Crop Surveying UAV

May 1st, 2023

Client: Don Roe

Team Members: Nathaniel Dersom, Connor Williams, Morgan Maurer, Noah Stephens, Garett Fry, Isaac Harley, Caleb Syler, Luke Shoen, and Owen Paulus

Faculty/Advisors: Dr. Jose Oommen, Dr. Yuan Meng, Dr. Patrick Majors, and Dr. Baonhe Sob

Abstract

The Mount Vernon Nazarene University engineering seniors were tasked with building an unmanned aerial vehicle (UAV) to survey crops for farmers in Guatemala. Don Roe, director of the AgInno Institute in Chisec, Guatemala, expressed a need for an unmanned aerial system to reduce the time and energy that farmers spend observing the health of their crops on a regular basis. The engineering senior design team quickly decided to design and build a fixed wing aircraft equipped with a multispectral imagery system and autonomous flying capabilities. After several months, the team successfully built a custom fixed wing aircraft that met all of the given requirements. The body of the aircraft was designed from scratch using computer aided design software and was built using a combination of balsa wood and composite material. The imagery system offers a dual camera setup with a wavelength range from 400 nm to 700 nm and 730 nm to 950 nm. The aircraft was also equipped with image processing software, a battery safety monitoring system, and fully autonomous flying capabilities. Alongside the UAV itself, the team designed a handheld weather station to monitor the weather conditions prior to a given flight. The UAV also comes with an instructions manual that explains how to operate the aircraft safely. This report explains in detail the process taken to design the UAV as well the lessons learned throughout the year.

Acknowledgements

The team would like to thank each faculty advisor: Dr. Jose Oommen, Dr. Yuan Meng, Dr. Patrick Majors, and Dr. Baonhe Sob, for all of their guidance and support throughout the entirety of this project. Each member extends their thanks to the faculty for their time they have invested in every student over the past four years. The team would also like to thank Dr. LeeAnn Couts, Sheryl Arden, Justin Longfellow, and the Department of Natural and Social Sciences. Sheryl Arden deserves much thanks for her work in handling each of the team's purchase orders and Dr. Couts for allowing the opportunity to learn and display the skills learned throughout this project to the School. The guidance that Justin gave during the building phase of the project is also very appreciated. Lastly, the team's gratitude goes towards Don Roe and Aaron Aude. The team is thankful to Don for allowing them this opportunity and for sponsoring the project as each member gained great experience in a variety of areas within the realm of engineering. Aaron Aude offered his time and expertise on unmanned aerial vehicles. His willingness to help the team throughout the year is deeply appreciated as the project may not have been successfully completed without his generous support.

Table of Contents

List of Abbreviations and Nomenclature

Table 1. List of Abbreviations

Abbreviation	Expansion
API	Application Programming Interface
BOM	Bill of Materials
CAD	Computer Aided Design
Cd	Drag Coefficient
CFD	Computational Fluid Dynamics
Cl	Lift Coefficient
CM	Compute Module
EPO	Expanding Polyolefin
EPP	Expanding Polypropylene
ESC	Electronic Speed Controller
FC	Flight Controller
FCC	Federal Communications Commision
FOV	Field of View
FPV	First Person View
GB	Gigabyte

Abbreviation	Expansion
GNDVI	Green Normalized Difference Vegetation Index
GPIO	General Purpose Input-Output
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HAT	Hardware Attached Top
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IR	InfraRed
LiPo	Lithium-Polymer (Battery)
MCU	Microcontroller Unit
NACA	National Advisory Committee for Aeronautics
NDRE	Normalized Difference Red Edge
NDVI	Normalized Difference Vegetation Index
NIR	Near InfraRed
NTSC	National Television Standard Committee
ODM	Open Drone Map (Software)
OS	Operating System

Abbreviation	Expansion
OSAVI	Optimized Soil-Adjusted Vegetation Index
OSD	On Screen Display
PAL	Phase Alternate Line
PCB	Printed Circuit Board
PPM	Pulse Position Modulation
PWM	Pulse Width Modulation
QBC	Quad Bayer Coding
RC	Radio Controlled
RAM	Random Access Memory
RH	Relative Humidity

Abbreviation	Expansion
RGB	Red, Green, Blue
RGGB	Red, Green, Green, Blue
RPI	Raspberry Pi
SPD	Spectral Power Distribution

SPI	Serial Peripheral Interface
TCARI	Transformed Chlorophyll Absorption in Reflectance Index
UART	Universal Asynchronous Receiver-Transmitter

1 Introduction/Problem Definition

The client of this project was Don Roe, who is the CEO of the AgInno Institute. He previously worked at the Procter & Gamble Co. for 35 years and has more than 320 U.S. Patents. He and his wife, Lana, have been involved with around 50 mission trips in conjunction with the Church of the Nazarene. The El Puente station of the AgInno Institute is based in Guatemala, and focuses on Compassionate Ministries and Agricultural Research. The first deals with making "life improvements," like pouring concrete flooring to reduce disease, or getting better access to water. The Agricultural Research portion has to do with improving growing techniques of crops.

In conjunction with the Agricultural Research effort, Don Roe asked the team to "develop a low-cost, easy-to-use aerial field assessment system…to use to assess local farmers' fields before planting, during crop growth, and/or at harvest."[1] The team morphed this phrase into its own goal statement of "design and test a UAV used to determine the health of plants in Guatemala." This begs the question, what is a UAV?

UAV stands for unmanned aerial vehicle. This is essentially what it sounds like. It is a vehicle that flies in the air with nobody on board. Most people think of these as drones with four propellers. However, there are also fixed wing UAVs. These can be thought of as small airplanes. UAVs are remote-controlled and some have autopilot features.

Thus, the goal of this project was to create a UAV that can take pictures of a field of crops and determine the overall health of them to help farmers make better decisions while caring for them. This was not all the requirements needed, however. Through talking to Don Roe, it was determined that he needed first person view (FPV), manual and automatic control, communication, weather proofing, near infrared (NIR) and visual light (RGB) cameras, small form-factor, a user interface, and easy maintenance. On top of this, Mount Vernon Nazarene University's department of engineering and supporting faculty required the team to factor in the requirements and constraints in the table below.

Table 1.1. Requirements and Constraints

Creativity and Innovation	Form and Function
Marketability	Documentation
Presentations	Economic Concerns
Environmental Concerns	Social Concerns
Political Concerns	Ethical Concerns
Health and Safety	Manufacturability
Sustainability	Supply Chain Issues

Creativity and innovation refer to the ability to find solutions to complicated problems and foresee the possible consequences of one's work. Form and function deals with making sure the product design works as expected. For marketability, the project must be something that can be sold (theoretically). The documents associated with this project, like the instructions and final report, fall under the category of documentation. This project must also have multiple presentations throughout the duration. Economic concerns are dealing with money and buying components, as well as costs for other things like labor. Environmental concerns focus on making sure that the project does not harm the environment. In the category of social concerns, the team must make sure that people do not have a negative view of the product. Political concerns must take into account the laws of governing bodies and avoid choosing a political side. In ethical concerns, engineers must protect the safety and welfare of the public above all else. This also helps to explain the health and safety constraints. The product must be something that can be made repeatedly for it to cover the manufacturability category. The product must also have a reasonable lifespan for it to be sustainable. Finally, this project had to deal with supply chain issues where components were unavailable to be purchased.[2]

1.1 Team Structure

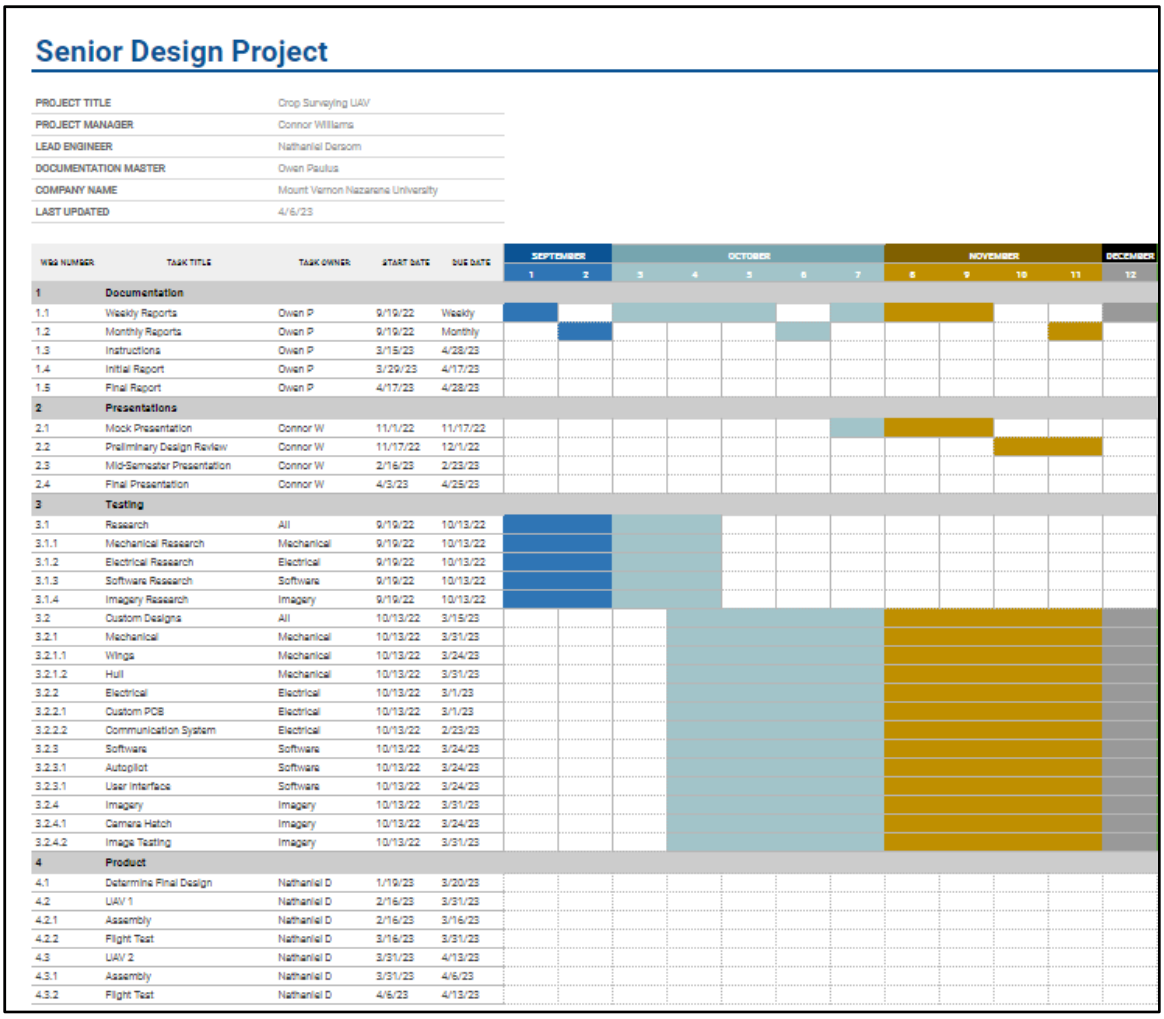
The team was given four roles to fill: project manager, lead engineer, documentation master, and product engineers. The project manager was in charge of the budget, meetings, scheduling, and presentations. The lead engineer was in charge of directing each team on what to research and do, as well as combining the efforts of each group. The documentation master managed the documents like the reports and instructions. Finally, the product engineers' jobs were to research and build the UAV. The first three roles would be filled by one person each, with the others being the rest of the team. It was decided that Connor Williams would be the project manager, Nathaniel Dersom would be the lead engineer, and Owen Paulus would be the documentation master. The product engineers included Morgan Maurer, Noah Stephens, Garett Fry, Isaac Harley, Caleb Syler, and Luke Shoen.

Aside from these positions, everyone was put into smaller groups in order to focus on, and become an expert in one area. These groups consisted of the Electrical, Imagery, Design and Manufacturing, and Software teams. Within the Electrical team was Owen, Isaac, and Caleb. This team's goals were split into three different areas: the flight controller, peripherals, and designing a custom PCB. The Imagery team included Noah and Luke. This team's goals involved taking pictures while in flight and then analyzing them to ascertain the health of the crops. The Design and Manufacturing team included Morgan and Garett. This team's goals were to create and build the body of the UAV. The Software team consisted of Connor and Nathaniel. This team worked as the glue for the other teams, as well as the developers of a large portion of the software embedded in the UAV. Each team goes more in depth into their work in Chapter 2.

1.2 Gantt Chart

Throughout the year, a Gantt chart was created and updated to guide the team. A Gantt chart is a schedule designed to be quickly and easily understood with a glance. The final Gantt chart can be seen below. Going from left to right, the chart is split into months and weeks. The labels on the left side are split into four deliverables: Documentation, Presentations, Testing and Product. These are the four major deliverables the team determined were needed by the end of the project. Each of these deliverables then go into

more detail with the Task Title. This gives an overview of what needs to be done. The Task Owner is the person most in charge of each task. The Start and End Dates describe the timeline of the task. Finally, Percent of Task Complete shows an estimate of how much has been accomplished for the task. This section was readjusted weekly.

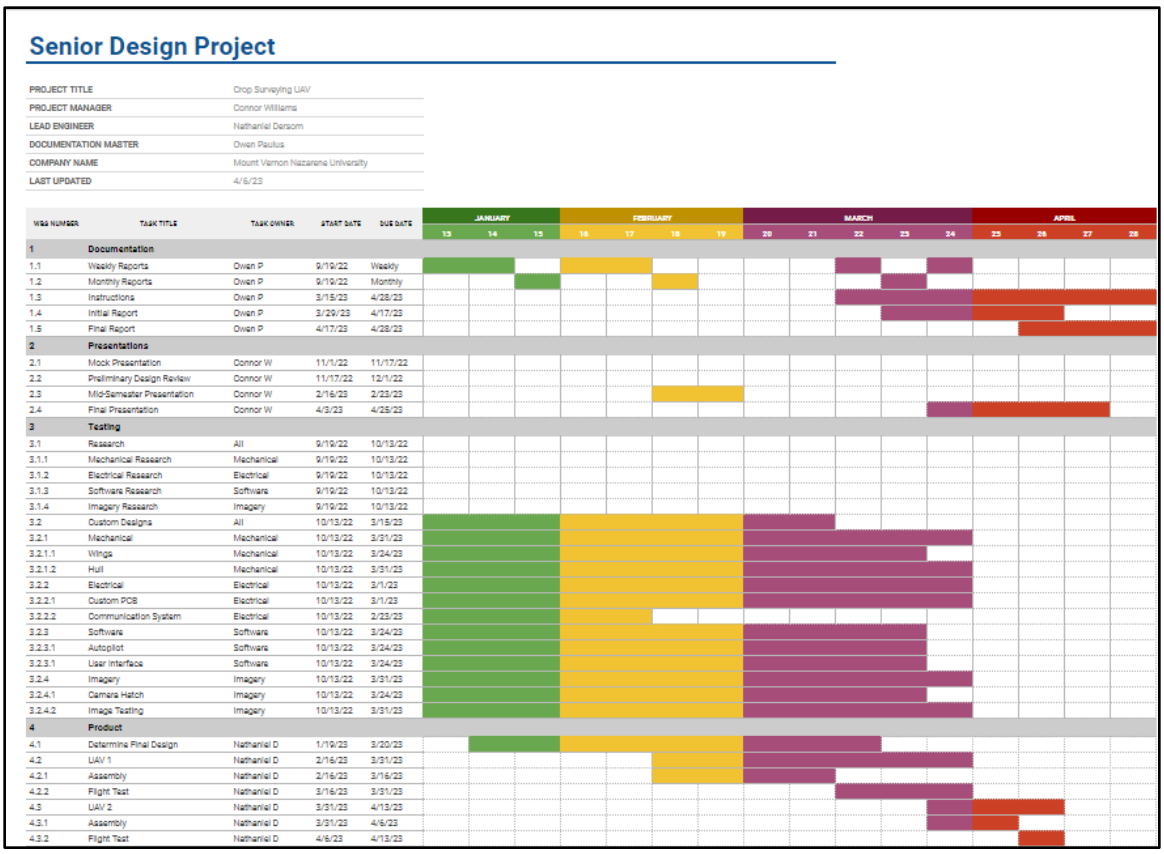


Senior Design Project

PROJECT TITLE	Crop Surveying UAV
PROJECT MANAGER	Connor Williams
LEAD ENGINEER	Nathaniel Dersom
DOCUMENTATION MASTER	Owen Paulus
COMPANY NAME	Mount Vernon Nazarene University
LAST UPDATED	4/6/23

