAR ......................................................................... 22
3.2.3.3. Time of Flight (ToF) Principle ....................................................................... 23
3.2.3.4. Connections to Raspberry Pi .......................................................................... 23
3.3. Computer Networks .................................................................................................. 24
3.4. Photogrammetry ....................................................................................................... 25
3.4.1. What Is NeRF? .................................................................................................... 25
3.4.2. Theory .................................................................................................................. 26
3.4.3. Implementation .................................................................................................... 27
3.4.4. Hardware ............................................................................................................. 27
3.4.4.1. Software ......................................................................................................... 28
3.5. Structural Analysis with Finite Element Analysis .................................................... 28
3.5.1. What is it? .............................................................................................................. 28
3.5.2. Theory .................................................................................................................. 28
3.6. Flight Controller to Companion Computer Communication .................................... 30
3.6.1. What is it .............................................................................................................. 30
3.6.2. Theory .................................................................................................................. 30
3.6.2.1. Mavlink .......................................................................................................... 30
3.6.2.2. Ardupilot ........................................................................................................ 30
4. Engineering Requirements Specification ......................................................................... 31
4.1. Engineering Standards Specification ......................................................................... 32
4.1.1. Safety ................................................................................................................... 32
4.1.2. Communication .................................................................................................... 33
4.1.3. Data Formats ....................................................................................................... 33
4.1.4. Design Methods ................................................................................................... 33
4.1.5. Programming Language ........................................................................................ 34
4.1.6. Connector Standards ........................................................................................... 35
• Figure 5.3.2: Camera sample capture
• Figure 5.3.3: Benewake LiDAR test measurement
• Figure 5.3.4: LiDAR code first iteration
• Figure 5.3.5: LiDAR code second iteration output
• Figure 5.3.6: LiDAR code second iteration
• Figure 5.3.7: LiDAR code second iteration
• Figure 5.4: Level 0 Network Architecture
• Figure 5.5: User Interface Photo
• Figure 5.5.1: Radiance Field Of A HVAC System Pre Training
• Figure 5.5.2: Radiance Field Of A HVAC System mid Training
• Figure 5.5.3: Radiance Field Of A HVAC System Training Towards End
• Figure 5.5.4: COLMAP Generated Camera Positions for Images used for HVAC System
• Figure 5.5.5: MATLAB Code for Basic FEA work
• Figure 5.5.6: Mission Planner with Mission to Orbit Kolbe Hall
• Figure 5.5.7: Drone Kit Example Code for Take off
• Figure 5.5.8: Dronekit Example Code for Starting a Square Mission
• Figure 5.6: User Interface Code
• Figure 5.7: Command Line Handler Code
• Figure 5.8: Live Camera Feed Code
• Figure 5.9: Capture/Record Photo Code
• Figure 5.10: Level 1 User Program Software
• Figure 5.11: Level 1 Flowchart – Drone Processor Software
• Figure 6: Mechanical Sketch

List of Tables:
• Table 3.1: Battery comparisons
• Table 3.2: Weight calculations
• Table 3.3: Battery life calculation for flight time
• Table 3.4: Battery life calculations with equations
• Table 3.5: LiDAR and Raspberry Pi 3B+ Pins
• Table 3.7: Marketing Requirements
• Table 5.0: Level 0 Functional Requirements Tables
• Table 5.1: Level 1 Functional Requirements Tables – Camera and LiDAR Hardware
• Table 5.5: Level 1 User Program Software
• Table 5.6: Level 1 Functional Requirements – Data Processor Software
1. Abstract

Remote damage analysis plays a crucial role in lowering the risk associated with human presence at dangerous sites. This project focuses on developing a system for remote structural examination using photogrammetric techniques and analysis. The system utilizes a drone equipped with cameras for photogrammetry, and LiDARs for navigation. This setup can enable efficient structural model generation with the collection of images from multiple points. A novel structural analysis is performed to detect potential damage or points of failure in the structure.

Key components include a teleoperated drone, a software pipeline for photogrammetric analysis, collision protection mechanisms, and a user interface. The proposed design aims to improve remote structural assessment, enhancing safety and efficiency in hazardous environments.

2. Problem Statement

2.1. Need

According to the National Center for Biotechnology Information, the implementation of robotics in repetitive site work such as concrete drilling, drywall finishing, and damage detection decrease the number of hours spent of these tasks between 25% to 90% [1]. Simplifying the process of observing and recording damage of various objects around the site could increase task efficiency and decrease the environmental risks significantly.

In the case that the damage is in a dangerous locale or is a hazardous object itself, the automation of this process can produce an average of 72% reduction in time spent on a dangerous task [1]. In addition, accuracy can be increased by 55% and can reduce the need for
rework by over 50% [1]. An issue with a significant amount of these robotics, however, is their inability for collaboration with humans in the workspace [2]. Almost, if not all, of these automated are cumbersomely sized, lack capable close-quarter autonomous navigation, and produce noise or debris which makes the typical drone difficult to spend substantial amounts of time with outside of heavily industrialized areas.

Designing a collaborative, miniature drone for damage detection in industrial and commercial spaces could ideally decrease the need for human intervention in hazardous damage sites.

2.2. Objective

The project's objective is to design and prototype a small-scale drone for damage detection and image processing that can operate close-quarters navigation in an industrial zone and commercialized environments. The drone will be able to collect data on the affected area or object. Then, transmit the information to a central computer for processing and interpretation. The processed data will then display a before and after model to the user. Sensors will be used for navigation of the local environment and damage site analysis. The implementation of features such as safety, noise reduction and accuracy will enable the miniature drone to be operated around humans in non-industrial environments. In addition, teleoperation will allow for manual control which will be implemented using wireless communication.

2.3. Marketing requirements

- Decreasing the need for human presence in toxic or hazardous damage sites is the primary goal of designing a miniature damage detection drone.
• It will be able to navigate areas that are typically inaccessible to humans such as vaulted ceilings, which allows for easier maintenance and problem diagnosis.

• The robot will be designed to be safe to work around warehouse environments that include moving vehicles and people.

• With power being sourced from battery pack, power efficiency can be minimized, allowing it to potentially complete tasks at a faster rate.

• It will be able to create a 3-dimensional mesh of the object or area of interest for observation and analyzation.

2.4. Background

The theory behind the damage scanning drone is it will be a tool for factories or industrial areas, that will help to alleviate damage detection tasks from the common workload. It will have sensitive and accurate proximity detection sensors for navigation, one example is that of LIDAR, which is “remote-sensing technology that measures distance to a target.” as defined by G. Yang [1].

The LIDAR sensors will help with path navigation of the drone. Due to the innate complexity of these areas, some of them having hard to reach places or potential human obstacles, quick response time proximity sensing ensures proper and safe navigation. The drone can also be in hazardous environments, that pose a potential risk for humans to operate in. The proximity sensors can also help with navigating through potentially dangerous environments. Especially, in a factory setting where there could be dangerous gases, chemical spills, or simply just far-to-reach places. This device can deter workplace injuries and illnesses. It
can also be taken outdoors which can help with potential climate and terrain hazards.

The damage scanning drone will have a form of 3D-modeling in photogrammetry. The drone will be programed to take multiple photos of a targeted area. Based on the experiments conducted in the “3D Line Segment Based Model,” the number of images we may need can range from 11 to 100 images [2]. The images will be of various positions of the same object, to get a better scope of the object for processing. A computer is our best option for processing the data because of its storage capabilities.

The overarching concept is to have a drone that uses 3D modeling technologies to detect damage on a specified object. The drone can operate in enclosed spaces, hazardous environments and around people. The data can be sent to a computer to be processed using photogrammetry software. As drones are becoming more popular lately, it is safe to say there are other similar technologies out there.

According to both [3] and [4], the most common type of data collection is with that of Light imaging, Detection, and Ranging or LiDAR. This solution provides a large amount of depth data with high accuracy that also can provide images regardless of the homogeneity of various surfaces [2]. For indoor mapping and object scanning typically these tasks are either performed with a handheld device, a backpack mounted device or with a trolley [3]. These methods of data acquisition are entirely human driven and operated, requiring a significant amount of human input and skill to acquire quality data.

Due to the cost and technological difficulties and complexity associated with using LiDAR, there has been recent experimentation with using photogrammetry as a cost-effective and less bulky form of data collection and synthesis. Taking the images required for
photogrammetry is done manually or through basic automation. The data is then collected and
then processed into a 3D mesh and the photos taken are utilized to apply a texture to the mesh.
Photogrammetry does not give positional information, however, like a similar LiDAR system
can, so a separate system must be used, such as an example [1] using Time of Flight sensors for
positional calibration or in example [2] using GPS (Global Positioning Sys) for locational data
[3].