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE
1	**Documentation**			
1.1	Weekly Reports	Owen P	9/19/22	Weekly
1.2	Monthly Reports	Owen P	9/19/22	Monthly
1.3	Instructions	Owen P	3/15/23	4/28/23
1.4	Initial Report	Owen P	3/29/23	4/17/23
1.5	Final Report	Owen P	4/17/23	4/28/23
2	**Presentations**			
2.1	Mock Presentation	Connor W	11/1/22	11/17/22
2.2	Preliminary Design Review	Connor W	11/17/22	12/1/22
2.3	Mid-Semester Presentation	Connor W	2/16/23	2/23/23
2.4	Final Presentation	Connor W	4/3/23	4/25/23
3	**Testing**			
3.1	Research	All	9/19/22	10/13/22
3.1.1	Mechanical Research	Mechanical	9/19/22	10/13/22
3.1.2	Electrical Research	Electrical	9/19/22	10/13/22
3.1.3	Software Research	Software	9/19/22	10/13/22
3.1.4	Imagery Research	Imagery	9/19/22	10/13/22
3.2	Custom Designs	All	10/13/22	3/15/23
3.2.1	Mechanical	Mechanical	10/13/22	3/31/23
3.2.1.1	Wings	Mechanical	10/13/22	3/24/23
3.2.1.2	Hull	Mechanical	10/13/22	3/31/23
3.2.2	Electrical	Electrical	10/13/22	3/1/23
3.2.2.1	Custom PCB	Electrical	10/13/22	3/1/23
3.2.2.2	Communication System	Electrical	10/13/22	2/23/23
3.2.3	Software	Software	10/13/22	3/24/23
3.2.3.1	Autopilot	Software	10/13/22	3/24/23
3.2.3.1	User Interface	Software	10/13/22	3/24/23
3.2.4	Imagery	Imagery	10/13/22	3/31/23
3.2.4.1	Camera Hatch	Imagery	10/13/22	3/24/23
3.2.4.2	Image Testing	Imagery	10/13/22	3/31/23
4	**Product**			
4.1	Determine Final Design	Nathaniel D	1/19/23	3/20/23
4.2	UAV 1	Nathaniel D	2/16/23	3/31/23
4.2.1	Assembly	Nathaniel D	2/16/23	3/16/23
4.2.2	Flight Test	Nathaniel D	3/16/23	3/31/23
4.3	UAV 2	Nathaniel D	3/31/23	4/13/23
4.3.1	Assembly	Nathaniel D	3/31/23	4/6/23
4.3.2	Flight Test	Nathaniel D	4/6/23	4/13/23

Fig. 1.1. Gantt chart fall semester



Senior Design Project

PROJECT TITLE	Crop Surveying UAV
PROJECT MANAGER	Connor Williams
LEAD ENGINEER	Nathaniel Dersom
DOCUMENTATION MASTER	Owen Paulus
COMPANY NAME	Mount Vernon Nazarene University
LAST UPDATED	4/6/23

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE
1	**Documentation**			
1.1	Weekly Reports	Owen P	9/19/22	Weekly
1.2	Monthly Reports	Owen P	9/19/22	Monthly
1.3	Instructions	Owen P	3/15/23	4/28/23
1.4	Initial Report	Owen P	3/29/23	4/17/23
1.5	Final Report	Owen P	4/17/23	4/28/23
2	**Presentations**			
2.1	Mock Presentation	Connor W	11/1/22	11/17/22
2.2	Preliminary Design Review	Connor W	11/17/22	12/1/22
2.3	Mid-Semester Presentation	Connor W	2/16/23	2/23/23
2.4	Final Presentation	Connor W	4/3/23	4/25/23
3	**Testing**			
3.1	Research	All	9/19/22	10/13/22
3.1.1	Mechanical Research	Mechanical	9/19/22	10/13/22
3.1.2	Electrical Research	Electrical	9/19/22	10/13/22
3.1.3	Software Research	Software	9/19/22	10/13/22
3.1.4	Imagery Research	Imagery	9/19/22	10/13/22
3.2	Custom Designs	All	10/13/22	3/15/23
3.2.1	Mechanical	Mechanical	10/13/22	3/31/23
3.2.1.1	Wings	Mechanical	10/13/22	3/24/23
3.2.1.2	Hull	Mechanical	10/13/22	3/31/23
3.2.2	Electrical	Electrical	10/13/22	3/1/23
3.2.2.1	Custom PCB	Electrical	10/13/22	3/1/23
3.2.2.2	Communication System	Electrical	10/13/22	2/23/23
3.2.3	Software	Software	10/13/22	3/24/23
3.2.3.1	Autopilot	Software	10/13/22	3/24/23
3.2.3.1	User Interface	Software	10/13/22	3/24/23
3.2.4	Imagery	Imagery	10/13/22	3/31/23
3.2.4.1	Camera Hatch	Imagery	10/13/22	3/24/23
3.2.4.2	Image Testing	Imagery	10/13/22	3/31/23
4	**Product**			
4.1	Determine Final Design	Nathaniel D	1/19/23	3/20/23
4.2	UAV 1	Nathaniel D	2/16/23	3/31/23
4.2.1	Assembly	Nathaniel D	2/16/23	3/16/23
4.2.2	Flight Test	Nathaniel D	3/16/23	3/31/23
4.3	UAV 2	Nathaniel D	3/31/23	4/13/23
4.3.1	Assembly	Nathaniel D	3/31/23	4/6/23
4.3.2	Flight Test	Nathaniel D	4/6/23	4/13/23

Fig. 1.2. Gantt chart spring semester

Going more in depth with the task overviews, the Documentation category is what the team leaves behind for others to learn from. This includes weekly and monthly reports to show the overall progress of the project, as well as instructions for operation, maintenance, and troubleshooting on the UAV. The Presentation category can be seen as a simplified version of the Documentation deliverable. There are two major presentations throughout the year, the Preliminary Design Review at the end of the first semester and the Final Presentation at the end. These presentations describe what the team has done and how within about an hour to an hour and a half.

The Testing category is the biggest and most demanding. This includes the initial research that each team had to do in order to understand the project, as well as additional research and work on certain aspects of the project. The additional research components and implementation thereof is classified as Custom Designs because it is what the tem did to make the UAV unique.The Design and Manufacturing team made customized wings and a hull. The Electrical team created a custom printed circuit board (PCB) and

connected the communications systems to each other. The Software team wrote code for the single board computer (SBC). Lastly, the Imagery team designed a protective case and changed filters for the cameras.

The last category is the Product category. This basically includes the final UAVs that will be given to Don Roe and MVNU. Both the assembly and flight test of each UAV was added, as both of these needed to be done for the product to be finished.

2 Design

2.1 Electrical Team

The electrical team focused their work in four main areas throughout this project. The areas include the flight controller, the communication systems, the power system, and additional custom designs. Each area will be expounded upon throughout the remainder of this section.

Flight Controller

The brain of the UAV is the flight controller. This module controls all of the flight aspects of the aircraft and enables control and autopilot as well as flight planning. This is accomplished through multiple sensors and inputs such as, the GPS module, the airspeed sensor, the telemetry system, and the RC system. Combining this with the internal gyroscope and compass, as well as control over the aircraft's control surfaces and propeller allows the Flight Controller to conduct and control the flight of the aircraft autonomously in tandem with a ground station. The flight controller allows the Crop Surveying UAV to behave autonomously allowing for ease of data collection after a flight has been planned and uploaded.

The flight controller chosen for the UAV was the Matek 743-Wing V3. This particular flight controller makes use of an STM32 microprocessor and has inputs for all the necessary systems listed above. This flight controller was chosen because of the familiarity of the STM32 microcontroller architecture to the electrical team, as well as its similarity to the flight controllers demonstrated by Aaron Aude.

Fig 2.1. The Flight Controller

Preparing the flight controller for flight was a lengthy task of making numerous changes to the physical structure of the device, as well as flashing the flight controller and configuring numerous parameters for each attached system. Flashing is the action of replacing the data on a microcontroller with something new, in this case the ArduPilot firmware was flashed. The numerous physical changes made include the addition of soldering pin headers for:

- Header Group 1: Cts7, Rts7, Tx7, Rx7, 5V, G. These pin headers are for connecting the telemetry radio.
- Header Group 2: DA1, CL1, Tx2, Rx2, 4V5, G. These pin headers are for connecting the GPS module.
- Header Group 3: Rx6, 4V5, G. These pin headers are for connecting the radio receiver.
- Header Group 4: C2, Vsw, G. These pin headers are for connecting the FPV camera (relay must be switched to activate camera 2).
- Header Group 5: G, 5V, LED, VTx. These pin headers are for connecting the FPV transmitter.

- Header Group 6: S1, and S3-S7. S1 and S2 are reserved for ESC outputs, S3-S10 are customizable. S3-S6 are being used for elevon and aileron outputs and S7 is used to trigger photo capture mode.
- All of the added female pin headers can be found in Fig. 2.2.

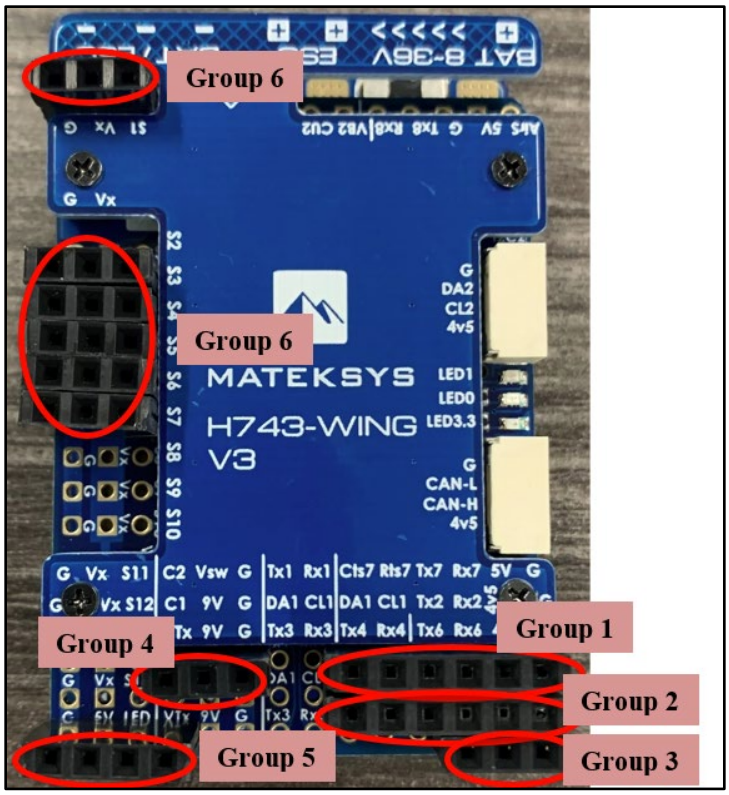


Fig. 2.2. The Addition of Pin Headers

Additional changes include connecting the top board to the bottom electrically through the ports shown in Fig. 2.4, as well as setting the voltage of the VSW to five volts by bridging the pad circled and zoomed below in Fig. 2.3. This voltage is necessary for the use of the FPV system.

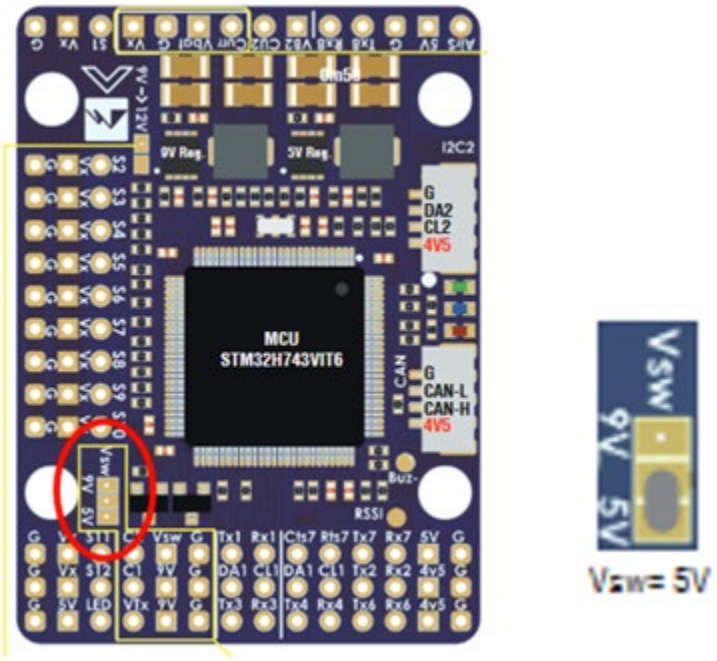


Fig 2.3. Setting the VSW[3]

The electrical connection of the bottom and top boards was completed through the soldering of silicon wire to the ports circled in Fig 2.4 and Fig 2.5.

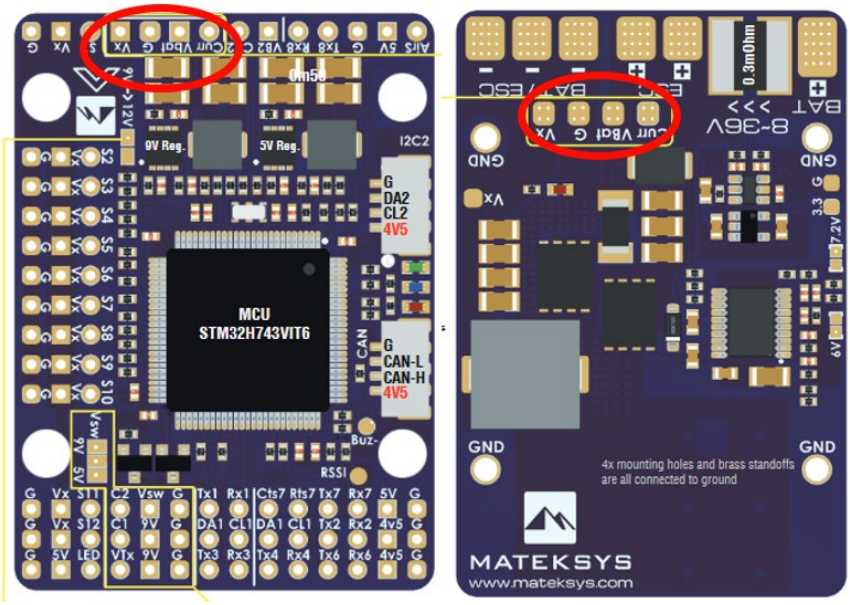


Fig 2.4. Electrically Connecting the Boards[3]

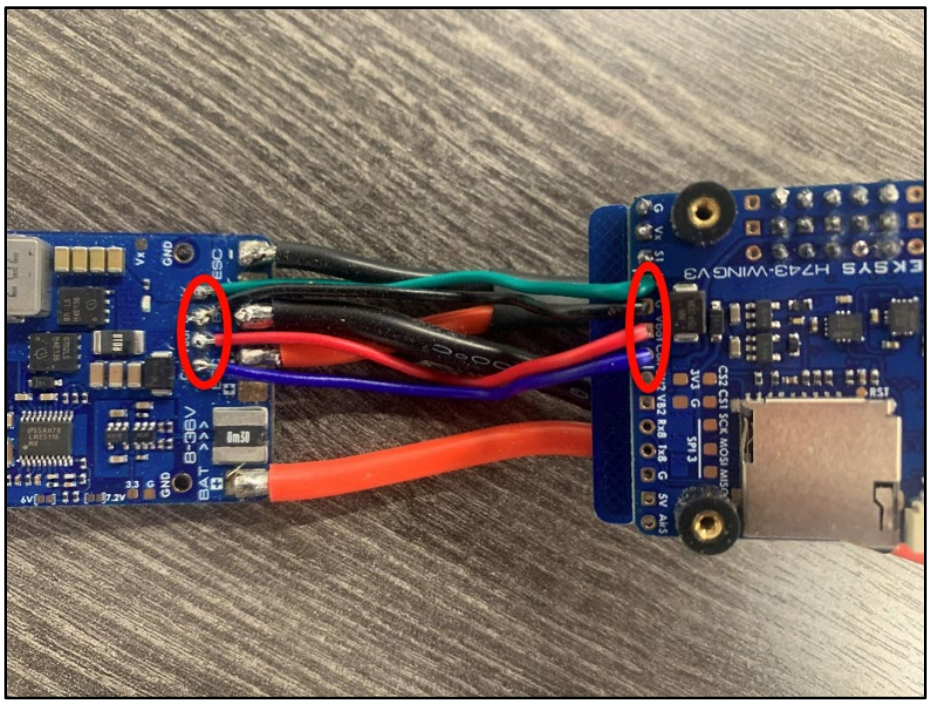


Fig 2.5. Method of Electrical Inter-board Connection

A final addition is the connectors for powering the flight controller, and powering the ESC which controls and powers the motor. These connectors are soldered to the pads circled in Fig 2.6 and Fig 2.7.

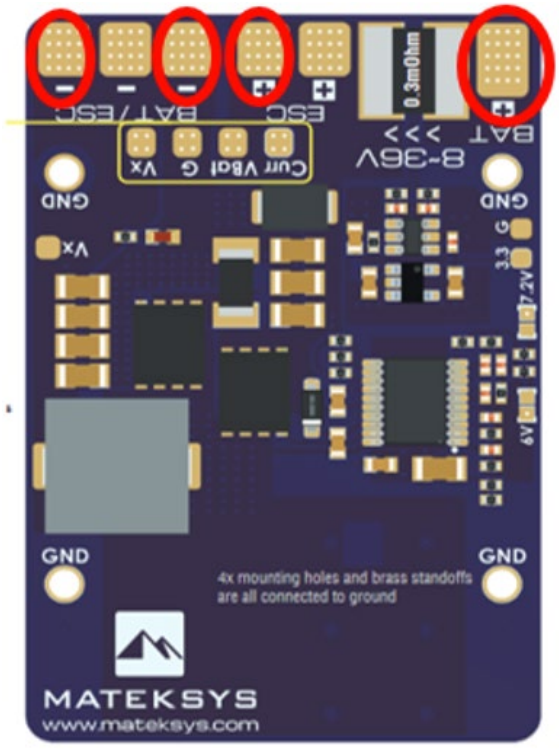


Fig. 2.6. Connection of the Power/ESC Connectors[3]

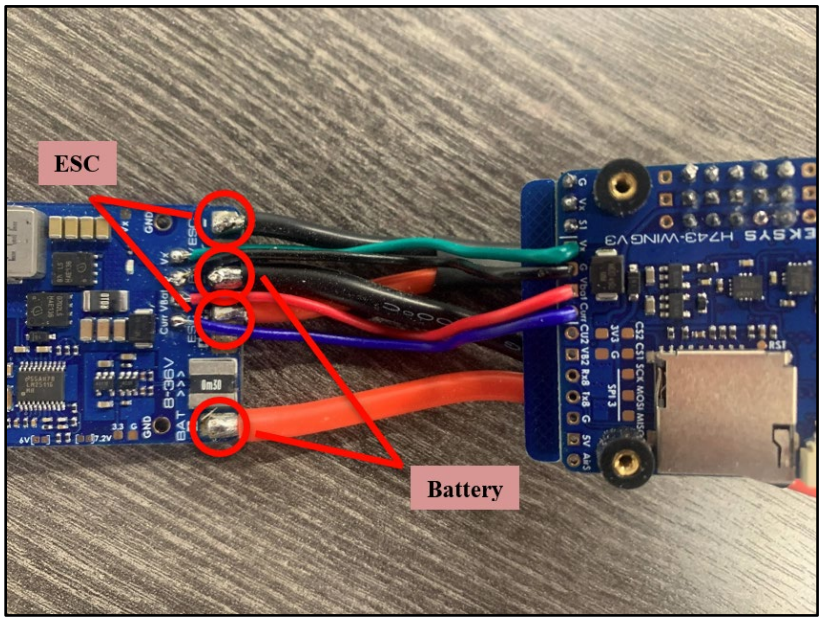


Fig. 2.7. Method of Power/ESC Connection

After this, the flight controller was electrically ready for all the necessary inputs and outputs that would be used by the UAV.

The flight controller must be flashed with firmware that allows it to interface with a ground station in order to program and control the flight controller. The ArduPilot firmware was used for the control of the UAV and was flashed to the flight controller. In order to do this, the STM32Cube Programmer software was used to erase the microcontroller data and flash the selected version of ArduPilot Plane.

Fig. 2.8. The Software Used to flash the Flight Controller[4]

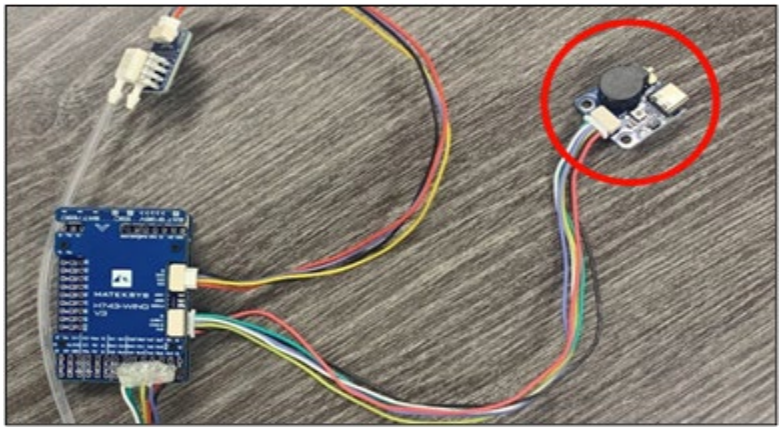


Fig. 2.9. The USB connection to the Flight Controller

The Cube Programmer connects to the flight controller through the USB connection provided by the USB/Buzzer Dongle included in the flight controller kit from Matek. This USB connection is the main way the flight controller (FC) is programmed when grounded.

The FC, having been prepared for connection through the physical changes and firmware flashing, was then connected to the ground station software. The ground station software used was called Mission Planner. This software is closely connected to the ArduPilot website and as such has considerable documentation. This software was used to configure the FC and calibrate all inputs and outputs in order to allow the FC to correctly control the UAV autonomously, as well as manually with the use of the RC system. After the connection of the FC to the Mission Planner Software, the FC was configured.

There were many parameters that were set in order to enable the interconnection of all the peripherals as well as the control surfaces. From Mission Planner, the RC system was connected and then calibrated. The servos that control the flight surfaces were then set to their corresponding preset positions. For example, servo three was set to the Right V-tail preset. After an extensive setting of parameters and calibrations through the use of Mission Planner, as well as setting the min/max of the servos, the flight controller could be controlled through the use of RC. It could also be controlled by Mission Planner though the telemetry system. The FPV system was also configured through Mission Planner as it controls the channel that the FPV will be transmitted through to be set, as well as setting an On-Screen Display (OSD) internal to the FC.

Parameters Set:

- SERIAL1_BAUD =57 (default), controls the baud rate of the telemetry system.
- SERIAL1_OPTIONS = 0 (default)
- SERIAL1_PROTOCOL = 2 (default), option two is for MAVlink 2.
- ARSPD_USE = 1
- Servo 1 set to Throttle
- Servo 2 set to Disabled
- Servo 3 set to Vtail Right
- Servo 4 set to Vtail Left
- Servo 5 set to right Aileron
- Servo 6 set to left Aileron
- Servo 7, set to RC5 for photo capture activation on Raspberry pi

- MIXING_GAIN = 1.2
- BRD_ALT_CONFIG = 1, this makes RX6 become the SERIAL7 port's RX input pin.
- SERIAL7_PROTOCOL = 23
- SERIAL7_OPTIONS = 12
- RSSI_TYPE = 3
- SERIAL7_BAUD = 57 (default)
- FLTMODE_CH = 7, controls what RC channel swaps flight modes
- Flight modes set to: Manual, Autotune, FBWB
- VTX_POWER (Left at default)
- RELAY_PIN2 = 82 for camera 2 on/off.
- VTX_ENABLE = 1
- VTX_CHANNEL = 0 (default is 0)
- VTX_BAND = 0 (default is 0 for band A)
- INITIAL_MODE=0 (manual flight)
- COMPASS_AUTODEC = 1
- COMPASS_AUTO_ROT = 2

Flight modes were also configured through Mission Planner allowing the FC to be controlled manually, or to fly autonomously, as well as to enter a mode that calibrates the autopilot over time according to each unique aircraft.

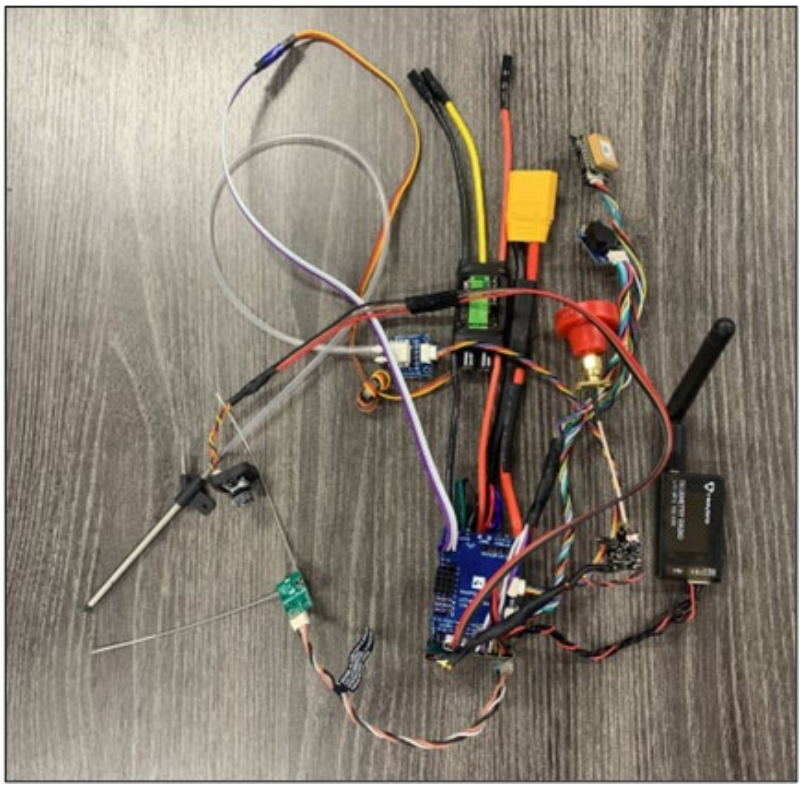


Fig. 2.10. The First Flight Controller

Once Mission Planner was utilized to configure the FC, the FC was able to control the UAV. Mission Planner could then be used to monitor the flight through the use of the telemetry system. This combined with the manual input from the RC radio allows the FC to conduct flights. This is how the first test flight of the UAV was completed. Once a first FC was completed, a second FC was set up in much the same way with the benefit of lessons learned. This will allow Don and his team to complete flights in Guatemala while working together with the imaging system to collect the required data.

Communication Systems

The transfer of signals is one of the key aspects of the functionality of every UAV. For the UAV that the team developed, there are a total of four communication systems to achieve this in four different ways: telemetry, radio control (RC), first person view (FPV), and the global positioning system (GPS). With three different signals within the same medium, there were some factors to be considered to mitigate any potential

distortion, namely directionality, polarization, and frequency. Another factor considered was the legal standard of communication bands within Ohio, the United States, and Guatemala. Finally, the constraint of optimizing both at the least cost and the best functionality was considered in the selection of each component.

Electromagnetic fields and waves and the theory behind them is important to consider when considering radio communication, as well as general communication theory. In fact, interference is important to consider when estimating the range of the entire communication system. Constructive interference is how to direct a signal to prolong the distance that it can propagate before attenuation while sacrificing how wide the angle of the beam would be: 360° for one is considered as "omnidirectional". The directivity of an antenna is given by the ratio of the maximum power of a signal transmitted relative to the total power of a signal transmitted, as defined in IEEE standards.[5] Eq. 2.1. shows the multivariable function using spherical angles representing a signal's directivity for a certain set of spherical angles:

$$D(\theta,\phi) = \frac{U(\theta,\phi)}{\frac{1}{4\pi}\int_{\phi=0}^{\phi=\pi}\int_{\theta=0}^{\theta=2\pi}U(\theta,\phi)sin\theta d\theta d\phi},$$

(Eq. 2.1)

where θ is the zenith or elevation angle, ϕ is the azimuth or horizon angle, $D(\theta,\phi)$ is directivity, and $U(\theta,\phi)$ is the radiation intensity. In order to find the maximum value for $D(\theta,\phi)$, called directive gain, simply replace $U(\theta,\phi)$ with the maximum of $U(\theta,\phi)$ in Eq. 2.1, because of the unchanging nature of the integral in the denominator. This is the case where the direction is not specified.[5] In order to quantify this with a type of unit used extensively in antennas, Eq. 2.2 shows how to relate the directivity of one antenna with that of another:

$$D_{dB} = 10log_{10}[\frac{D}{D_{ref}}], \quad \text{(Eq. 2.2)}$$

where D is the directivity in question, D_{ref} is the directivity of the reference antenna, and D_{dB} is the resulting compared directivity in decibels [dB]. By using an ideal isotropic antenna for D_{ref} in Eq. 2.3, the equation then becomes:

$$D_{dBi} = 10log_{10}[D], \quad \text{(Eq. 2.3)}$$

where D_{dBi} is the resulting directivity in decibels relative to isotrope [dBi]. This directive gain is how the different antennas are compared in functionality. Fig. 2.11 compares the

directive gain of an isotropic antenna, an omnidirectional antenna, and a directional antenna as well as visually shows the generalized area of coverage that a signal using these antennas would be subject to.

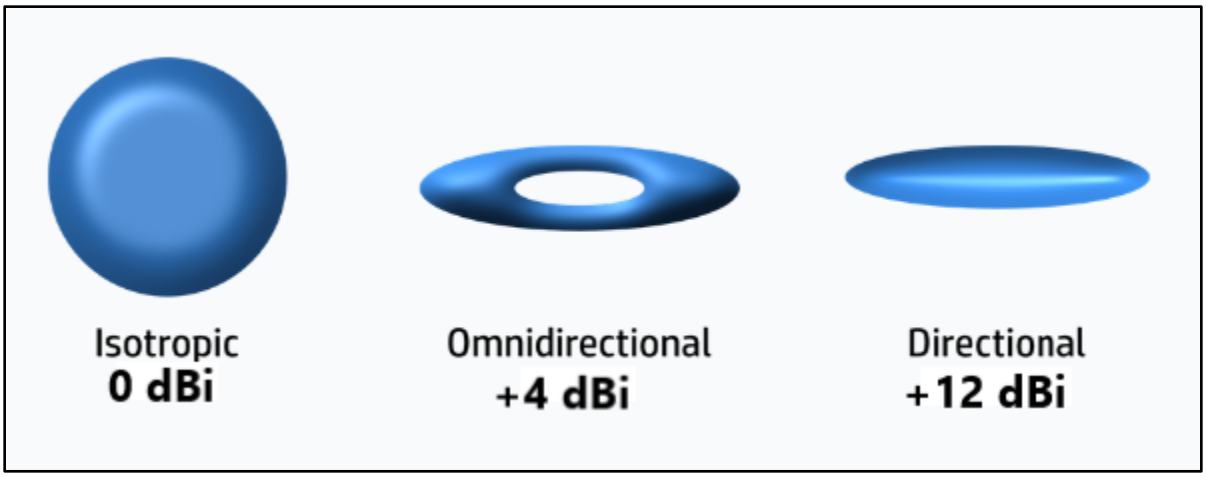


Fig. 2.11. A Visual Example of the Difference of Gains Using Different Directivity[6]

Another factor considered in the selection of antennas was the polarization of the signal. Polarization is the changing of shape of an electromagnetic signal in order to combat interference by objects in the way of the signal. A reason to do this, especially in the case of UAVs, is to assure that the highest percentage of a signal is received, regardless of the location and potential of physical obstacles to obstruct the path of the signal and attenuate its strength. This is especially important to signals that require higher frequencies, because there would be more of a chance that a large amount of data would be attenuated because of an obstacle. This is where circular polarization comes in, shown in Fig. 2.12.