The concept of Unmanned Aerial Vehicle (UAV) assisted scanning is used in large,
outdoor spaces for scanning buildings, urban environments and more [2], but its use in smaller,
indoor spaces is currently being researched as well. When paired with photogrammetry as its
main method of data acquisition and processing, a lightweight drone can be used as the UAV.
This is due to a LiDAR based system being significantly heavier and requires a larger drone to
carry the system [2].

In Micro Electromechanical System (MEMS) gyroscopes, drifts are a major source of
error that can be caused due to thermal, vibrational, or mechanical stress issues. A paper
published in 2013 discusses the utilization of a Kalman filter to estimate and minimize the drift
from the gyroscope’s output. The proposed method was proven to compensate for drift and thus
increase the accuracy in the gyroscope’s output. In terms of limitations, the paper assumed linear
drift behavior which may not provide a clear process of the ability to handle nonlinear drift
behavior in gyroscope outputs. Further research needs to be done to predict nonlinear drift
behavior in gyroscope outputs [5].

Since navigation in an indoor environment requires many specialized sensors and
equipment to operate, this can strain payload capacity. For instance, the reconstruction of 3D
point clouds, which are typically used to generate 3D models or maps, requires 360 LIDAR for an accurate construction and the ability to match image content to a constructed map model. An increase in payload creates an increase in power consumption and a decrease in flight time, thus reducing the possible amount of data collected in one flight. It also leads to a significant increase in price as the hardware becomes more complex [6].

Current technology requires a significant amount of human intervention to operate and to get manageable results [6] naturally lends itself to human error and issues concerning reliability and replicability. According to [7], this leads to a 55% increase in accuracy and reduces the need for rework by 50%. Site work is also reduced between 25% and 90% in 12 cases of robot implementation [7].

With similarities there are aspects that are fundamental to the concept of photogrammetry and drone operation found in both the indoor and outdoor drone systems outlined in and [2]. A similarity between our drone and the indoor system is that they will both use calculated positions for which to position and face the drone to obtain a robust data set. They will also both use Ego-centric local mapping as its main method of navigation towards various set points.

With the outside system outlined in [2], another similarity is the use of key points. As stated by [2], “Tie points (or key points) are points of interest that can be easily recognized in many photos.” This combined with the positioning data can have the capacity to create accurate models and reduce computational time. A focus on high point density is also a similarity between our concept and [2], since higher point densities can allow smaller objects to be scanned. The differences between the systems prototyped in [1] and [2] primarily lie in the scope of the
projects and with that scope, another difference is what objectives they need to achieve. The indoor system [1]'s primary goal is to map and model an entire room while the main objective of our concept is to scan a specific object or structure of varying size amongst other properties accurately and efficiently.

Outdoor systems also have significantly different constraints and opportunities than that of an indoor system. Due to effects such as window and other weather conditions, that can affect the reliability of the platform and the quality of the data received, thus manual operation this the main method of navigation used. With outdoor systems as well, due to the innate inaccuracies of 9-DOF systems, they primarily rely on GPS [3] for positional data.

There exist systems for an “automatic aerial photographing method for arbitrary-view-point rendering.” [1] They use multiple drones to take photos of an object at specific coordinates. We will be using one drone to take multiple images of an area. In the patent, ”it is determined which portion of the subject lacks color information from the ratio of color information recorded on each face of the hexahedron set.” [1] They use exposure information to determine where their drone will move.

Whereas compared to the damage scanning device, the device moves to a specified location and takes pictures of said area. The user will determine the size of the damage and the device will take pictures accordingly. The patented device has a flight formation for dynamic subjects. [1] The device also has a separate setting for stationary objects. The damage scanning device is focused on stationary to limited movement type of objects.

There are existing systems, like ours, for “computer-based image processing.” [2] The invention focuses on “restoring a 3D scene from a 2D image.” [2] The patent uses a hands-on
approach. They have the user label images to regions and consistent points. Then, use the computer to automatically build the model. Most of the calculations are done by hand then plugged into the software.

The damage scanning device uses a different approach. The patented invention does most of the calculations by hand. The scanning device will have the data sent to the computer. That data will be processed using algorithms. In the patent, it mentions using 3 different points to take the picture to simulate 3 dimensions. [2] The scanning device will also be using a similar method when capturing the specified area. The device will use more than 3 points when scanning a damaged area. The goal is to make it easier for factory workers and engineers to use.
3. Engineering Analysis

3.1. Drone

Motors and propellers were utilized to lift the drone off the ground. The Camera and two LiDARs will provide real-time video footage and distance measurement, aiding in navigation and altitude ranging. Using the video and images captured by the camera, the three-dimensional model will be generated and sent to the user interface. For manual control, the user will be able to control the drone using a game controller.

(Figure 3.0: Drone with assembled components)
3.1.1. Components
Main system components include a Raspberry Pi 3B+, Flight Controller (F405-VTOL), four AKK 2205 brushless motors, four ESCs (Electronic Speed Controllers), two LiDARs, Camera, and a 4S battery pack.

3.1.2. Flight Controller: MATEKSYS F405-VTOL
MATEKSYS F405-VTOL is a lightweight flight controller that weighs 25g and supports 25x25 and 35x35 mounting patterns for easy integration into drone systems. It features a comprehensive set of functionalities, including a Barometer, six UARTS, and a Power Distribution Board (PDB) for five individual ESC/Motors. With a power input range of 6.8-30V DC IN, it can support a 2-6S LiPo.

(Figure 3.1: MATEKSYS F405-VTOL)
3.1.3. Battery

The Size, Weight, and Power (SWaP) requirements were determined based on the AKK motors' capabilities. Each AKK motor can handle approximately 744g of thrust, and since there are four motors in total, the combined total thrust capability is 3096g. This calculation led to the selection of a 4S battery to meet the SWaP requirements for the drone design. Two battery models will be discussed for the purpose of the project. The HRB 4S LiPo Battery was utilized for testing purposes while the Zeee 4S LiPo Battery is the final intended battery for operation purposes. The table below compares their specifications.

<table>
<thead>
<tr>
<th>Battery</th>
<th>Zeee 4S LiPo 9000mAh</th>
<th>HRB 4S LiPo 3000mAh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td><img src="image1" alt="Zeee 4S LiPo 9000mAh" /></td>
<td><img src="image2" alt="HRB 4S LiPo 3000mAh" /></td>
</tr>
<tr>
<td>Weight (g)</td>
<td>725</td>
<td>297</td>
</tr>
<tr>
<td>Capacity (mAh)</td>
<td>9000</td>
<td>3000</td>
</tr>
<tr>
<td>Voltage (V)</td>
<td>14.8</td>
<td>14.8</td>
</tr>
<tr>
<td>Plug</td>
<td>EC5 connector</td>
<td>T plug connector</td>
</tr>
</tbody>
</table>

(During the drone design testing phase and performance evaluations, using a lighter battery helps in achieving a better thrust-to-weight ratio, which is crucial for assessing flight dynamics, maneuverability, and overall performance. The 3000mAh battery was intended solely for testing purposes while the Zeee 4S LiPo Battery is the final intended battery for operation purposes. The table below compares their specifications.)
purposes prior to receiving the 9000mAh battery. In addition, it allowed for preliminary testing of the drone's flight dynamics and PID tuning in a cohesive manner, ensuring that the drone's performance could be thoroughly evaluated and optimized.

In the final drone design, the decision to incorporate the Zee 4S Lipo 9000mAh battery stems from its advantageous features that align well with the intended purpose of navigating and scanning an area in an industrial environment. Firstly, the higher capacity of 9000mAh offers the capability for longer flight times, a critical factor for extended missions or tasks that require sustained flight operations without the need for frequent battery changes. This extended endurance is particularly beneficial in industrial settings where continuous aerial surveillance and mapping is necessary.

Despite its slightly heavier weight compared to the 3000mAh battery, the 9000mAh variant still maintains a manageable weight while providing a higher discharge rate of 100C. This combination enhances the drone's agility, maneuverability, and overall performance during high-demand maneuvers, ensuring reliable operation in challenging industrial environments. Overall, the Zee 4S Lipo 9000mAh battery emerges as the preferred choice for the final drone design due to its ability to deliver longer flight times, optimized performance, and suitability for the specific requirements of navigating and scanning areas in industrial settings.

3.1.4. Weight calculations

Each AKK 2205 brushless motor provides a maximum thrust of 774g. The design includes four motors which combine to provide a total thrust of 3096g. In consideration of the drone's overall weight, the individual components contribute to a total weight of 344.87g without a battery. This
includes the weight of the AKK motors, along with other essential components, such as the Raspberry Pi and Flight controller. The tables below depict the calculations.

With the addition of a 3000mAh battery, the total weight is 643.87 g. Similarly, with a 9000mAh battery, resulting in a total weight of 1069.87 g. The drone's weight remains well below the maximum thrust capacity of 3096g provided by the motors, ensuring a safe margin for flight operations.

This well-balanced design optimizes performance while maintaining a safe margin for flight, allowing for extended flight times without compromising the thrust-to-weight ratio or flight dynamics. Overall, this cohesive approach ensures reliable and stable aerial operations.
<table>
<thead>
<tr>
<th>Component</th>
<th>Quantity</th>
<th>Individual Weight (g)</th>
<th>Total Weight (g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi 3B+</td>
<td>1</td>
<td>58.97</td>
<td>58.97</td>
</tr>
<tr>
<td>F405-VTOL</td>
<td>1</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>TF-Mini-S LiDARs</td>
<td>2</td>
<td>5.3</td>
<td>10.6</td>
</tr>
<tr>
<td>Raspberry Pi Camera board v2</td>
<td>1</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>GT-U7 GPS Module GPS Receiver</td>
<td>1</td>
<td>5.7</td>
<td>5.7</td>
</tr>
<tr>
<td>35A ESCs</td>
<td>4</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>AKK 2205 brushless motor</td>
<td>4</td>
<td>28.8</td>
<td>115.2</td>
</tr>
<tr>
<td>HRB 4S LiPo Battery 3000mAh</td>
<td>1</td>
<td>299</td>
<td>299</td>
</tr>
<tr>
<td>Zee 4S Lipo Battery 9000mAh</td>
<td>1</td>
<td>725</td>
<td>725</td>
</tr>
<tr>
<td>225mm FPV Racing Drone Frame Carbon Fiber</td>
<td>1</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td><strong>Total Weight without battery</strong></td>
<td></td>
<td></td>
<td>344.87</td>
</tr>
<tr>
<td><strong>Total Weight with 3000 mAh battery</strong></td>
<td></td>
<td></td>
<td>643.87</td>
</tr>
<tr>
<td><strong>Total Weight with 9000 mAh battery</strong></td>
<td></td>
<td></td>
<td>1069.87</td>
</tr>
</tbody>
</table>

(Table 3.2: Weight calculations)
3.1.5. Thrust calculations

During the investigation into flight time, two scenarios were explored. With an HRB 3000mAh 4S battery, operating at 15% throttle allows for approximately 19 minutes of flight time, while max throttle usage results in about 3 minutes of flight time. In comparison, a Zee e\textsuperscript{4}S 900mAh battery provides around 24 minutes of flight time at 35% throttle and 8 minutes at max throttle.

<table>
<thead>
<tr>
<th>Battery</th>
<th>Capacity</th>
<th>Minimum Flight Time (Minutes, Throttle Percentage)</th>
<th>Max Flight Time at Max Throttle (100%), Minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRB 4S</td>
<td>3000 mAh</td>
<td>18.5 @ 15%</td>
<td>2.681</td>
</tr>
<tr>
<td>Zee e\textsuperscript{4}S</td>
<td>9000 mAh</td>
<td>23.768 @ 35%</td>
<td>8.043</td>
</tr>
</tbody>
</table>

(Table 3.3: Battery life calculation for flight time)

<table>
<thead>
<tr>
<th>Battery</th>
<th>Maximum Flight Times (Minutes @ Min Throttle)</th>
<th>Minimum Flight Times (Minutes @ Max Throttle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRB 4S</td>
<td>44.4/144 = 18.5</td>
<td>44.4/993.6 = 2.681</td>
</tr>
<tr>
<td>Zee e\textsuperscript{4}S</td>
<td>133.2/336 = 23.786</td>
<td>133.2/993.6 = 8.043</td>
</tr>
</tbody>
</table>

(Table 3.4: Battery life calculations with equations)

3.2. Proximity sensing

In the drone design, one camera and two LiDAR sensors are crucial components for photogrammetry and proximity sensing, respectively. The primary camera captures high-resolution images for detailed mapping and can provide real-time video feedback to the operator. Future iterations include implementing a stereo camera to enhance photogrammetry capabilities.
The addition of the flight controller with inertial sensors, such as IMUs, enhances the drone's sensing system by providing vital data on orientation and motion. This integration ensures precise control and navigation during flight operations.

Altitude measurements and obstacle detection are facilitated by two LiDAR sensors strategically placed on the drone body. This setup enables accurate ranging and obstacle avoidance, contributing to safe and efficient flight maneuvers.

A key consideration in sensor selection is the SWaP (Size, Weight, and Power) requirements, emphasizing components below 100g to meet payload constraints without compromising performance. Additionally, leveraging open-source software like ROS (Robot Operating System) or Raspberry Pi programming allows for flexible sensor integration and advanced functionality, enhancing the overall capabilities of the drone system.

### 3.2.1. Camera Module Version 2

The Raspberry Pi Camera Module v2 is connected to the board via the CSI port. During the submodule demonstration, the Camera Module v2 was utilized to obtain real-time video feedback. The next step in submodule development involves using Python to transmit video data via a shared network. This includes writing Python scripts for capturing frames with the camera and a shared server that will take the input frames. Since Raspberry Pi boards include a Camera Serial Interface (CSI) port, the camera can be directly connected with the board.
3.2.2. Capturing and Storing Images

As part of the camera code and process, the captured .jpg images are stored in a folder named "Sensors" on the Raspberry Pi board. The Sensors folder includes the text files generated from the LiDAR codes and the images from the camera. The folder serves as a central repository for storing captured images. Additionally, the folder was later modified to host a shared folder for the seamless transmission of files to the user interface. This streamlined approach facilitates efficient data management and accessibility, ensuring that captured images are readily available for analysis and visualization.
3.2.3. **TF-Mini-S LiDARs**

The drone's proximity sensing relies on two LiDAR sensors for accurate distance measurements from objects. These LiDAR sensors are connected differently: one through GPIO pins and the other via USB-TTL, enabling versatile data acquisition.

3.2.3.1. **UART enabled LiDAR**

In the proposed design, the TF Mini S LiDAR operates in UART mode via GPIO14 and GPIO15 pins for communication. UART (Universal Asynchronous Receiver-Transmitter) communication allows for serial data transfer between the LiDAR and the Raspberry Pi 3 B+. This direct communication method offers low-latency data exchange, making it suitable for real-time applications such as drone navigation and obstacle detection. By leveraging UART communication, the LiDAR provides accurate distance measurements and reliable data feedback to enhance the drone's operational capabilities.

3.2.3.2. **USB TTL Connected LiDAR**

In the proposed design, an additional TF Mini S LiDAR is connected to the Raspberry Pi via USB TTL connectivity. The TX and RX lines of the programmer are utilized for asynchronous serial communication, transmitting data between the LiDAR and the computer via USB. On a Linux-based system like the Raspberry Pi, the USB port for the LiDAR is typically named /ttyUSB0. Using USB TTL connectivity was chosen for its ease of differentiation while obtaining distance data, as demonstrated in the accepted technical design LiDAR code section.
### 3.2.3.3. Time of Flight (ToF) Principle

The TF Mini S LiDAR operates based on the Time of Flight (ToF) principle, a technique used for measuring the time taken for light to travel to a target and back. This principle enables the LiDAR to accurately calculate distances by analyzing the time delay between emitted and received light signals. In the context of drone systems, the ToF principle plays a crucial role in altitude ranging and obstacle detection. By accurately measuring distances, the LiDAR helps drones navigate complex environments with precision, avoiding collisions and ensuring safe flight paths. The ToF technology enhances the overall performance and reliability of drone systems, making them more efficient and capable in various applications, including structural examination, mapping, and surveillance.

### 3.2.3.4. Connections to Raspberry Pi

For GPIO connections to the Raspberry Pi, the specific pins are listed in the table below:

<table>
<thead>
<tr>
<th>TF-Mini-S Pin</th>
<th>Raspberry Pi 3 B+ Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5V</td>
<td>Pin 2</td>
<td>Supplies 5 volts of power to the LiDAR sensor</td>
</tr>
<tr>
<td>GND</td>
<td>Pin 6</td>
<td>Completes the circuit by connecting to the ground of the LiDAR sensor.</td>
</tr>
<tr>
<td>TXD</td>
<td>Pin 8, GPIO 14</td>
<td>Facilitates data transmission from the Raspberry Pi to the LiDAR sensor.</td>
</tr>
<tr>
<td>RXD</td>
<td>Pin 10, GPIO 15</td>
<td>Manages data reception from the LiDAR sensor to the Raspberry Pi.</td>
</tr>
</tbody>
</table>

(Table 3.5: LiDAR and Raspberry Pi 3B+ Pins)
The second LiDAR sensor was connected to the Raspberry Pi using USB TTL, which stands for Universal Serial Bus - Transistor-Transistor Logic. It utilizes the same pins as listed in the table above: 5V for power, GND for ground, TXD for transmitting data, and RXD for receiving data. The figure below depicts the LiDAR connections.