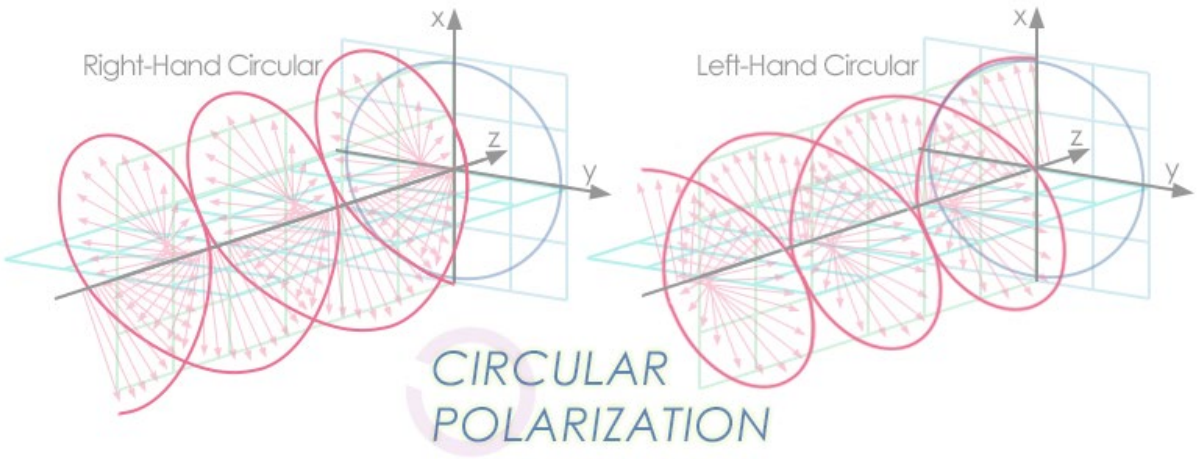


Fig. 2.12. A Graphical Example of Circular Polarization[7]

As shown in Fig. 2.12, circular polarization twists the signal about the axis of propagation, sometimes referred to as the Poynting vector in electromagnetic theory, to mitigate the disruption of signals of the same frequency or obstacles that would absorb the signal should it be linear instead. Also seen in Fig. 2.12 is the fact that there are two different types of circular polarization given the two different ways a signal can twist geometrically.

Lastly, the frequency of each system was carefully selected to be both legal on an international, federal, and state scale and optimized for the best functionality to maximize distance and minimize power draw. Within the realm of legality, upon initial research by using the standards by the Federal Communications Commission (FCC), the bands allocated for telemetry are within any of the bands that are primarily used for radiolocation.[8] But out of safety, and in considering the international standards as well, the lowest frequency that could be selected within constraint is 433 MHz because that band has a secondary purpose labeled “amateur”.[8] Similarly, for the RC system, 2.4 GHz was selected because it falls within the primary category of “amateur”[8] as well as it being a very common frequency used for remote controls to ease the selection of components. For the FPV system, a higher frequency is desirable because the higher the frequency, the faster higher resolutions of data can be transferred. This meant that, for full color live FPV, 5.8 GHz was selected for being within the same primary and secondary categories as 433 MHz, being radiolocation and “amateur” respectively.[8]

With regards to the GPS, the band cannot be outside of either the range from 1164 to 1240 MHz or the range of 1559 to 1610 MHz by FCC standard[8], so the module was selected based on the best compatibility for the flight controller, the Mateksys module M8Q-5883 for compass and GPS was selected as shown in Fig. 2.13.

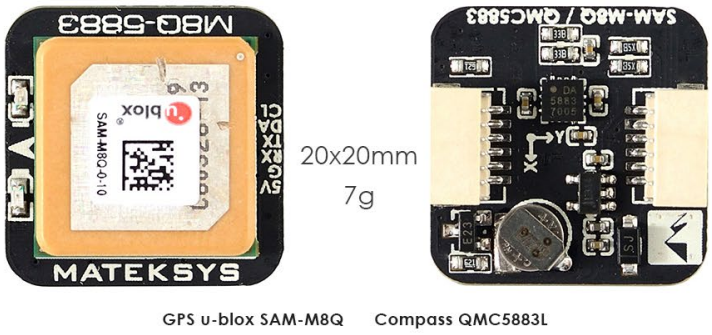


Fig. 2.13. Mateksys M8Q-5883 Compass and GPS Module[9]

As seen in Fig. 2.13, the GPS module uses a patch antenna. There was no need to alter this module's setup.

An intermediate step was accidentally overlooked when selecting the communication systems, specifically for the RC system. As for the GPS, all wireless communication systems have to have matching *wired* communication systems with the flight controller. As seen in the previous subsection, the FPV system, telemetry system, GPS, and R/C system all must have some form of serial communication known as universal asynchronous receiver-transmitter (UART). Unfortunately, in selecting the remote controller—a similar controller as used by Aaron Aude in his demonstration during the research phase—for the UAV, shown in Fig. 2.14, this was overlooked for the receiver of the RC system, where it outputted multiple pulse-width modulated (PWM) signals for a servo or the electronic speed controller (ESC) of a motor.

Fig. 2.14. Spektrum SPMR1010 R/C Controller[10]

A solution was investigated to combine these signals into a singular pulse-position modulated (PPM) signal or a serial value by an additional external module. A PPM signal is a chain of PWM signals separated by different time intervals in a specified order, whereas the serial value is a more compact and faster way of representing the value of a PWM signal with a hexadecimal value. In fact, it was better to replace the entire receiver with another one that does this on the same board to reduce any potential latency in computing, which provides potentially a faster response time than of an external compute module. So, in order to fix the issue, the selected receiver used in the UAV is the SPM4650 by Spektrum, shown in Fig. 2.15.

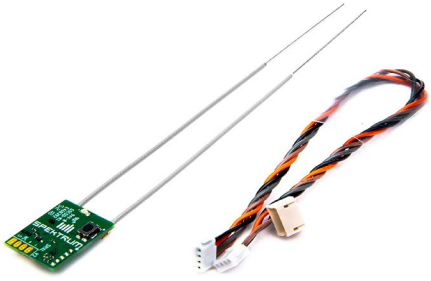


Fig. 2.15. Spektrum SPM4650 RC Receiver[11]

Since this receiver has automatic antenna switching, it theoretically simulates the functionality of a dipole antenna. As for the remaining systems, there was no compatibility issue between the devices and the flight controller.

The FPV system uses a camera that can follow either National Television Standard Committee (NTSC) or Phase Alternate Line (PAL) standard via switching, shown in Fig. 2.16.

Fig. 2.16. Caddx Ant Wide Dynamic Range Camera for FPV capture[12]

This camera then sends its image data directly to the flight controller into its onboard on-screen display (OSD) integrated circuit (IC) for an additional overlay of data for the data from the other peripherals of the UAV (eg. air-speed, direction, roll and pitch, etc.), and then is sent to the selected FPV transmitter, shown in Fig. 2.17.

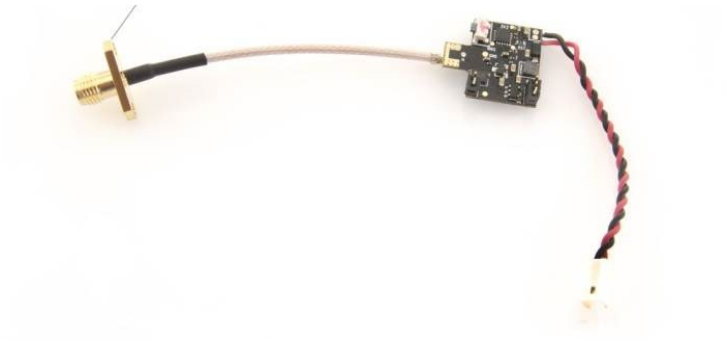


Fig. 2.17. Lumenier SM-25 Video Transmitter (VTX)[13]

Since the video transmitter is at such a high frequency, in order to secure a clean connection with minimal interference, the signal was left-hand circularly polarized by using a pagoda antenna, shown in Fig. 2.18.

Fig. 2.18. Emax Pagoda Antenna[14]

There was another type of shorter pagoda antenna from the same manufacturer used in the prototyping process, but unfortunately that antenna has been discontinued. Luckily the gain across both of these antennas is approximately the same, so the functionality should only be affected marginally. The FPV signal then is transmitted to either another one of the antennas shown in Fig. 2.18 or in a patch-pagoda antenna as shown in Fig. 2.19.

Fig. 2.19. Realacc Direction Circularly Polarized Pagoda Patch Antenna[15]

The signal from either of these antennas are finally shown in the video receiver, shown in Fig. 2.20. This is achieved by either automatic or manual switching, depending on the setting of the receiver.

Fig. 2.20. Zerone LCD Screen Monitor for FPV Signal Receiving[16]

Finally, the telemetry system was a pre-packaged system that can use either a module directly with USB or hard-wired into the flight controller, shown in Fig. 2.21. It

uses the selected 433 MHz and connects with the flight controller using serial wire connection.

Fig. 2.21. HolyBro SiK Telemetry Radio v3 for the Telemetry System[17]

This system uses the stock monopole antennas included with the system. However, if there were more time in this project, the antennas for this system could also be customized to optimize the range even further.

Power System

Considering all of the components utilized within the UAV, the team performed a power consumption study prior to selecting a battery to power the system. Before performing the study, the team had decided to use a 4-cell lithium polymer (LiPo) battery to power the UAV. LiPo batteries are commonly used in both fixed wing and multirotor UAVs and were highly recommended by the UAV domain expert, Aaron Aude. LiPo batteries are compact and offer a high energy density compared to other batteries with similar energy specifications. Once the team decided to utilize a LiPo battery for the main power source on the UAV, calculations could be made to predict battery performance under specific loads. The power consumption study gave the team insight into what specifications on a LiPo battery were necessary for safely powering the aircraft as well as maximum flight times under a specific load. Fig. 2.22 displays a breakdown of the study on an Excel spreadsheet.

Component	Input Voltage (V)	Max Continuous Current Draw per Individual Voltage (A)	Power Consumption per Individual Voltage (W)	Max Current Draw from Battery (14.8V) (A)	
GPS Module	5	0.04	0.2	0.013513514	
Airspeed	5	0.005	0.025	0.001689189	
Raspberry Pi	5	1	5	0.337837838	
Flight Controller	14.8	2	29.6	2	
Telemetry Transmitter	5	0.1	0.5	0.033783784	
FPV Camera	5	1	5	0.337837838	
Video Transmitter	9	0.35	3.15	0.212837838	
Custom PCB	5	1	5	0.337837838	
RC Receiver	5	0.1	0.5	0.033783784	
Servos	5	1.08	5.4	0.364864865	
Motor	14.8	30	444	30	
	Total	36.675	498.375	**33.67398649**	
				Expected Motor Draw ~5 amps (17%)	
				Expected Average Amp Draw for Given Flight	**7.674-8.674 amps**
Option 1: 6600mah, 12C, 4S	$51.32	Weight: 598g			
Flight Time Range	45.7-51.6 mins	Size: 144x42x51mm			
Max Current Discharge	79.2 A for 5 mins	Connection Type: JST-XH and XT-60	https://hobbyking.com/en_us/turnigy-high-capacity-battery-6600mah-64s-12c-drone-lipo-pack-xt60.html		
Option 2: 5000mah, 25C, 4S	$41.97	Weight: 552g			
Flight Time Range	**34.6-39.1 mins**	Size: 147x49x33mm			
Max Current Discharge	125 A for 2.4 mins	Connection Type: JST-XH and XT-90	https://hobbyking.com/en_us/turnigy-battery-5000mah-4s-25c-lipo-pack-xt-90.html		

Fig. 2.22. Power Consumption Study

All of the components that consume electrical power within the UAV were included in this study. The team studied each component's datasheet in order to find their maximum current or power draw values and their operating voltages. Once these values were located, they were compiled into the spreadsheet and the maximum power draw for each component was calculated using Ohm's Law found as Eq. 2.4. Using Ohm's Law once more, the maximum power draw for each component was divided by 14.8 volts to obtain the maximum current draw from each component directly off the LiPo battery's four cells (3.7 V per cell). The maximum current draw values could then be summed together to realize the whole system's maximum current draw when each component is running at their full capacity. The team calculated that the system's maximum current draw was roughly 34 A. However, it is not expected to ever reach 34 A because the motor has been predicted to only draw around 5A at the high-end during a given flight. This prediction is based on both the thrust test explained in Section 2.3 and Aaron Aude's experience in monitoring motor performance on his personal fixed wing UAVs. Although the system will be much more efficient than calculated in this study, the team wanted to

gather data considering the worst-case scenario. This data gave us insight into how long the UAV could fly during a worst-case scenario using different LiPo batteries.

$$P = V \times I\ [Watts] \quad \text{(Eq. 2.4)}$$

In this study, the specifications on two different LiPo batteries were compared. Using their specifications and the maximum current draw value calculated for the UAV system, it was determined how long the batteries would last during a given flight. Eq. 2.5 was used to determine the maximum flight times. After comparing the specifications, the team decided on a 5000 mAh, 25C, 4-cell LiPo battery made by Turnigy as seen in Fig. 2.23. This battery offers the UAV around 35 minutes of flight time when all components are running at their maximum capacity and the motor is drawing around 5 A as predicted. However, the team believes that the UAV can last well over an hour before the battery is depleted because of the electronic speed controller (ESC) that connects the motor to the flight controller. The ESC varies the voltage supplied to the motor to increase its efficiency mid-flight. This feature reduces the power supplied to the motor which in turn allows for longer flights. Another factor that will extend the flight time is the amount of gliding that will be performed during a flight. When the UAV is making passes over a field, there will be times when the motor is driven at very low speeds which will reduce the power drawn from the battery. Although the battery's life for an average flight is not known, the pilot will have visible access to the battery's charge percentage on Mission Planner in real-time during a flight. The Turnigy 5000 mAh LiPo battery will be a reliable battery with little to no performance loss over time as long as the battery is properly used and stored.

$$Battery\ Life = \frac{Battery\ Capacity\ (Ah)}{Load\ Current\ (A)}\ [Hours] \quad \text{(Eq. 2.5)}^{[18]}$$

Fig. 2.23. Turnigy 5000 mAh LiPo Battery[19]

Additional Custom Designs

In addition to the UAV, the team decided to design a custom circuit board and a handheld weather station to ensure safe use of the product. Early in the fall semester the team brainstormed on how to implement a custom printed circuit board (PCB) into the UAV's design. After considering the tropical climate of Guatemala and the nature of LiPo batteries, the team decided to design and implement a circuit board that would monitor the battery's external surface temperature and compare it to the relative humidity and ambient temperature of the surrounding environment. LiPo batteries are temperature sensitive power sources that have an operating temperature range of -4° to 140° F.[20] The temperature of the battery naturally increases when connected to a load. Considering the nature of this type of battery and the warm temperatures that will be present, the team realized the potential risk of overheating. If a LiPo battery exceeds a temperature of 140° F, it could potentially explode and cause damage to the surrounding components or the user. Though the LiPo battery is not expected to reach its maximum temperature during a given flight in Guatemala, the custom PCB would allow the user to monitor the temperature of the battery and its performance in different weather conditions. The custom PCB would also have the ability to send a signal to the flight controller if the battery reached a certain temperature as result of an electrical short onboard the UAV.