(Figure 3.3: LiDAR connections)

### 3.3. Computer Networks

The implementation of the communication network will follow the format of having a server act as an intermediary between the drone and client for communications to help with data
organization and storage. The drone will handle collecting the data and communicate with the server for incoming commands. Server will receive commands from the processing base station and then send them to the drone. The server will also handle the processing and receiving the unprocessed data from the drone. The client will receive transmission of video footage from the drone and receive the result of the photogrammetric synthesis.

For a communication protocol for the interactions between the various participants in the computer network, the protocol of WebSocket will be used. It is a communication protocol over the web focused on keeping a two-way communication channel also known as a duplex communication channel. It is based around handshakes and TCP communication underpinning it. This will be necessary as the data is important and cannot afford to be dropped.

In order to properly process the images into a functional 3-Dimensional representation of the desired object, software will need to be used and integrated to do so.

3.4. Photogrammetry
To properly process the images into a functional 3-Dimensional representation of the desired object, software will need to be used and integrated to do so. The software will have to be able to take a large number of images taken by the drone and be able to handle it in a timely manner. Optimization techniques will have to be employed to reduce computational time and provide information back to the user, albeit in a coarser form if the user’s time is of the utmost importance.

3.4.1. What Is NeRF?
NeRF, or Neural Radiance Fields, is a photogrammetric technique used to reconstruct complex three-dimensional scenes from a set of two-dimensional images. It is implemented through
functions whose parameters are generated using a network trained from sparse photos taken of a desired scene. NeRF enables the generation of scene geometry and the reflectance properties of the scene.

(Figure 3.4.1: A Radiance Field pre-training)

(Figure 3.4.2: A Radiance Field of a Coke Can. Notice handling of Reflectivity)

3.4.2. Theory

NeRF, or Neural Radiance Fields, performs view synthesis by directly optimizing parameters of
a continuous 5D scene representation to minimize rendering errors from captured images. This representation models a static scene as a continuous function that outputs radiance $C(r)$ and density $\sigma$ at each point in space $(x,y,z)$ and viewing direction $(\theta,\phi)$. Implemented as a deep neural network, NeRF regresses from a 5D coordinate $(x,y,z,\theta,\phi)$ to volume density $\sigma$ and view-dependent RGB color $c$. The rendering process involves marching camera rays through the scene, using neural network predictions to accumulate colors and densities into images through volume rendering techniques. NeRF’s differentiable nature allows for optimization using gradient descent, minimizing errors across multiple views to create coherent scene models.

### 3.4.3. Implementation

To reduce computational time and cost, Instant NeRF is employed, utilizing a smaller network without sacrificing quality to improve speed and reduce floating-point and memory access operations. It employs a multiresolution hash table containing feature vectors optimized through stochastic gradient descent. The software used for this implementation was developed by engineers at NVIDIA.

### 3.4.4. Hardware

NeRF implementation requires Nvidia CUDA for optimal performance. It is optimized to run on a singular GPU, necessitating the use of Nvidia's proprietary CUDA cores. CUDA cores are specialized processing units found in NVIDIA GPUs (Graphics Processing Units) designed to handle parallel processing tasks. Unlike traditional CPUs (Central Processing Units), which typically consist of a few powerful cores optimized for sequential processing, GPUs contain many smaller and more efficient CUDA cores optimized for parallel computations. These cores
can execute multiple tasks simultaneously, making them highly efficient for tasks that can be parallelized, such as graphics rendering, scientific simulations, and machine learning. CUDA cores are programmable and can be utilized by developers through NVIDIA's CUDA (Compute Unified Device Architecture) platform, allowing for the acceleration of a wide range of applications by leveraging the massive parallel processing power of GPUs.

### 3.4.4.1. Software

An essential aspect of training the software is having a reference for the camera's position when the photo was taken. To obtain this necessary position and angle information, a program called COLMAP is used. COLMAP is a general-purpose Structure-from-Motion (SfM) and Multi-view Stereo (MVS) pipeline. It can be utilized with ordered and unordered image collections to generate information about the camera state per picture.

### 3.5. Structural Analysis with Finite Element Analysis

#### 3.5.1. What is it?

Finite Element Analysis (FEA) is a computational technique employed for structural analysis, enabling the examination of complex structures through computational methods. It serves as a pivotal tool for simulating phenomena such as heat deformation and stress distribution within structures. The method operates by breaking down intricate structures into smaller, more manageable components known as finite elements. These elements are individually analyzed to provide insights into the behavior of the overall system.

#### 3.5.2. Theory

The theoretical foundation of FEA lies in the finite element method (FEM), a numerical
technique used to solve partial differential equations governing the behavior of systems. By discretizing space into smaller elements, the method transforms complex differential equations into linear forms, often polynomials. These equations are then solved using linear algebraic methods, allowing for the prediction of system behavior with a high degree of accuracy.

Finite Element Analysis (FEA) operates on the principle of discretization, where complex structures are subdivided into smaller, interconnected elements. Each element represents a simplified portion of the overall structure and is characterized by a set of governing equations derived from the original differential equations. These equations encapsulate the behavior of the element under various loading conditions, such as mechanical stress or thermal gradients. By applying boundary conditions and material properties to these elements, FEA transforms the original problem into a system of algebraic equations. Solving these equations yields numerical approximations of the system's response, providing insights into critical parameters like stress distribution, deformation patterns, and natural frequencies. Through iterative refinement and analysis of the results, engineers and researchers can optimize designs, predict failure modes, and ensure structural integrity across a wide range of applications.
3.6. Flight Controller to Companion Computer Communication

3.6.1. What is it
Communication between the flight controller, responsible for governing the operations and stability of a UAV drone, and the companion computer, which handles data collection and processing through the network, is crucial for seamless collaboration. This interaction allows the companion computer to relay instructions to the flight controller regarding the drone's desired position and actions, enabling coordinated flight operations.

3.6.2. Theory

3.6.2.1. Mavlink
Mavlink is a messaging protocol specifically designed for communication with drones and between drone components. It features a hybrid publish-subscribe communication pattern as well as point-to-point communication capabilities. Data streams in the form of topics, where data is published and subscribed to by other components, or through direct communication lines between specific points, are integral to Mavlink. All messages are formatted using XML files, ensuring a standardized dialect system.

3.6.2.2. Ardupilot
Ardupilot is an open-source firmware system that enables the autonomous operation of unmanned vehicles. It supports advanced navigation, autonomy, and control for various vehicles such as planes, copters, rovers, and submarines. In the context of this system, Ardupilot provides interfaces for controlling and interfacing with drones, along with support for various peripherals including GPS, accelerometers, gyroscopes, magnetometers, rangefinders, and cameras.
### 4. Engineering Requirements Specification

<table>
<thead>
<tr>
<th>Marketing Requirements</th>
<th>Engineering Requirements</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3,4</td>
<td>Be able to balance weight of camera and sensors on drone.</td>
<td>Having an unbalanced distribution of weight can cause unnecessary wear to the motors as well as creating an unstable platform.</td>
</tr>
<tr>
<td>1,2,3</td>
<td>Utilize a flight controller as the central hardware device.</td>
<td>A flight controller allows for communication between the drone and an outside device, allowing for remote control.</td>
</tr>
<tr>
<td>1,2,3</td>
<td>Must be able to process image and sensor data.</td>
<td>Drone would be rendered useless if sensors are not utilized</td>
</tr>
<tr>
<td>1,3</td>
<td>Must be able to present data as a 3D model.</td>
<td>Data presented in a 3D model will allow users to visually inspect the scanned object in a remote location.</td>
</tr>
<tr>
<td>1,3</td>
<td>Must be able to present video footage in real time.</td>
<td>Real time feedback will allow users to navigate and control the device remotely</td>
</tr>
<tr>
<td></td>
<td>Must allow the user to take a picture or scan the area.</td>
<td>Drone must be able to receive the command to begin scanning the area it is in.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1,2,3</td>
<td>Must be able to receive data from drone.</td>
<td>Data produced from drone is effectively useless if nothing happens to it.</td>
</tr>
<tr>
<td>1,3</td>
<td>Must be able to operate under self-power for no less than 30 minutes.</td>
<td>Battery power will allow the drone to remain untethered and increase its range.</td>
</tr>
</tbody>
</table>

**Marketing Requirements**

1. Product must be user friendly.

2. Product must be self-contained with remote control capabilities.

3. Product must produce data and be able to transmit it.

4. Product must be stable while flying.

(Table 3.7: Marketing Requirements)

### 4.1. Engineering Standards Specification

#### 4.1.1. Safety

Safety will be recognized by creating a safe working environment while creating the drone and will also be one of the main efforts whilst flying the drone, with the sensors being able to help avoid collisions with people or objects. Safety will be up to the end user as the drone will be flown by that user, although the device will be able to send warnings of potential collisions.
4.1.2. Communication
The data needs to be communicated between the recipient client and the drone without any loss. There also needs to be minimal latency between the display for the drone operator and the camera used for manual navigation for it to be operable. The drone must have protocols in place if it were to lose communication with the main client and vice versa.