Once the team decided to pursue the design of a battery monitoring PCB, research was performed on the components necessary for the design requirements. Firstly, the team decided to use the Si-7021 temperature and relative humidity sensor found on a

breakout board in Fig. 2.24. This sensor would be used for measuring the ambient temperature and relative humidity of the surrounding environment on board the UAV. In regards to measuring the temperature of the battery's surface, the team decided to use a thermocouple amplifier with a K-type thermocouple. The MCP-9600 I^2C thermocouple amplifier was chosen as seen in Fig. 2.25. to fill this role.

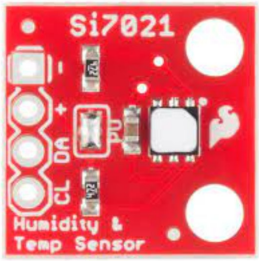


Fig. 2.24. Si7021 Breakout Board by SparkFun[21]

Fig. 2.25. MCP-9600 Thermocouple Amplifier Breakout Board by Adafruit[22]

Once the two sensors were chosen, the team had a decision to make regarding what kind of microcontroller to use to control the peripheral components on the board. A design based on the open-source Arduino microcontroller could have been easily done, decreasing complexity and compatibility issues. However, the electrical team decided to use an industrial microcontroller as the brain of the board to gain experience with these chips. This decision made the design process much more challenging due to the limited resources available compared to Arduino. However, the team believed that the experience would be worthwhile and very beneficial for any of the team member's future careers in the engineering field. With this in mind, the team decided to use a STMicroelectronics microcontroller. Two of the electrical team members had past experience using STMicroelectronics products which influenced the decision to use the brand. The team

also had direct access to STMicroelectronics prototyping boards in the STORM Lab which saved us from having to buy a prototyping board for the software design phase. Once the breakout boards for the necessary sensors were received in the mail, the electrical team began the software design phase by connecting the STMicroelectronics Nucleo-L476RG MCU board to the sensors using a small breadboard and multiple jumper wires as seen in Fig. 2.26. The code that was written to read from the two sensors was written in C language and I^2C communication protocol was used for data retrieval and transmission. The team had to learn how the I^2C driver worked on STMicroelectronics' Cube Integrated Development Environment (IDE) to be able to successfully read data from the sensors. The learning process was very beneficial, but took time in the initial portion of the design process. Once the code was successfully written, the data could be displayed on a serial monitor to confirm proper operation as seen in Fig. 2.27.

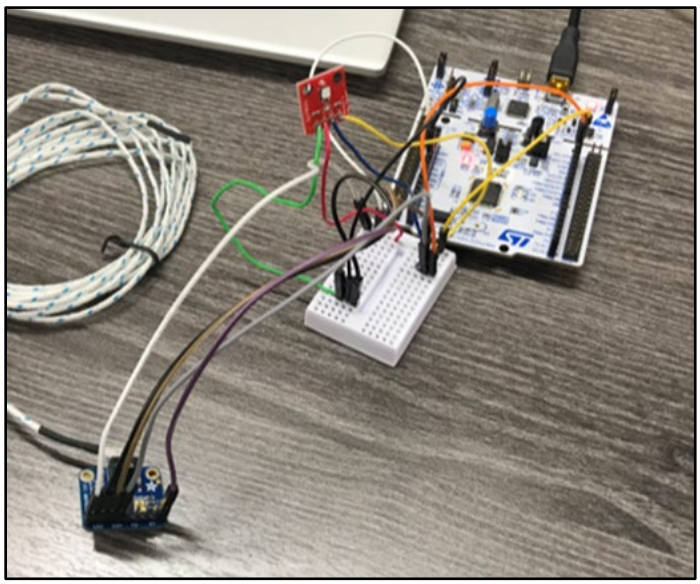


Fig. 2.26. Custom PCB Prototype

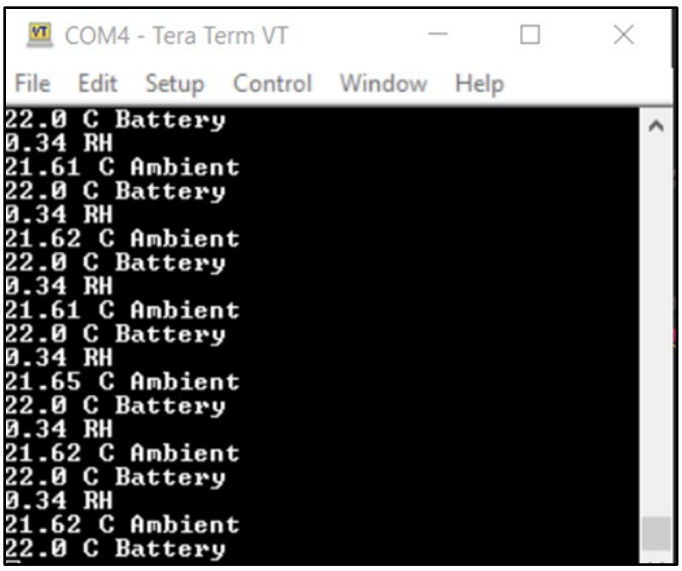


Fig. 2.27. Sensor Data Displayed on Serial Monitor

After the code was written for reading sensor data, a plan was created to transmit the data to be viewable by the user. The initial thought was to save all of the data onto an onboard SD card while also connecting the PCB to the flight controller so that signals could be sent to the pilot if the temperature of the battery exceeded the maximum threshold. Leaning toward this option, the team initiated the board design phase on EAGLE CAD. Throughout the board design phase, there were several obstacles that were faced. Firstly, the current electrical component shortage that has been present around the world for the past couple years severely impacted the progress within this phase. It was found that the MCP-9600 thermocouple amplifier chip was no longer available on any trusted supplier's website. The team had to return to the drawing board to find a component that would fulfill the battery temperature monitoring role. Eventually the MAX31855 SPI thermocouple amplifier was chosen as seen in Fig. 2.28. This thermocouple amplifier offered the same functionality as the MCP-9600, except for the use of a different communication protocol. The MAX31855 used SPI rather than I^2C. This difference forced the team to learn how to successfully use the SPI driver on the Cube IDE. Again, the learning and experience involved with this change was very beneficial, but it delayed the design process. Once the software was written for the MAX31855, the board design phase resumed. The final rendition of the software for the prototype can be found in the Appendices section of this report.

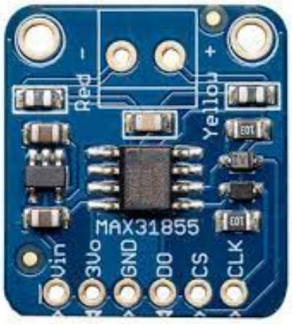


Fig. 2.28. MAX31855 Thermocouple Breakout Board by Adafruit[23]

Once the board was designed, the design underwent a series of revisions. The board had to be revised for multiple component substitutions due to JLCPCB's component stock being limited. JLCPCB was the circuit board fabrication company selected to fabricate the custom boards. After final revisions were made, the EAGLE design files were submitted into JLCPCB's website for fabrication. The following figures display the schematic, board layout, and JLCPCB rendering respectively.

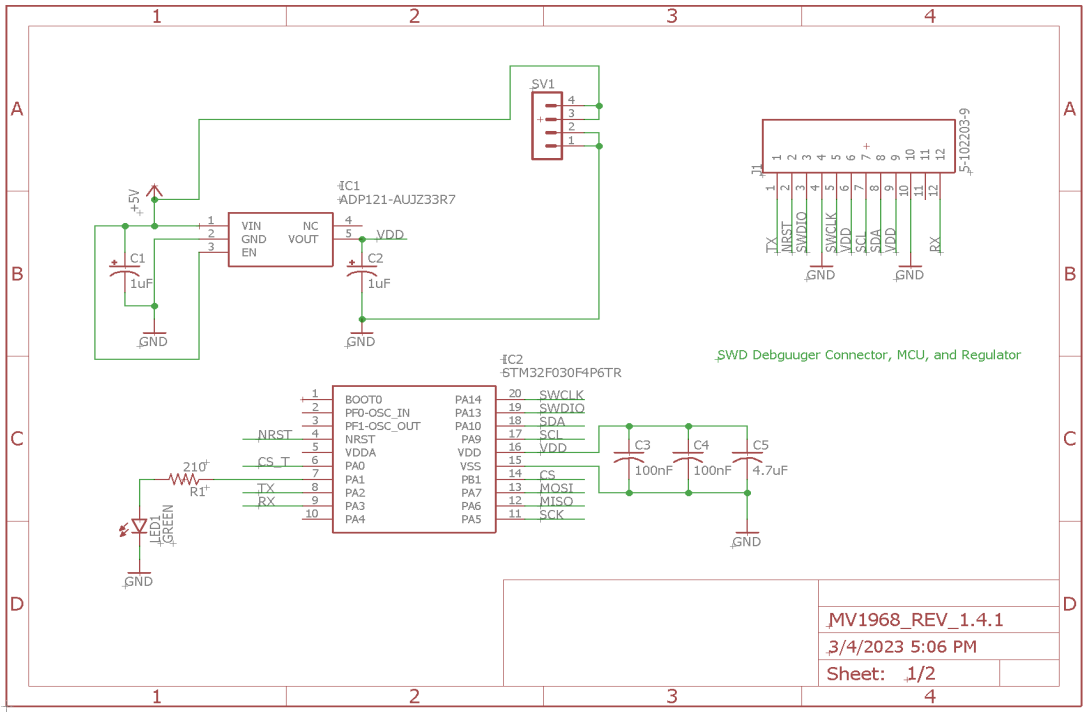


Fig. 2.29. Custom PCB Schematic Page 1

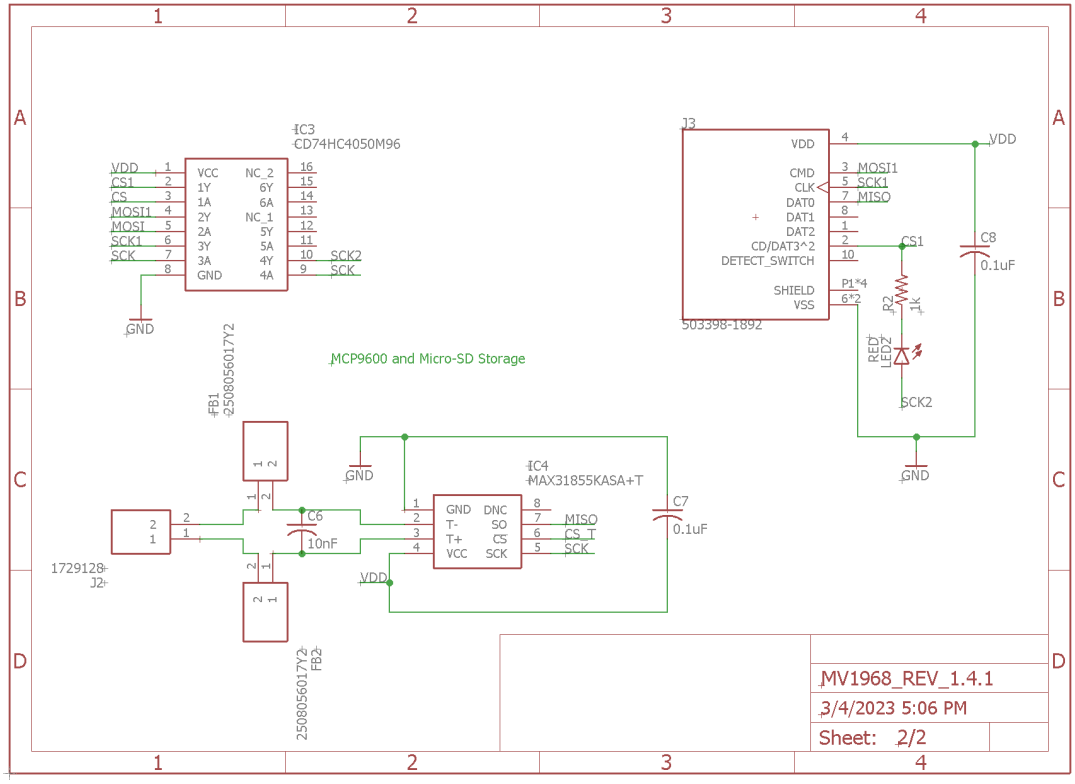


Fig. 2.30. Custom PCB Schematic Page 2

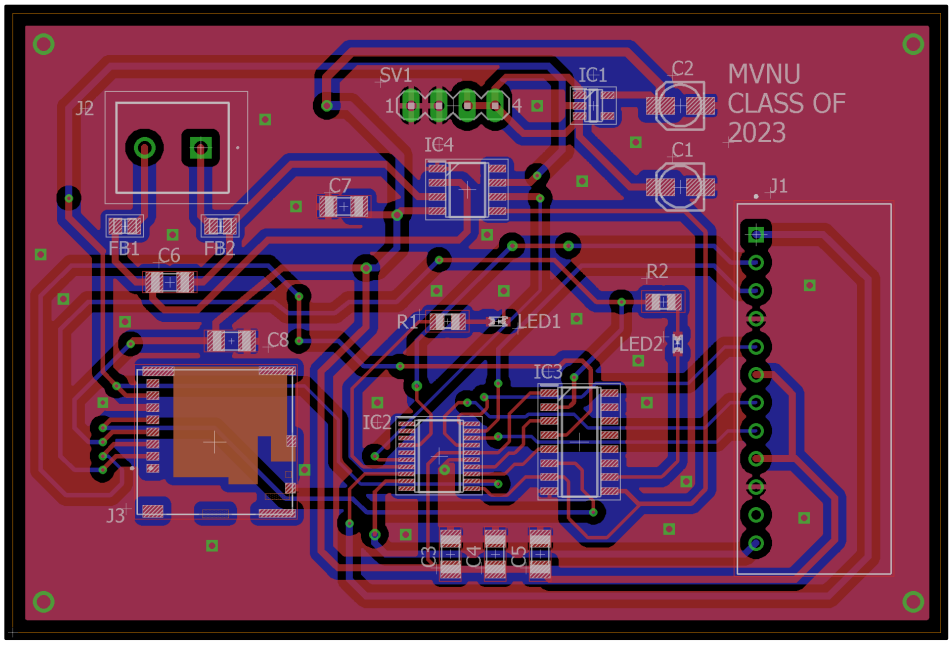


Fig. 2.31. Custom PCB Board Layout

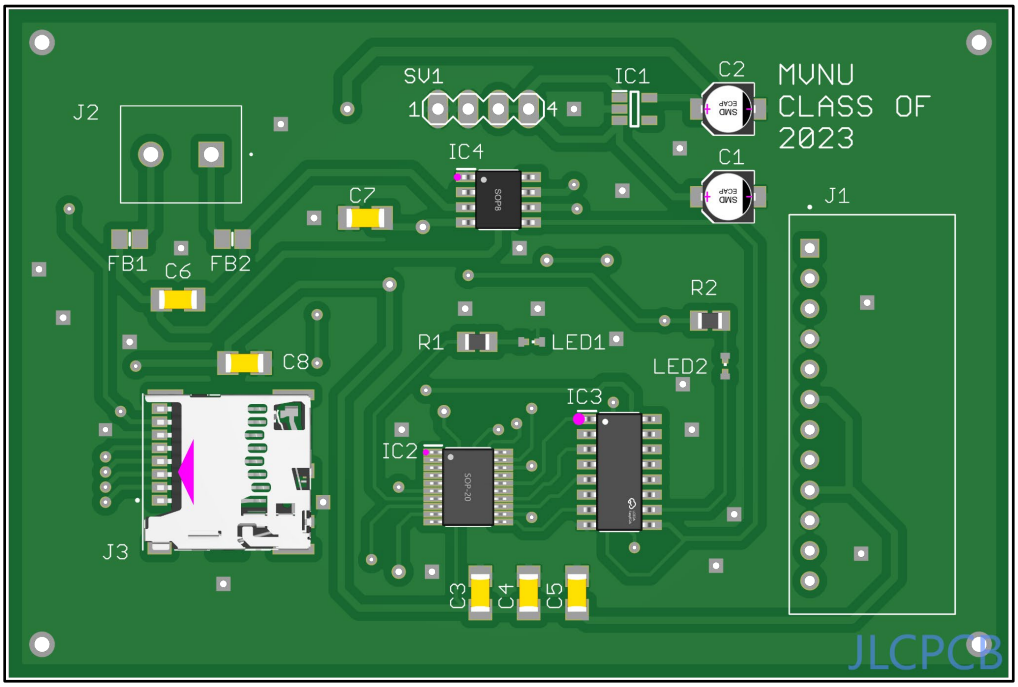


Fig. 2.32. Custom PCB JLC PCB Rendering (Top)