4.1.3. Data Formats
For the requisite images needed for performing the photogrammetric processing, the data from the camera sensors will be received and processed as jpegs. This data format was chosen as the desired input because it is lightweight concerning storage size and increasing the speed of transmission and processing. For output, the models would be output as obj files since this is an extremely popular file format that is supported in various environments in JavaScript.

4.1.4. Design Methods
For the Analysis and Navigation subsystems of the software, these subsystems were designed with a top-down block design. Starting with overarching ideas, requirements, and goals, these were broken down into individual functional blocks. Individual blocks are then broken down into smaller blocks until implementation is feasible.

The UI was broken down into hardest tasks first. The first task was establishing a connection between the Raspberry Pi and the host computer. Next, file transfer between the devices. The following was setting up a live camera feed and capturing photos. Next, display the 3D model and finally, UI for navigation. Navigation is last because it requires the drone to be built and flying to test.
4.1.5. Programming Language

The programming language used depends on the implementation environment, such as whether it is in the embedded system or the subsystem responsible for processing.

For the embedded system, the languages used would be Python. This is due to the fact that main concerns with developing a system like this is not that of processing power nor speed, but instead of ease of implementation. A system that is more feasible to implement takes precedence over optimizations at this stage. A significant amount of the software packages that would be desirable for this project have python implementations as well that can help contribute for a cohesive ecosystem.

For the processing subsystem, Python or C++ would be utilized. Since processing will involve more complicated processes such as photogrammetric synthesis and interpretation it will be important to get a baseline implementation for reference. Python would be extremely important for getting some form of functionality and providing a baseline. If speed is a present issue, then reimplementing in C++ would help with that.

Listed Below are other languages and programs that are going to be used. Along with a brief description of why they will be helpful:

1. Front-End
   a. Visual Studio C#
   b. Can integrate with most 3D libraries and frameworks like Unity3D and Unreal Engine, allowing for efficient 3D modeling.
   c. Support for multithreading and asynchronous programming, crucial for handling concurrent tasks associated with 3D modeling and live camera feed processing.
d. Can integrate well with the PowerShell command line.

4.1.6. Connector Standards
For connectors, they will need to be able to support the requisite power draw of the motors. It will also need to support the data transfer speeds required for communication between the sensors and the main processor. They will also need to be tolerant to failure and fatigue given the fragility of the drone itself.

5. Accepted Technical Design
   5.1. Overview
Below is the Level 0 block diagram of the overall system. This diagram goes over how the system will operate. The base station or user program will be the central processing unit of the overall system. The base station will take the data from the Camera and LiDAR units and process the data. Then, convert the data into a 3D model. We will go into further detail on the user program in the Software design section.

The processor within the drone will be handling the hardware and directing the proper signals and commands to their proper devices. Another aspect of the Drone Shell is the flight controller, motors, camera, LiDAR and power. We will go into detail on that within the hardware design section.

Listed below are the level n diagrams along with the hardware and software design of the system.
Level 0 – Overall Block Diagram

<table>
<thead>
<tr>
<th>Module name</th>
<th>User Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input arguments</td>
<td>User Input, Peripheral Data</td>
</tr>
<tr>
<td>Output arguments</td>
<td>3D Model, Commands, Server Connection</td>
</tr>
<tr>
<td>Description</td>
<td>This unit does all the processing for the data. The drone sends data to be processed and presented to the user here.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Raspberry Pi model 3B+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Flight controller F405-VTOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input arguments</td>
<td>Pi communicates commands via UART protocol</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Pulse Width Modulation (PWM) signal, UART commands</td>
</tr>
<tr>
<td>Description</td>
<td>The Flight controller controls and connects to the motors. The data is sent to the Raspberry Pi model 3B+ for subsequent processing at the base station.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Raspberry Pi model 3B+, Motors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Cameras</th>
</tr>
</thead>
</table>

(Figure 5.0: Level 0 Block Diagram)
<table>
<thead>
<tr>
<th>Input arguments</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output arguments</td>
<td>Image Data</td>
</tr>
<tr>
<td>Description</td>
<td>Will have cameras to take pictures of an object. Then, data is sent over a shared network to the base station.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Raspberry Pi model 3B+, User program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>LiDAR Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input arguments</td>
<td>UART, USB-TTL</td>
</tr>
<tr>
<td>Output arguments</td>
<td>LiDAR Data</td>
</tr>
<tr>
<td>Description</td>
<td>Will have LiDAR Sensors to get altitude data. Then, send data to the Raspberry Pi model 3B+.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Flight controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>AKK 2205 brushless motor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Hardware</td>
</tr>
<tr>
<td>Input arguments</td>
<td>DC Power</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Speed of Propellers</td>
</tr>
<tr>
<td>Description</td>
<td>Controls the flight of the system</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Flight Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input arguments</td>
<td>Commands</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Pulse-width modulation (PWM) signal</td>
</tr>
<tr>
<td>Description</td>
<td>Controls flight of the drone. Regarding speed, positioning, connecting motors, etc...</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Raspberry Pi</td>
</tr>
</tbody>
</table>

5.2. Hardware Design

5.2.1. Proximity Sensing

The primary proximity sensing design objectives are as follows:

- Sensors below 100g to meet drone payload requirements.
- Maximum supply voltage of 12 V
- Able to program with open-source software.
(Figure 5.1: Level 1–Hardware system diagram)
5.2.2. Functional Requirements

<table>
<thead>
<tr>
<th>Module name</th>
<th>Input arguments</th>
<th>Output arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi model 3B+</td>
<td>User Input and Commands</td>
<td>Sensor data, user commands via UART to Flight controller, I2C Commands to sensors</td>
<td>The Raspberry Pi model 3B+ oversees calibrating and connecting to the hardware such as the cameras. It communicates with the flight controller via the UART protocol to control the motors and LiDAR sensors.</td>
</tr>
<tr>
<td>LiDAR</td>
<td>I2C commands</td>
<td>LiDAR Data</td>
<td>Will have four LiDAR Sensors to get data of area such as the angle and distance. Then, the data is sent to the processor.</td>
</tr>
<tr>
<td>Camera</td>
<td>CSI</td>
<td>Image Data</td>
<td></td>
</tr>
</tbody>
</table>
### Description
Will have cameras to take pictures of objects. Then, send data to the Raspberry Pi model 3B+.

<table>
<thead>
<tr>
<th>Modules invoked</th>
<th>Drone Processor</th>
</tr>
</thead>
</table>

(Table 5.1: Level 1 Functional Requirements Tables – Camera and LiDAR Hardware)

#### 5.2.3. Camera Module Version Two

##### 5.2.3.1. Code Discussion

The camera code begins by importing the necessary libraries, including PiCamera from picamera for interfacing with the Raspberry Pi Camera Module v2 and datetime from datetime for generating timestamps. An instance of the PiCamera class is created, which sets the stage for subsequent camera operations. The start_preview() method is called to initiate the camera preview, providing real-time feedback on what the camera is capturing. A timestamp in the format of "YYYY-MM-DD_HH-MM-SS" is generated using str(datetime.now()). The capture() method is used to capture high-resolution images. The generated timestamp is automatically included in the filename, ensuring each image is uniquely identified. To conserve resources, the stop_preview() method is called, halting the camera preview when not needed. The code and a sample image have been depicted below.

(Figure 5.3.1: Camera capture code)
(Figure 5.3.2: Camera sample capture)
5.2.4. **TF-Mini -S LiDARs**

5.2.4.1. **Testing with Benewake Test Software**

Before integration, the LiDAR sensor's functionality was tested using the manufacturer’s (Benewake) test software. This allowed for thorough validation of its distance measurement capabilities. The figure below illustrates the application of Time of Flight (ToF) technology in measuring distances. The x-axis denotes the number of data points collected during LiDAR scanning, while the y-axis represents distances in centimeters (cm). This graphical representation showcases the LiDAR sensor's ability to accurately determine object distances by emitting laser pulses and measuring their round-trip time. Each data point on the graph corresponds to a specific distance measurement, providing a clear depiction of spatial awareness crucial for drone navigation and obstacle detection.

(Figure 5.3.3: Benewake LiDAR test measurement)
5.2.4.2. LiDAR code implementation

The code was developed using Python. Data from the TFMini-S LiDAR sensor was obtained using the serial library for serial communication. The TfMini class was utilized to handle interactions with the LiDAR sensor. This class manages initialization, data retrieval, logging, and serial port closure. The `get_data` method reads data from the serial port, extracts crucial distance and signal strength values, and then returns these values for further processing.

Moreover, the `data_thread` method continuously gathers data from the LiDAR sensor at predefined intervals. The distance and timestamp data are logged into designated lists (`dist` and `timex`).