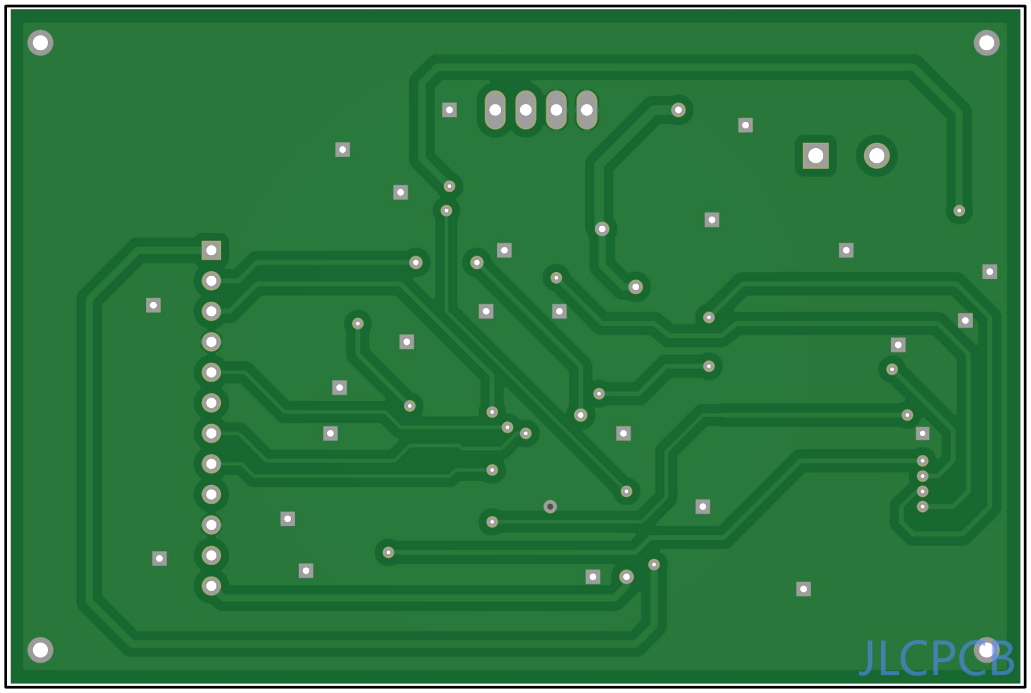


Fig. 2.33. Custom PCB JLC PCB Rendering (Bottom)

As one may observe, there is a slot for a microSD card to be inserted for data saving. However, there was not time to finish writing the software necessary to enable data to be saved on a microSD card. The code for enabling this function proved to be complex after several attempts at trying to save data, which led to the decision to use the TX line from the custom PCB to transmit data to the Raspberry Pi board for further processing. Writing software for enabling the microSD card was placed in the future-works category. All of the components viewed on the board have been placed in a bill of material which can be found in the table below.

Table 2.1. Custom PCB Bill of Materials

Comment	Designator	Footprint
1uF	C1,C2	CPOL-US153CLV-0405
100nF	C3,C4	C-USC1206
4.7uF	C5	C-USC1206
10nF	C6	C-USC1206
0.1uF	C7,C8	C-USC1206
15K	R5,R6	R0603
2508056017Y2	FB1,FB2	2508056017Y2
ADP121-AUJZ33R7	IC1	ADP121-AUJZ33R7
STM32F030F4P6TR	IC2	STM32F030F4P6TR
CD74HC4050M96	IC3	CD74HC4050M96
MAX31855KASA+T	IC4	MAX31855KASA+T
5-102203-9	J1	5-102203-9
1729128	J2	1729128
503398-1892	J3	503398-1892
GREEN	LED1	LEDCHIPLED-0603-TTW

RED	LED2	LEDCHIPLED-0603-TTW
210	R1	R-US_R0805
1k	R2	R-US_R0805
MA04-1	SV1	MA04-1

Once the custom board was received in the mail, the team began the final preparations and testing phase on the board. There were a few components that were not available in JLCPCB's stock that were ordered separately through DigiKey. With that being said, both surface mount and through-hole soldering were performed on the remaining components. The electrical team had sufficient experience performing through-hole soldering in the past, but had only observed surface mount soldering. Thus, another very beneficial learning experience was obtained after successfully soldering all of the components. The procedure taken for surface mount soldering the components is as follows:

1. Spread solder paste on the bare pads on the board where each component will be placed.
2. Place each component onto their respective pads with tweezers.
3. Place board onto heated bed to begin solder reflow process. See Fig. 2.34 and Fig. 2.35. Use an infrared thermometer to observe the temperature of the pads during the process.
4. Once the reflow process is complete, remove the board from the heated bed to cool the board to room temperature.

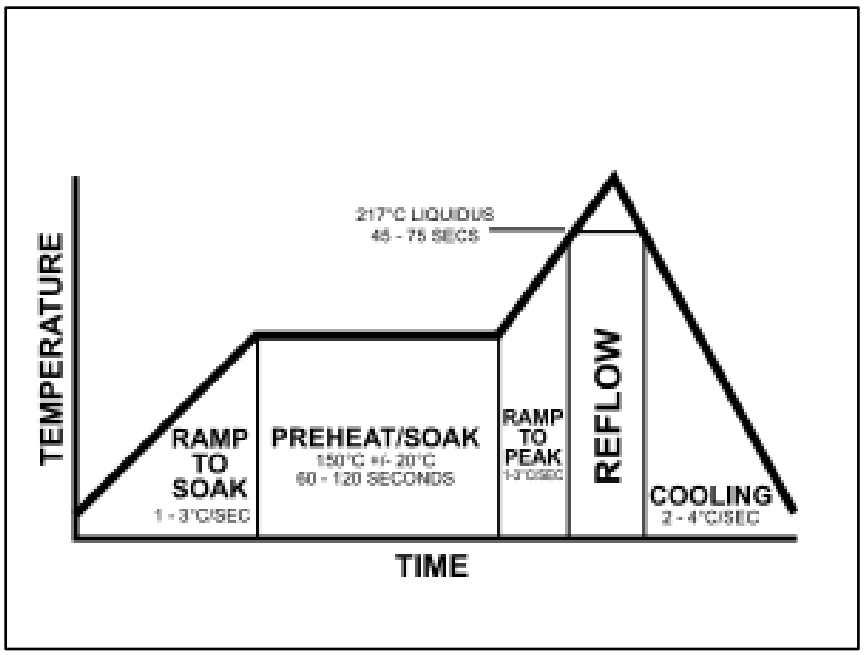


Fig. 2.34. Reflow Process[24]

Fig. 2.35. Performing Reflow Process with Heated Bed

After surface mount soldering was complete, the team began to test the connections with a multimeter. The final PCB can be found as Fig. 2.36. Once all

connections were confirmed with the schematic, the board was to an ST-Link debugger as seen in Fig. 2.37. The debugger was used in an attempt to flash the software to the board. However, the team faced another obstacle during this phase. While trying to flash the software to the board, a connection error would occur between the debugger and the PCB. After doing some research on the error, the team realized that the VDDA pin on the microcontroller shown in Fig. 2.38 must be connected to VDD (+3.3V) in order for software to be flashed to the board. It is believed that this connection was not made in the design, causing the issue.

Fig. 2.36. Final Custom PCB

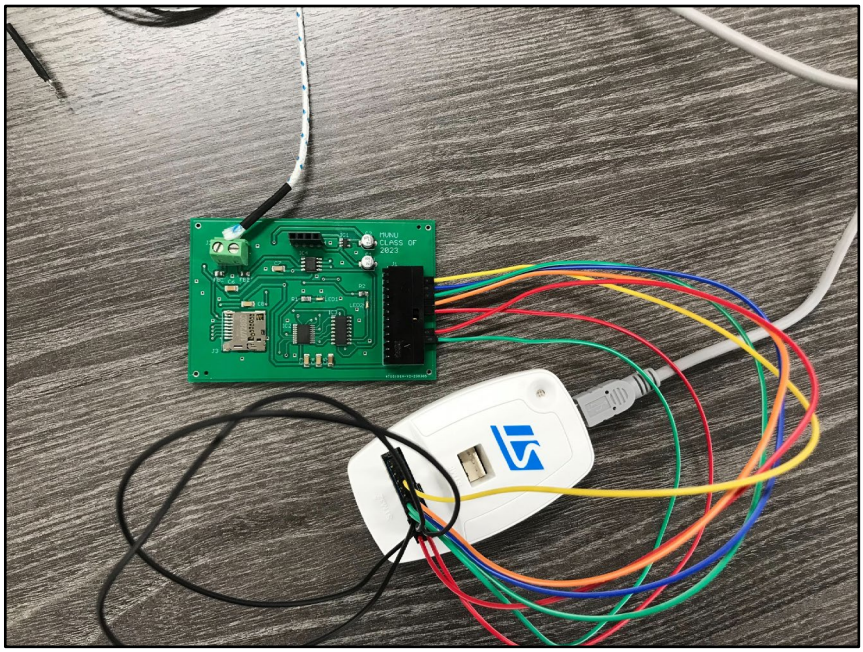


Fig. 2.37. Custom PCB with ST-Link V2 Debugger

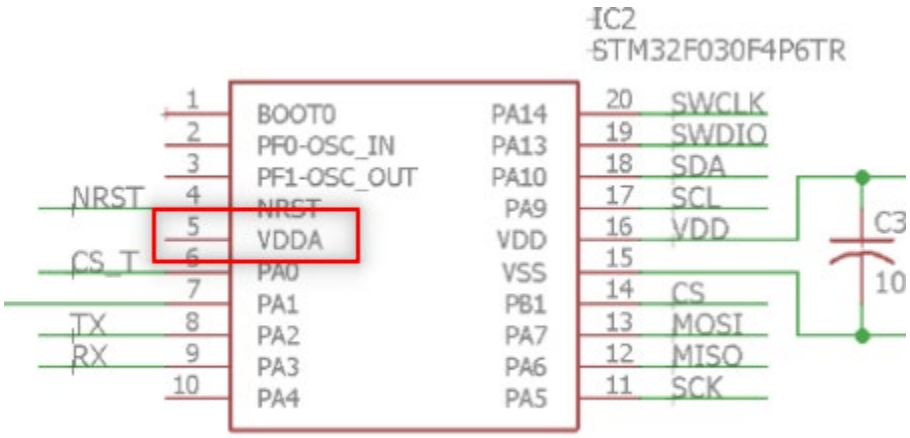


Fig. 2.38. VDDA Pin on Schematic Page 1

After an attempt to connect the VDDA pin to the VDD node by hand soldering it to a GPIO pin that was connected to VDD, it was realized that a direct connection was needed to VDD. The electrical team then acquired one of the extra custom PCBs sent from JLCPCB and restarted the soldering process. Once the second PCB had been

soldered, one of the electrical team members successfully soldered a single strand of copper wire to the VDDA pin to be connected to VDD. The connection can be seen in Fig. 2.39 and 2.40.

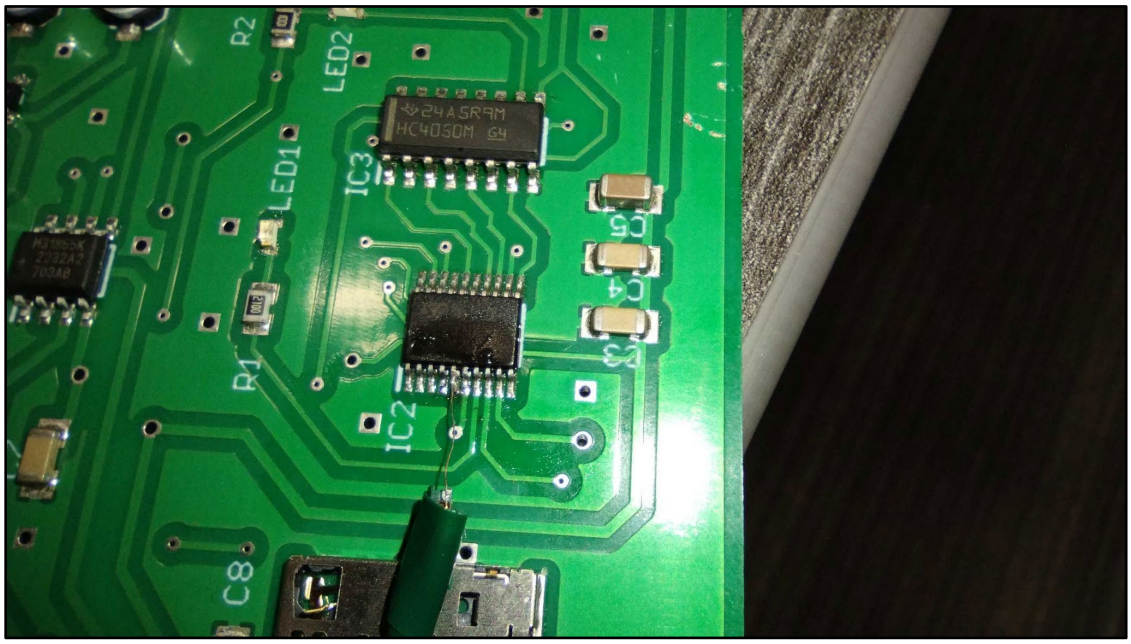


Fig. 2.39. Solder Connection to VDDA

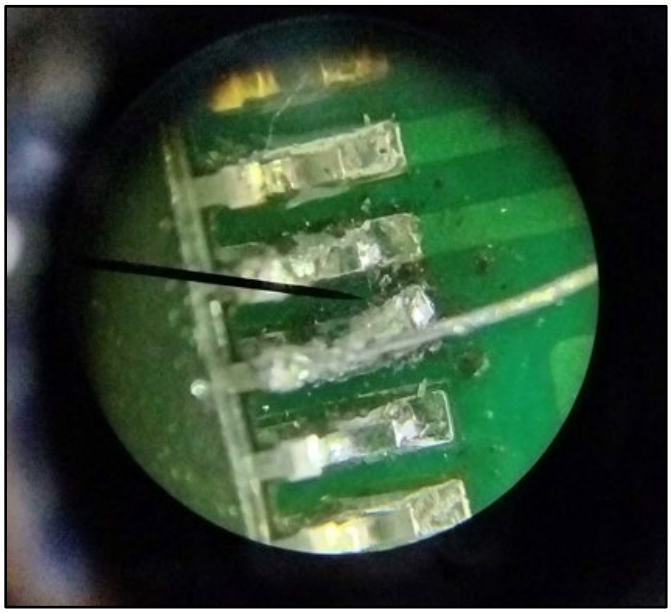


Fig. 2.40. Solder Connection to VDDA under Microscope

After the connection was made between VDD and VDDA, the team then connected the board to a DC power supply and the debugger on the STMicroelectronics Nucleo-L476RG board used for prototyping. The team then tried to flash software to the board again. This time around, the attempt was successful, which proved the suspicion with the VDDA pin. The setup for successful flashing can be found in Fig. 2.41.