Additionally, a `print_data_thread` method continuously displays the obtained distance and signal strength data from the LiDAR sensor, aiding in real-time monitoring and diagnostics. The first iteration of the code outputs the distance in centimeters and the signal strength.

(Figure 5.3.4: LiDAR code first iteration)
The second iteration outputs the distance in centimeters, the timestamp and signal strength

(Figure 5.3.5: LiDAR code second iteration output)
```python
import serial
import time
import matplotlib.pyplot as plt
import datetime as dt

dist=[]
timeplot=[]
timex=[]

class TfMini():
    def __init__(self, serial_port):
        self._ser = serial.Serial(serial_port, 115200)
        if self._ser.is_open == False:
            self._ser.open()
        self._distance = 0
        self.distance_min = 10
        self.distance_max = 1200

    def get_data(self):
        time0 = time.time()
        while True:
            count = self._ser.in_waiting
            distance = -1
            if time.time() > time0 + 1: break
            if count > 8:
                recv = self._ser.read(9)
                self._ser.reset_input_buffer()
                if recv[0] == 0x59 and recv[1] == 0x59:  # python3
                    self._ser.reset_input_buffer()
                    break
            self._distance = distance
            self._strength = strength
            self._time = time0
            return(distance, strength)

@property
    def distance(self):
        return(self._distance)

    def data_thread(self):
        while True:
            distance = self.get_data()
            timeplot=dt.datetime.now().strftime('%H:%M:%S')
```

(Figure 5.3.6: LiDAR code second iteration)
5.3. Software Design

5.3.1. Network Architecture

One of the main considerations in terms of the network is what shape and form the architecture would be. This means dictating what paths the data would follow and through which nodes or devices. There are such constraints as latency, processing power, and user interface to determine what part of the process gets executed where. In which case, after observing the resources each problem requires, the following decisions are made about that. Time sensitive components such as teleoperation of the drone in the case of manual operation require the client and the drone to talk directly to each other which no in-between processing to minimize latency. Given the nature
of photogrammetric processing, presume that user accessing and using the software is equipped with the most processing power available, then processing the data can be delegated to a server or otherwise known as the processing base station. General commands, such as status updates on the system are not time critical, nor are they computationally intensive, so they can route through the server for organizational sake. Here is a diagram representing these pathways.

(Figure 5.4: Level 0 Network Architecture)

(Figure 5.5: User Interface Photo)
5.3.2. Application / User Interface

The primary objectives of the application design are as follows:

- Efficiently process data and images obtained from the drone.
- Convert the collected data into a 3D model for visualization.
- Display the 3D model to the user in an intuitive and user-friendly manner.
- Provide a live camera feed from the drone for real-time monitoring within the application.

The connection at the start of the two flowcharts below Figure 5.9 and Figure 5.10 are connected via Wi-Fi. Originally using clients and servers via C#. Larger files could not be sent in the original way. Using the ‘rsync’ command in the command line was a better approach to sending all the files. The other UI features used the command line as well. The code is more cohesive in its features. The file transfer between the drone and the user program is using TCP protocols via Sockets. Listed below is the Command Line Code for the file transfer. All the front-end/base station codes are using C# within Visual Studio.

The ‘rsync’ command in the User Interface is button labeled “Sync Files.” This feature allows the user to synchronize the files from a specific location on the Raspberry Pi to a specific location on the Base Station’s local hard drive. ‘Rsync’ is a Linux command, this project was done on a Windows laptop. To get this command working the Windows Subsystem for Linux (WSL) needs to be installed along with changing a few settings. Inbound and Outbound rules need to be changed for Port 22 as well as virtualization settings.

With the command line setup up, we can Secure Shell (SSH) into the Raspberry Pi and operate the Pi remotely. The next task is to automate this process using C#. Listed below is the code for
the Command Line. Using SSH requires a username and password. To automate this process, a ‘key’ was generated and put into the Pi’s files to recognize the device. When SSH-ing into the Pi, it will automatically sign you in. This is a crucial step because the command line C# code does not prompt for username, password, or yes/no questions. If that were to come up while trying to synchronize the files, then the command would time out because it could not connect to the Raspberry Pi.

The live camera feed and capture photo is another feature that uses the command line. This feature uses SSH to connect to the Pi then runs the python file on the Pi. For the live feed, the python file uses an API called Flask. Flask is a web framework that helps with web development, in this case Flask is being used to host a local server on port ‘5000.’ Where the camera data stream is being consistently updated and sent to the server, creating a live feed. The live feed is a web address, the C# code opens the browser and displays the feed. The code for the live feed is also listed below.

The final features using the command line are the 3D display and Fly Drone option. The Fly Drone button prompts the command line for the user, which would invoke the Mavlink software that controls the drone. The 3D Model button uses an ‘invoke-item’ command which opens a file using the default method. Windows has a built in 3D object viewer called 3D Viewer. The command uses the 3D viewer application to view the chosen object.

5.3.3. NeRF Photogrammetry
5.3.3.1. Implementation

Instant NeRF introduces several innovations to improve the efficiency of NeRF while maintaining quality. One key aspect is the utilization of a multiresolution hash table, a data
structure that stores feature vectors representing local scene properties at different levels of detail. This allows for efficient storage and retrieval of scene information during both training and inference. Feature vectors are compact representations of scene properties learned by the neural network, facilitating quick access and manipulation of scene data. Additionally, Instant NeRF incorporates stochastic gradient descent, an optimization technique that randomly samples a subset of training data in each iteration to compute an estimate of the gradient. By employing these methods, Instant NeRF reduces computational time and memory requirements without sacrificing quality. This combination of innovations enables Instant NeRF to achieve faster inference and training speeds compared to traditional NeRF implementations.

(Figure 5.5.1: Radiance Field Of A HVAC System Pre Training)
5.3.3.2. **Software - COLMAP**

(Figure 5.5.2: Radiance Field of an HVAC System mid training)

(Figure 5.5.3: Radiance Field of an HVAC System Training Towards End)
Structure-from-motion (SfM) is a technique used in computer vision to reconstruct 3D structures of objects or scenes from a collection of 2D images. The process involves estimating the camera poses and 3D structure of the scene simultaneously by analyzing the correspondences between features in multiple images. Multi-view stereo (MVS) is a complementary technique that extends SfM by refining the reconstruction to produce dense 3D point clouds or meshes from the sparse point clouds generated by SfM. COLMAP, a popular software package, combines both SfM and MVS into a unified pipeline. It first performs SfM to estimate camera poses and sparse point clouds, then refines the reconstruction using MVS to generate dense 3D representations. This pipeline enables COLMAP to reconstruct detailed 3D models of scenes or objects from a set of 2D images, making it a powerful tool for applications such as 3D modeling, augmented reality, and robotics.

(Figure 5.5.4: COLMAP Generated Camera Positions for Images used for HVAC System)

5.3.3.3. Installation and Setup

For Instant NeRF implementation, the setup involves installing and configuring both the NeRF
software and COLMAP. COLMAP is used to provide camera position and angle information for training the NeRF model.

5.3.4. Structural Analysis using Finite Element Analysis

5.3.4.1. Implementation

In terms of implementation, FEA is commonly executed using software such as MATLAB, renowned for its versatility in engineering tasks, including structural analysis. MATLAB offers specialized toolkits like the Partial Differential Equation Toolbox, which streamline the setup and solution of structural analysis problems. This software provides a user-friendly interface for arranging finite element models and solving them efficiently.

5.3.4.1.1. Matlab Code

(Figure 5.5.5: MATLAB Code for Basic FEA work)
5.3.4.2. Installation and Setup

To utilize FEA effectively, installation and setup typically involve acquiring MATLAB and installing the necessary add-on packages, such as the Partial Differential Equation Toolbox. Once installed, engineers and researchers can readily employ FEA to conduct comprehensive structural analyses, making informed decisions about the design and performance of complex systems.