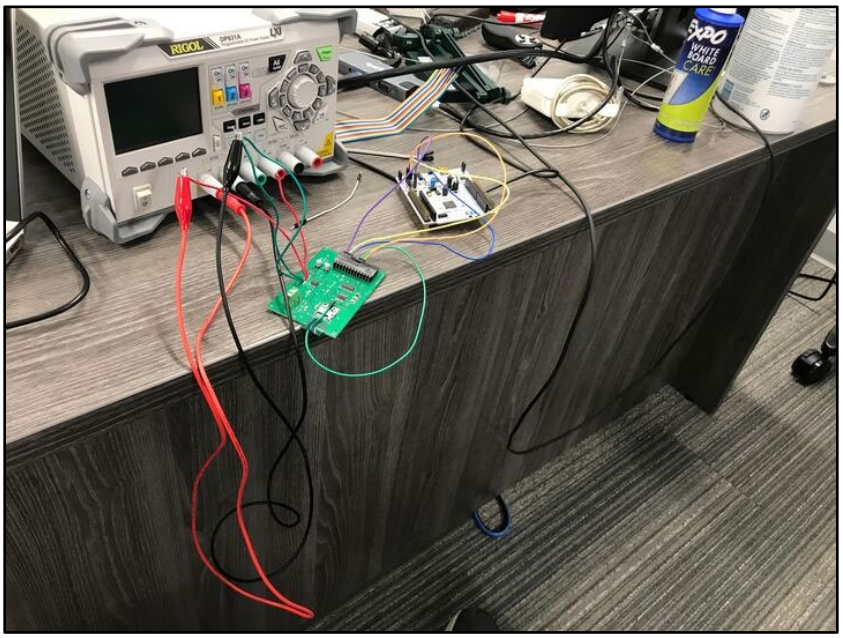


Fig. 2.41. Software Flashing Setup

Although the team was successful at flashing software to the board, another issue arose. The microcontroller onboard the PCB did not offer enough flash memory to handle the hardware abstraction layer (HAL) drivers for the I^2C and SPI communication protocols. Therefore, the sensor data could not be read. Thus, the root cause was the incorrect selection of a microcontroller for this application. However, after discussing a solution, the team decided to make use of the breakout boards that were used in the prototyping phase. The team shifted the battery temperature monitoring capabilities to the Raspberry Pi board since the Raspberry Pi had the bandwidth and pinouts available to add this function with ease.

In conclusion, the custom PCB had come a long way throughout the year. There were several setbacks and errors that arose, but for every error, the team found a solution. Looking ahead, in order for the PCB to function as it did in the prototyping phase a few adjustments have to be made. Firstly, software will need to be successfully written to enable use of a microSD card. Secondly, a new microcontroller will need to be selected that has at least 32KB of flash memory and the VDDA pin must be connected to the VDD node. Lastly, the electrical team recommended adding more through-holes for a row of male header pins so that more GPIO pins can be used if more functionality is developed within the software. Although the team was not able to implement the board into the UAV, a fully functional custom PCB was almost made. The experience that was gained in writing software for various sensors using an industrial microcontroller, surface mount soldering, circuit board design, and electronics troubleshooting was very beneficial to the growth of the electrical team as inexperienced engineers.

As mentioned previously, the second custom design that the electrical team worked on was a handheld weather station. The weather station was designed to measure temperature, relative humidity, and wind speed. It was also designed to be compact for easy handling. The team used an Arduino Nano as the brain of the weather station system. Along with the Nano, the Sparkfun Si7021 breakout board was used to measure temperature and relative humidity, and an anemometer from Adafruit was used to measure wind speed. The system is powered by a 2-cell LiPo battery and the sensor data is displayed on an OLED display. The user has the option to read temperature in degrees Fahrenheit and degrees Celsius as well as wind speed in meters per second (m/s), kilometers per hour (km/h), knots, and miles per hour (mph). Each unit can be read by turning the knob on a potentiometer. The wiring diagram for the device can be found as Fig. 2.42. The small black sensor in the wiring diagram represents the anemometer.

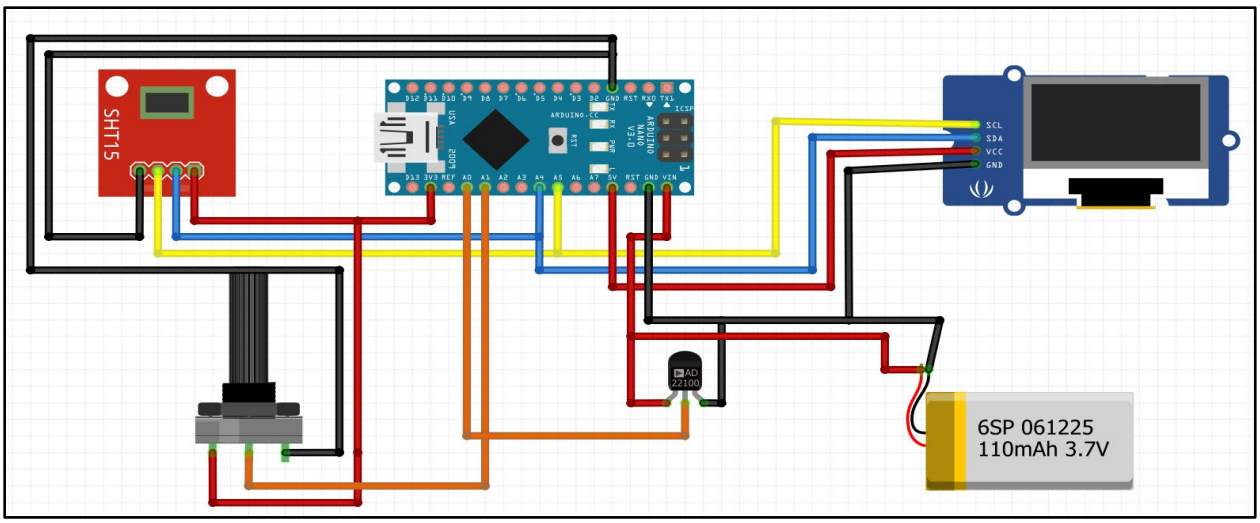


Fig. 2.42. Handheld Weather Station Wiring Diagram

Once a plan was initially developed for the weather station, the components, excluding the anemometer, were connected to a breadboard for the software design phase. The written code can be found in the Appendices Section of this report.

After the Arduino code was written, the components were moved to perfboard and connected using through-hole soldering techniques. The initial connections on the perfboard can be found in Fig. 2.43.

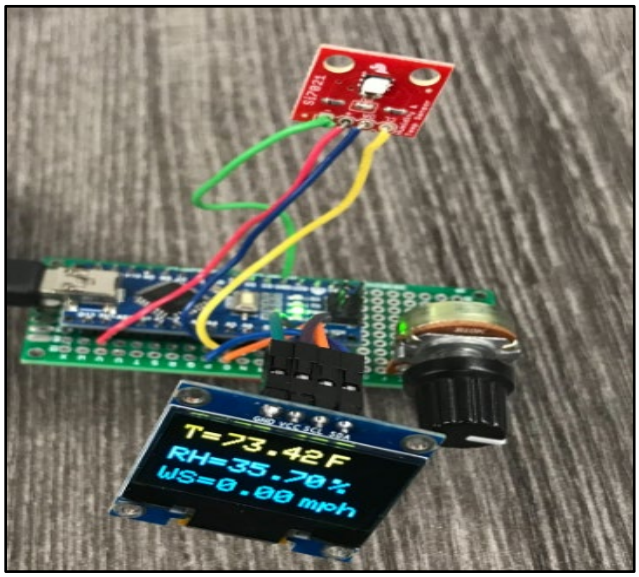


Fig. 2.43. Weather Station Perfboard Setup

In parallel with making the solder connections on perfboard, a design was started for the housing for the electrical components on Autodesk Inventor. The designed case and lid can be found in Fig. 2.44 and 2.45. The team made sure to limit the dimensions of the case to ensure a compact design that can be carried easily in one hand. Once the design of the housing was complete, a 3D printer was used to print the case and lid with PLA filament.

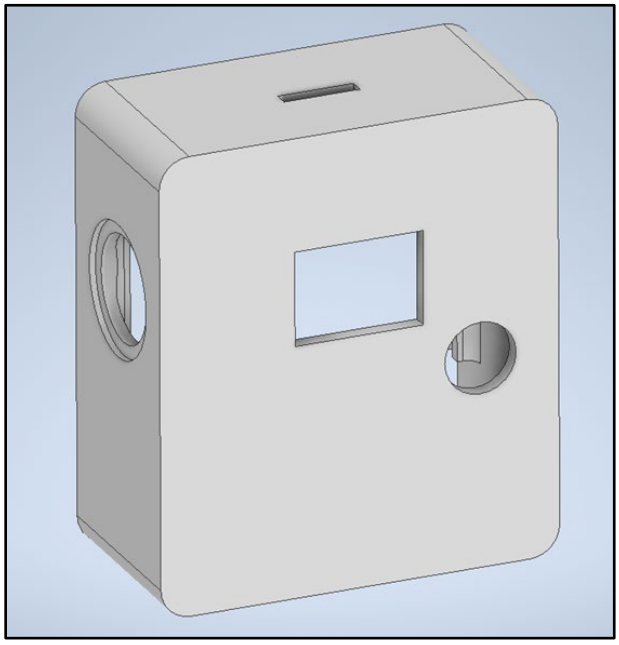


Fig. 2.44. Weather Station Case

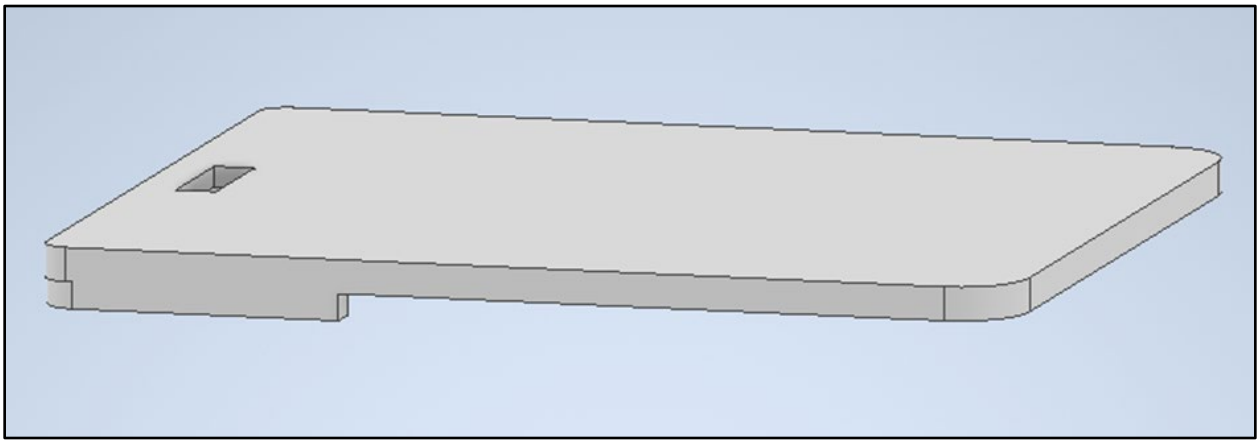


Fig. 2.45. Weather Station Lid

Once the case and lid were printed, the parts were sanded down to ensure that the components would fit properly. The perfboard containing the Nano and supporting components were then placed into the housing along with a power switch for easy battery saving capabilities. The components within the case can be found in Fig. 2.46. Before completing the final touches on the weather station, the accuracy of the anemometer had to be tested. The anemometer was tested by holding it out of the window of one of the team member's cars and comparing the wind speed data to the reading on the speedometer. The team ensured the series of tests were performed on very calm days with little to no wind. After performing the test four times and making changes to the wind speed conversion formula embedded in the Arduino code, the team successfully calibrated the anemometer data to the speedometer data. The wind speed data is +/- 2mph off when the wind speed is below 30 mph and is +/- 5 mph off when wind speeds exceed 30 mph. Once the series of wind speed tests were complete, a transparent plastic cover was placed over the OLED display and the finishing touches were made to the physical design. The final weather station can be found in Fig. 2.47. The same process was repeated for the assembly of the second weather station. The second weather station contains all of the components except for the anemometer.

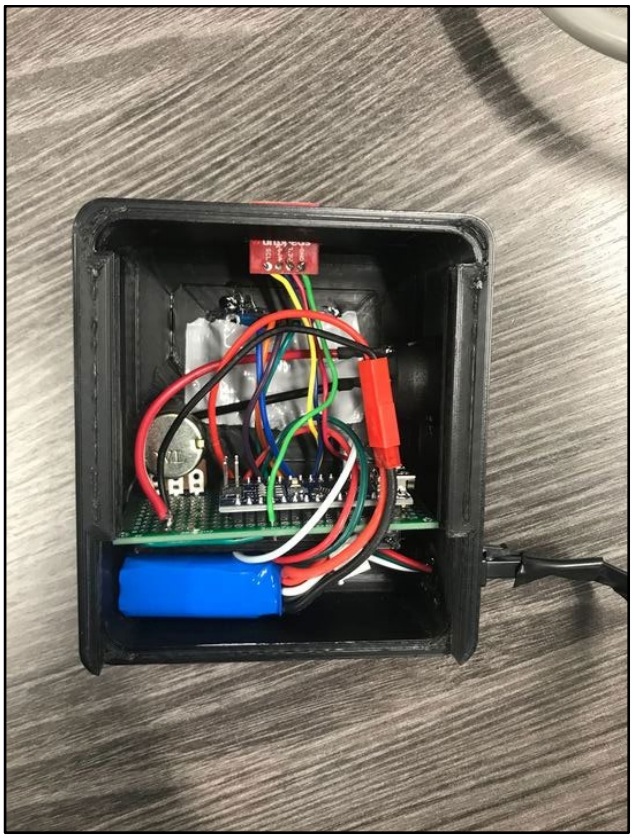


Fig. 2.46. Inside of Weather Station Case

Fig. 2.47. Weather Station (Full Assembly)

2.2 Imagery Team

Crop Analysis

The main task for the UAV is to acquire information on the health of crops. The imagery team has focused on this requirement by first examining how crop analysis works. Normalized Difference Vegetation Index (NDVI) is the most common method for analyzing plant health and other important characteristics for evaluating overall crop health. Other similar indices such as GNDVI (Green-Normalized Difference Vegetation Index), OSAVI (Optimized Soil Adjusted Vegetation Index), and NDRE (Normalized Difference Red-Edge) can be used to evaluate plant health in a variety of conditions and soil quality. These indices are all based upon the principle that healthy vegetation absorbs more visible light and reflects more near-infrared (NIR) light.[25] NDVI is calculated by taking the difference between the NIR and red reflectance values of an area and dividing it by their sum.

$$NDVI = \frac{(NIR-Red)}{(NIR+Red)} \quad \text{(Eq. 2.6)}$$

This produces a value between -1 and 1, with higher values indicating more healthy and vigorous vegetation. The analysis of the crops can provide valuable information to farmers about weak areas in their field, stressed crops, and pre and post comparisons for pesticide treatments. Each index requires acquisition of specific wavelengths of light for accurate calculations. For example, NDVI requires wavelengths of light at 670 nm (deep red in visible spectrum) and 800 nm (NIR region). Therefore, a camera system must be developed that is sensitive to both visible and NIR wavelengths in the electromagnetic spectrum.

Camera System

The first stage was to determine the most accurate and feasible system that could capture the necessary wavelengths of light needed for NDVI analysis. An options analysis approach was taken to resolve any ambiguity between the caveats of each system. A broad approach can be seen in Table 2.2, where four systems were established

and compared on the merits of NDVI accuracy, price, complexity, post-processing workload, general advantages, and drawbacks. In this table, a “Full Spectrum” camera refers to a camera that has the built in IR filter removed, therefore having unrestricted access to all wavelengths of light, hence the name. Furthermore, a single camera with a filter switch is a full spectrum camera with a filter switching mechanism. The filter mechanism would start with an IR pass filter, travel across the field and take all pictures, then come back to the same starting point. At this point, the filter would be switched to an IR block filter and RGB photos would be taken. This method would require twice the in-air flight time and increase battery usage, but only require one camera to be purchased.