5.3.5. Flight Controller To Companion Computer Interface

5.3.5.1. Implementation

5.3.5.1.1. Mission Planner

Mission Planner is ground control station software designed for configuring and controlling unmanned aerial vehicles equipped with Ardupilot firmware. It offers a user-friendly interface for creating missions, planning flight routes by setting waypoints, defining actions, and specifying parameters such as altitude and speed. Mission Planner supports various flight modes, telemetry data display, sensor calibration, PID tuning, pre-flight checks, mission scripting, and automation through MAVLink commands.
5.3.5.1.2. **Drone kit**

DroneKit is an open-source software development kit (SDK) used for building applications that interact with UAVs using Python. It provides APIs for communicating with drones, sending commands, and receiving telemetry data. DroneKit supports a wide range of UAV platforms, including those running ArduPilot firmware, offering high-level abstractions for common UAV operations such as mission planning, waypoint navigation, and vehicle control. It enables developers to access flight parameters, create custom behaviors, and integrate with other systems through standard communication protocols like MAVLink.
5.3.5.1.2.1. Drone kit Example Code

```python
def arm_and_takeoff(targetAltitude):
    # Arms vehicle and fly to targetAltitude.

    print("Basic pre-arm checks")
    # Don't let the user try to arm until autopilot is ready
    while not vehicle.is_armable:
        print("Waiting for vehicle to initialise...")
        time.sleep(1)

    print("Arming motors")
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed:
        print("Waiting for arming...")
        time.sleep(1)

    print("Taking off!")
    vehicle.simple_takeoff(targetAltitude) # Take off to target altitude

    # Wait until the vehicle reaches a safe height before processing the goto
    # after Vehicle.simple_takeoff will execute immediately.
    while True:
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt >= targetAltitude*1.095:
            print("Altitude reached!")
            break
        time.sleep(1)
```

(Figure 5.5.7: Drone Kit Example Code for Take off)
5.3.5.2. **Installation and Setup**

Installation and set-up require an installation of the prerequisite packages and requirements before enabling use of the software. For the drone kit, it already came with a python implementation and as such can be installed only with the Python Install Package manager or PIP with ease. The Mission Planner software can be installed using the already built Windows System Installer file set up for installation in a windows environment.
Main code using C# in Visual Studio

```csharp
public partial class Form1 : Form
{
    private int photoCount = 0;
    private bool isFeedOn = false;
    private readonly CLIManager cli = new()

    //Raspberry Pi Login Info
    private const string USER_PI = "user"; //personal PI = "raspberry" SDP PI = "user", SDP2.0 = "uber"
    private const string IP_ADDRESS_PI = "192.168.33.3"; //Personal IP = 192.168.0.169 Other Ip's: AmborHome = "192.168.0.135"; edu
    private const string PASSWORD_PI = "rpi"; //personal and SDP2.0 = "password" SDP = "rpi"

    //CLI Commands
    private const string RSYNC = "rsync -uv --delete + USER_PI + "@" + IP_ADDRESS_PI + ":/\home\mark\MyWork\SDP_DroneData";
    private const string CAM_CAPTURE = "python Documents/code/capture.py";
    private const string START_STREAM = "python Documents/code/canStream.py";
    private const string STOP_STREAM = "kill -9 \"python Documents/code/canStream.py\"";
    private const string URL_STREAM = "start http://" + IP_ADDRESS_PI + ":5000/stream";
    private const string PROMPT_CLI = "start powershell";
    private const string MODEL = "invoke-item \""; //\c:\\SDP_DroneData\\3DModels\\3DModelTest.obj\"

    1 reference
    public Form1()
    {
        InitializeComponent();
        this.FormClosing += Form1_FormClosing;
    }

    /**
     * Synchronizes the local and remote directory as specific locations.
     * Note: Goes off the premises there's an ssh key in place
     */
    1 reference
    private void btnSync_Click(object sender, EventArgs e)
    {
        lblSync.Text = "File Processing...";
        string output = cli.Command(RSYNC);
        lblSync.Text = "Sync Completed";
        txtbSync.Text = output;
    }
```
/**
 * Connects to Pi via SSH and Runs the Python Script for capturing photos
 */
private void btnCam_Click(object sender, EventArgs e)
{
    lblCam.Text = "Cam Processing";
    // Sign into Pi via SSH
    using (var client = new SshClient(IP_ADDRESS_PI, USER_PI, PASSWORD_PI))
    {
        client.Connect();
        client.RunCommand(CAM_CAPTURE); // Run PI Python Script
        client.Disconnect();
    }
    photoCount++;

    lblCam.Text = photoCount + " Photo Captured";
}

/***
 * Browse The File Directory and Display the image
 */
private void btnBrowse_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        txtBrowse.Text = openFileDialog1.FileName;
    }
}
private void btnStream_Click(object sender, EventArgs e)
{
    lblWeb.Text = "Feed Status: Active";
    Thread thread = new Thread(new ThreadStart(this.StreamTask));
    thread.SetApartmentState(ApartmentState.STA);
    thread.IsBackground = true;
    thread.Start();
    cli.Command(URL_STREAM);
}

private void StreamTask()
{
    isFeedOn = true;

    // Sign into Pi Via SSH and Run Script
    using (var client = new SshClient(IP_ADDRESS_PI, USER_PI, PASSWORD_PI))
    {
        client.Connect();
        client.RunCommand("START_STREAM");
        client.Disconnect();
    }
}
/** *
 * Close camera feed stream
 */

private void StopStream()
{
    if (isFeedOn)
    {
        // Sign into Pi Via SSH and Run Script
        using (var client = new SshClient(IP_ADDRESS_PI, USER_PI, PASSWORD_PI))
        {
            client.Connect();
            client.RunCommand(STOP_STREAM);
            client.Disconnect();
        }
        lblWeb.Text = "Feed Status: Inactive";
        isFeedOn = false;
    }
}

/** *
 * Close stream if running on exit.
 */

private void Form1_FormClosing(object sender, EventArgs e)
{
    StopStream();
}

/** *
 * Display 3D Model and stops live stream if running
 */

private void btnModel_Click(object sender, EventArgs e)
{
    // CLIHandler cli_Model = new CLIHandler();
    cli.Command(MODEL + txtBrowse.Text + "\"");
    StopStream();
}

/** *
 * Opens a Powershell command prompt in another window and display Instructions on connecting to drone.
 */

private void btnFlight_Click(object sender, EventArgs e)
{
    cli.Command(PROMPT_CLI);

    // Powershell instructions to connect to drone
    MessageBox.Show("To connect to drone please enter: msl; ssh " + USER_PI + "@" + IP_ADDRESS_PI + "\nNavigate to drone controls file and run file using: python filename.py");
}

(Figure 5.6: User Interface Code)
Command Line Code

Command Line Handler in C# using Visual Studio:

```csharp
using System.Management.Automation;
using System.Text;

namespace SDPUI
{
    internal class CLIHandler
    {
        private readonly PowerShell cli;
        internal CLIHandler()
        {
            cli = PowerShell.Create();
        }

        public string Command(string script)
        {
            string errorMsg = string.Empty;
            cli.AddScript(script);
            cli.AddCommand("Out-String");
            //Collects output results
            IAsyncResult result = cli.BeginInvoke<PSObject, PSObject>(null, outputCollection);
        }
    }
}
```
Camera Code

Live camera feed python file:

```python
import io
import picamera
from flask import Flask, Response

app = Flask(__name__)

def generate_frames():
    # Generates camera/image quality specs
    with picamera.PiCamera() as camera:
        camera.resolution = (640, 480)  # Options 1280x720, 640x480
        camera.framerate = 30
        stream = io.BytesIO()

        # Continuously process the frames
        for _ in camera.capture_continuous(stream, 'jpeg', use_video_port=True):
            stream.seek(0)
            yield b'--frame
Content-Type: image/jpeg\r\n\n' + stream.read() + b'\r\n'
            stream.seek(0)
            stream.truncate()

@app.route('/stream')
def stream():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, threaded=True)
```

(Figure 5.8: Live Camera Feed Code)
Capture/Record photo python file:

```python
from picamera import PiCamera
from time import sleep
from datetime import datetime

camera = PiCamera()
camera.start_preview()
camera.capture('/home/user/Sensors/image_%s.jpg' % (str(datetime.now())))
camera.stop_preview()

camera = PiCamera()
camera.start_preview()
camera.start_recording('/home/user/Sensors/video_%s.h264' % (str(datetime.now())))
sleep(120) # wait 2 min
camera.stop_recording()
camera.stop_preview()
```

(Figure 5.9: Capture/Record Photo Code)
## Level 1 User Program

(Figure 5.10: Level 1 User Program Software)

### Functional Requirements Tables Level 1 User Program

<table>
<thead>
<tr>
<th>Module name</th>
<th>User Connects to Application and Drone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Start</td>
</tr>
<tr>
<td>Input arguments</td>
<td>none</td>
</tr>
<tr>
<td>Output arguments</td>
<td>none</td>
</tr>
<tr>
<td>Description</td>
<td>The user/server connects to the drone</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>User flies to destination</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>User flies to destination</th>
</tr>
</thead>
</table>

65
<table>
<thead>
<tr>
<th>Module type</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input arguments</td>
<td>none</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Navigation commands</td>
</tr>
<tr>
<td>Description</td>
<td>Navigates drone to position of object</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>User presses scan area or take photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>User presses scan area or take photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Decision</td>
</tr>
<tr>
<td>Input arguments</td>
<td>none</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Take photo or scan command</td>
</tr>
<tr>
<td>Description</td>
<td>Users choose to take photos or scan an area. Sends the command to drone.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Process Photo Data, Process LiDAR data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Process Photo Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Process</td>
</tr>
<tr>
<td>Input arguments</td>
<td>Command</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Image Data</td>
</tr>
<tr>
<td>Description</td>
<td>Receives the photo files from the drone and sends them to 3D modeling Module.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Process Sensor Data (LiDAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Process</td>
</tr>
<tr>
<td>Input arguments</td>
<td>Command</td>
</tr>
<tr>
<td>Output arguments</td>
<td>LiDAR Data</td>
</tr>
<tr>
<td>Description</td>
<td>Receives the LiDAR files from the drone and sends them to 3D modeling Module.</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Storage</td>
</tr>
<tr>
<td>Input arguments</td>
<td>LiDAR and Camera data</td>
</tr>
<tr>
<td>Output arguments</td>
<td>none</td>
</tr>
<tr>
<td>Description</td>
<td>The base station’s local hard drive</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>3D Model Processing</td>
</tr>
</tbody>
</table>
### Module name: 3D-Modeling Process

**Module type:** Process  
**Input arguments:** Peripheral Data  
**Output arguments:** 3D Model  
**Description:** Converts the peripheral data into a 3D model.  
**Modules invoked:** Display Model

### Module name: Display Model

**Module type:** End  
**Input arguments:** Processed Data  
**Output arguments:** 3D Model  
**Description:** Displays the 3D model for the user  
**Modules invoked:** none

---

**Level 1 Software - Drone Processor**

(Figure 5.11: Level 1 Flowchart – Drone Processor Software)
### Functional Requirements Tables Level 1 – Drone Processor

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drone connected to base station</td>
<td>Processor connects to Server on base station.</td>
</tr>
<tr>
<td>Start</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect navigation cameras</td>
<td>Connects the streaming camera to user interface.</td>
</tr>
<tr>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td></td>
</tr>
<tr>
<td>Video stream to user program</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait for command</td>
<td>Waits for commands from base station could be navigation command or a take photo command</td>
</tr>
<tr>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>Command from base station/server</td>
<td></td>
</tr>
<tr>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is navigation command?</td>
<td>Checks which command was sent then sends to proper task.</td>
</tr>
<tr>
<td>Decision</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Whether to flight controller for navigation or cameras for a photo.</td>
</tr>
<tr>
<td>I2C signal</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drone takes phot and does scan, Navigate to destination</td>
<td></td>
</tr>
<tr>
<td>Module name</td>
<td>Drone takes photos and does the scan</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Module type</td>
<td>Process</td>
</tr>
<tr>
<td>Input arguments</td>
<td>Command</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Camera data, LiDAR data</td>
</tr>
<tr>
<td>Description</td>
<td>Takes a photo and sends the LiDAR data</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Stores the peripheral data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Stores the peripheral data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Data</td>
</tr>
<tr>
<td>Input arguments</td>
<td>Camera data and LiDAR data</td>
</tr>
<tr>
<td>Output arguments</td>
<td>none</td>
</tr>
<tr>
<td>Description</td>
<td>Store the data on processor memory</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Wait until docked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Wait until docked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>Process</td>
</tr>
<tr>
<td>Input arguments</td>
<td>none</td>
</tr>
<tr>
<td>Output arguments</td>
<td>none</td>
</tr>
<tr>
<td>Description</td>
<td>Waits until device has been docked</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>Send data to Base Station/Server</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module name</th>
<th>Send data to Base Station/Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module type</td>
<td>End</td>
</tr>
<tr>
<td>Input arguments</td>
<td>Notification of docked device</td>
</tr>
<tr>
<td>Output arguments</td>
<td>Camera data LiDAR data</td>
</tr>
<tr>
<td>Description</td>
<td>Sends the data back to server/base station</td>
</tr>
<tr>
<td>Modules invoked</td>
<td>none</td>
</tr>
</tbody>
</table>

(Table 5.6: Level 1 Functional Requirements – Data Processor Software)
6. **Mechanical Sketch**

(Figure 6: Mechanical Sketch)

7. **Team Information**

Amber Long, Computer Engineering

Lee James Nestor, Computer Engineering/Applied Mathematics

Natasha Ninan, Electrical Engineering
Emmanuel Jensen, Computer Engineering

8. **Budget analysis**

In the fall semester of 2023, the project’s initial phase began with a proposal submission, accompanied by the procurement of essential design components totaling $237.90. However, as the project progressed into the spring semester of 2024, it became apparent that the NewBeeDrone body and propellers were inadequate for mounting the Raspberry Pi 3B+ due to weight and thrust considerations. This prompted the acquisition of a 9000mAh battery and a new set of motors and propellers, with the latter being purchased by another team member independently.

Additionally, an extra stereo camera was procured for testing purposes during this phase. However, due to challenges encountered during flight dynamics testing, particularly concerning cable compatibility and configuration complexities, the camera was ultimately not mounted on the drone. These purchases and decisions reflect the iterative and problem-solving nature of the project, where adjustments and acquisitions were made to meet performance requirements and technical constraints.

The total expenditure during the spring semester of 2024 amounted to $276.11, combining with the fall 2023 expenses to yield an overall project cost of $514.01.
<table>
<thead>
<tr>
<th>Qty.</th>
<th>Part Num.</th>
<th>Description</th>
<th>Suggested Vendor</th>
<th>Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B07BDR5 PDW</td>
<td>Raspberry pi 3 B+</td>
<td>Amazon</td>
<td>$49.95</td>
<td>$49.95</td>
</tr>
<tr>
<td>2</td>
<td>TFmini-s</td>
<td>TFmini-s Lidar Sensor</td>
<td>Amazon</td>
<td>43.99</td>
<td>$87.98</td>
</tr>
<tr>
<td>1</td>
<td>B0B9ML S51W</td>
<td>Lithium Battery Protection Board</td>
<td>Amazon</td>
<td>13.99</td>
<td>$13.99</td>
</tr>
<tr>
<td>1</td>
<td>21549</td>
<td>NewBeDrone FLOW</td>
<td>getFPV</td>
<td>45.99</td>
<td>$45.99</td>
</tr>
</tbody>
</table>

### Fall 2023

**Total** $514.01

<table>
<thead>
<tr>
<th>Qty.</th>
<th>Part Num.</th>
<th>Description</th>
<th>Suggested Vendor</th>
<th>Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IMX219-83</td>
<td>IMX219-83 Stereo Binocular Camera Sensor Module</td>
<td>Amazon</td>
<td>$56.99</td>
<td>$56.99</td>
</tr>
<tr>
<td>1</td>
<td>B0953NN 3CW</td>
<td>chenyang Flat Slim FPC</td>
<td>Amazon</td>
<td>7.99</td>
<td>$7.99</td>
</tr>
<tr>
<td>1</td>
<td>N/A</td>
<td>FPVDrone 225mm FPV Racing Drone Frame Carbon Fiber</td>
<td>Amazon</td>
<td>34.99</td>
<td>$34.99</td>
</tr>
<tr>
<td>1</td>
<td>UCT_B01 2001</td>
<td>Arducam Multi Camera Adapter Module</td>
<td>Amazon</td>
<td>49.99</td>
<td>$49.99</td>
</tr>
<tr>
<td>2</td>
<td>B0CT2G G2LG</td>
<td>waveshare CSI FPC Flexible Cable 22Pin to 15Pin Cable</td>
<td>Amazon</td>
<td>$17.98</td>
<td>$17.98</td>
</tr>
<tr>
<td>2</td>
<td>LA094</td>
<td>Onyehn FFC Cable Flex Cable 15Pin to 22Pin 16CM and 30CM Ribbon Cable</td>
<td>Amazon</td>
<td>$15.18</td>
<td>$15.18</td>
</tr>
<tr>
<td>1</td>
<td>B07YP73 LMX</td>
<td>Zee 4S Lipo Battery 9000mAh 14.8V 100C RC Lipos EC5 Connector with Metal Plates</td>
<td>Amazon</td>
<td>$92.99</td>
<td>$92.99</td>
</tr>
</tbody>
</table>

**Spring 2024**

**Total** $514.01

9. **Conclusions and Recommendations**

In conclusion, the success of the project during the spring semester of 2024 was facilitated by maintaining consistent communication, taking notes, collaborating on responsibilities, and documenting work effectively. These practices ensured clarity and efficiency in project execution, contributing significantly to achieving project milestones. The project's success was also attributed to the diligence of the team members who consistently met deadlines and
contributed effectively to project tasks.

For future design iterations, integrating the 9000mAh battery with the Battery Management System can extend operational durations. The inclusion of a stereo camera is recommended for capturing higher quality 3D models. Another design improvement would be enabling gas leak detection capabilities, as suggested by some visitors during demonstrations.

Lastly, we express our sincere gratitude to Dr. Lichter, Dr. Ali, and Dr. Tran for their support and guidance throughout the project. Their expertise and encouragement were instrumental in navigating challenges and achieving project objectives.

10. References


Patents