Digital cameras are intrinsically sensitive to infrared wavelengths, which would interfere with normal RGB photography by confusing autofocus calculations or softening the image.[26] Softening of the image occurs since IR is refracted less compared to RGB wavelengths, which creates a non-perfect overlap of channels. Modern cameras come with a Bayer filter, which divides the photosensors into red, green, and blue regions. Since the Bayer filters in most digital cameras absorb a fraction of the infrared light, these cameras are sometimes not as sensitive as dedicated infrared cameras. To prevent issues, most digital cameras possess an IR-cut filter to prevent wavelengths greater than 650-700 nm from reaching the sensor. In this case, a Quad Bayer Coding (QBC) color filter exists on the Arducam 64MP camera. Quad Bayer Coding uses a RGGB Filter (Red, Green, Green, Blue). The IR sensitivity of this system is high, but some IR is absorbed by the filter. Beyond 850 nm, no color will be observed, prior to that, false color may be applied to IR wavelengths around 700 nm. It is essential to remove IR interference within RGB images, as IR data will oversaturate reds, leading to color degradation.

Table 2.2. Camera Systems Options Analysis

Criteria	*Option 1 (Single RGB)*	*Option 2 (Single Full Spectrum)*	*Option 3 (Dual System IR and RGB)*	*Option 4 (Single Full Spectrum with Filter switch)*
NDVI Accuracy	INACURATE - (No NIR or IR data measured) Estimates NDVI data using algorithms	SEMI-ACCURATE - With two-band-pass filter, but pure Red and NIR channels cannot be obtained precisely at the same time	MOST ACCURATE - Captures pure NIR and Red on separate cameras	MOST ACCURATE - Captures pure NIR and Red on single camera
Price	LOWEST (50$ or lower)	LOW (55$ or lower)	HIGHEST (125$)	HIGH (80$)
Complexity	SIMPLE - no filters or mechanism needed	SIMPLE - no filter needed, but carefully take filter out of RGB	COMPLEX - essentially doubling complexity of single camera system	MOST COMPLEX - need to build mechanism that can interchange two filters automatically
Post-Processing	Need algorithms to extract data, requires large processing power (cloud-based)	Need to extract IR and RGB channels from single image	Need software to merge and stack images properly	Need software to control mechanism and merging images
Advantages	-Cheap -Easy to install and operate	-Cheap -easy to operate -captures more accurate data than RGB	- Most accurate data	- Cheaper than dual camera while still getting accurate data
Drawbacks	Inaccurate data	Cannot achieve pure NDVI data	Expensive, lots of complexity on processing side	Requires two passes (NIR pass and RGB pass)

To better visualize this data, a spider/radar chart was constructed to compare the advantageous characteristics of each system (Fig. 2.48).

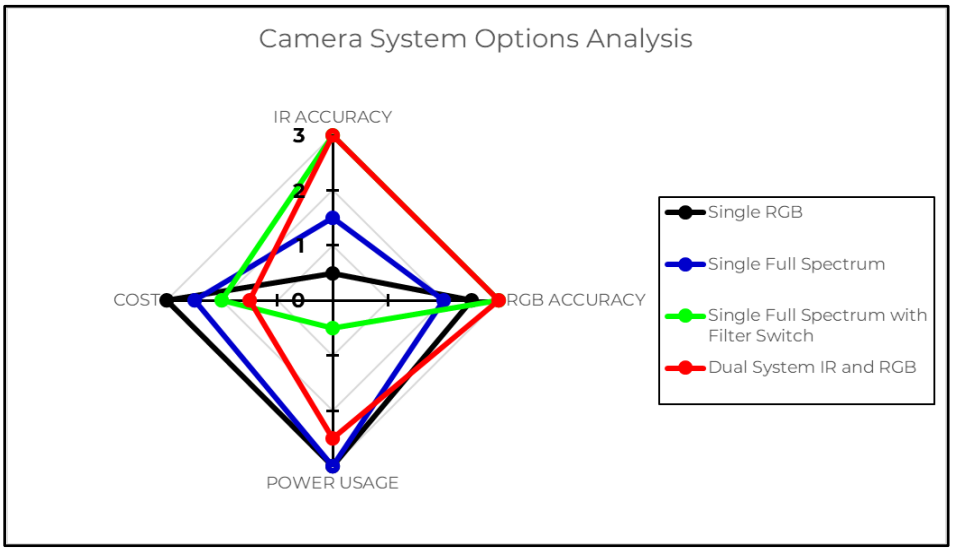


Fig. 2.48. Spider Chart of Camera System Options Analysis

The group prioritized the importance of data accuracy compared to the other features due to the nature of the project. The goal of the UAV is to assess plant health; therefore it is of paramount value to receive accurate data, otherwise the entire project fails with respect to the overall objective. Consequently, the team has concluded that a dual-camera system will suit the project's aim most effectively, at the cost of pricing. The system will consist of a dedicated RGB capable of capturing wavelengths up to ~ 700 nm. The optimal wavelength for NDVI calculation is 650 ± 10 nm. Therefore a 700 nm IR-Cut filter (blocks all spectral emissions that have a wavelength greater than 700 nm), will sufficiently acquire the required red spectrum. The other camera will be identical in composition to the RGB camera, except that the IR-Cut filter will be replaced with an IR pass filter. An IR pass filter, otherwise known as an IR bandpass, blocks all visible light and all emissions less than 720 nm. The optimal wavelength for NDVI calculation for the IR portion is 770±15 nm. A schematic of the system can be seen in Fig. 2.49.

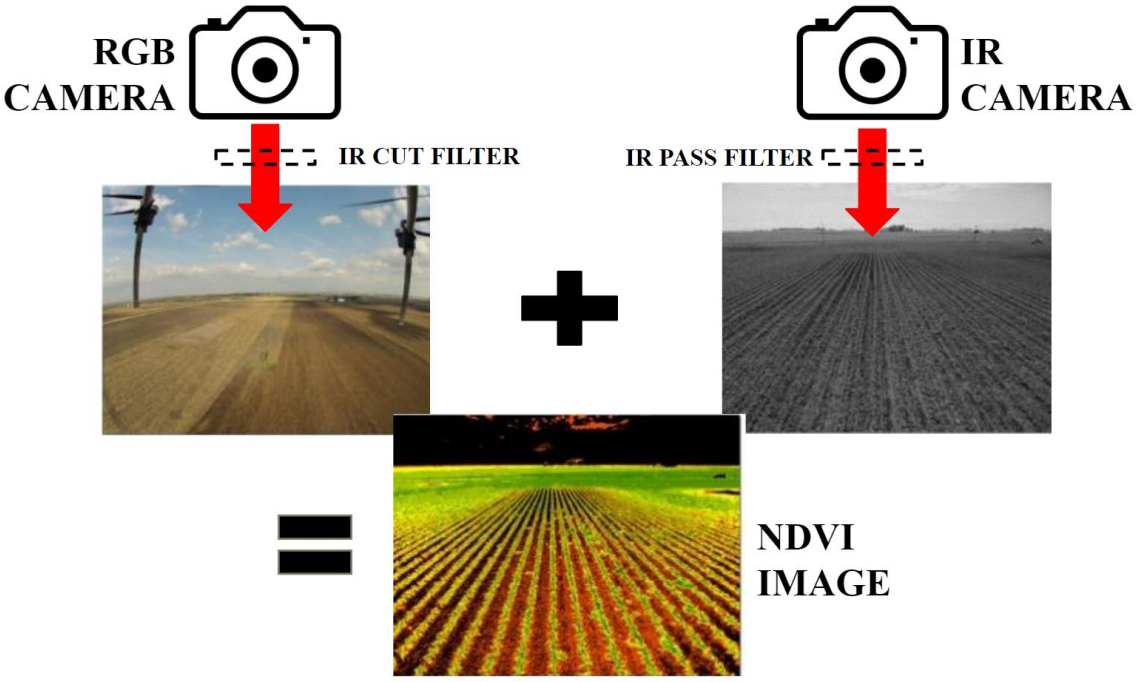


Fig. 2.49. Dual Camera System and Image Merging (RGB, IR, and NDVI images).[27]

A two-camera system is more desirable compared to a single camera because it allows for more accurate and precise measurements. A two-camera system can capture more data points (full resolution is devoted to either RGB and NIR, whereas a single camera would have to split resolution). With a proposed camera system determined, it was now possible to choose a camera that is capable of capturing the respective RGB and IR fields while achieving adequate resolution. When it came to comparing cameras, features that were taken into consideration were picture quality, size, weight, and price. The deciding feature regarded the presence of IR-cut filters. Prior to Spring 2023, there were no camera options that left out IR-cut filters, meaning that a disassembly of the cameras would be conducted to remove the built in filters. However, Raspberry Pi released their 12MP camera module 3 with the option to leave out the built in IR filter in February. This quickly became the best option for the dual camera system because no meticulous disassembly would be needed, which has a high probability to damage the camera sensors. With a reasonable price and size, the last thing to check was the resolution capabilities. Table 2.3 indicates the resolution calculations of the NoIR 12MP camera.

Table 2.3. Calculations and Properties for NoIR 12MP Module 3 (Wide).

NoIR 12 MP (Wide)				
Alt (ft)	**Field Width (ft)**	**Picture Area (ft^2)**	**Camera Specs**	
100	246.98	232389.37	Resolution Width (Pixels)	4608
200	493.96	929557.50	Resolution Height (Pixels)	2592
300	740.94	2091504.36	FOV Wide (degrees)	102
400	987.92	3718229.98	FOV Vertical (degrees)	67
Alt (ft)	**Field Height (ft)**	**Pixel Density ($pixels/ft^2$)**		
100	940.93	458.24		

NoIR 12 MP (Wide)				
Alt (ft)	**Field Width (ft)**	**Picture Area (ft^2)**	**Camera Specs**	
100	246.98	232389.37	Resolution Width (Pixels)	4608
200	1881.85	114.56		
300	2822.78	50.92		
400	3763.70	28.64		

The math for pixel density calculations comes from a governing equation of the project, given below.

$$\varphi_p = \frac{h_p^2 + w_p^2}{4d^2 tan^2(\frac{1}{2}\alpha)} \quad \text{(Eq. 2.7)}$$

where φ_p is pixel density, h is number of vertical pixels, w is number of horizontal pixels, d is altitude, and α is angle of view. Numbers of pixels vertically and horizontally along with angle of view are all given specs of the camera. Pixel density is necessary to know so that images can be captured with enough detail to allow for proper spectral analysis. From the pixel density column in Table 2.3, it is clear that this camera exceeds the minimum of 1 pixel per square foot by a large margin, especially at the expected operating altitudes (~200 ft). This camera also comes with autofocus, making it convenient for the user to not have to worry about manual focus given different flying altitudes.

With the resolution requirement met and the ease of the camera to be implemented into the exterior filtered dual system, the NoIR 12MP camera module was the definite choice. The camera is full-spectrum with no filter, meaning the team can control which light wavelengths are captured based on the exterior filter selections.

Spectral Analysis

A theoretical spectral analysis was conducted using the manufacturer provided transmissivity of the respective filter. These results were compiled into a graph (Fig. 2.50) and plotted transmission percentage versus wavelength in nanometers.

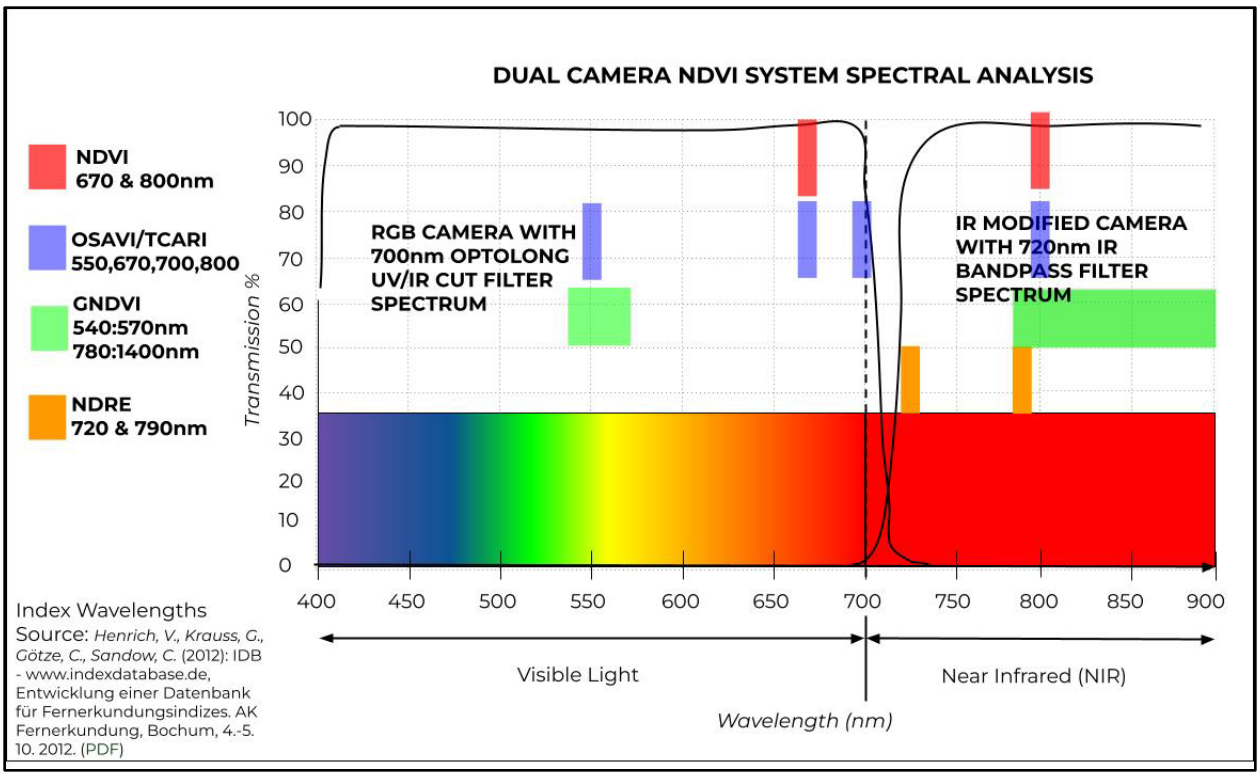


Fig. 2.50. Theoretical Spectral Analysis of Dual Camera System.[28]

Various crop vegetation indices are labeled and the required wavelengths necessary for its respective calculations are color coordinated. The proposed camera system and filters cover the necessary wavelengths needed for all common indices. A small gap is present around 700-720 nm due to the necessity for a margin of error between the two cameras. If both cameras are capturing the same wavelength, in post-processing, that data will be stacked and amplified, which will make that specific wavelength appear stronger compared to the others.

Each curve in the spectral graph represents the spectral power distribution (SPD) of the colors present in the cropped image. The SPD is a measure of the relative power of light at different wavelengths that is emitted, transmitted, or reflected by a light source or surface. In the context of the RGB image, the SPD represents the relative amount of red, green, and blue light that is present in the image at each wavelength. The resulting graph shows how the relative intensities of light change across the visible spectrum, with peaks indicating which wavelengths are more strongly represented in the image. In the program, RGB to XYZ tristimulus conversion is performed in order to convert an RGB image into

a spectral curve. This process involves converting RGB color values into a set of three tristimulus values (X, Y, and Z) that represent the amount of energy in the red, green, and blue spectral regions that are perceived by the human eye. The conversion from RGB to XYZ is necessary because the spectral data needed for the subsequent analysis cannot be obtained directly from the RGB color space. Instead, the XYZ tristimulus values provide a basis for calculating the spectral data using CIE color matching functions (in this case, CIE 1931), which describe the human eye's sensitivity to different wavelengths of light. Furthermore, normalization of pixel values to the range of 0 to 1 is conducted in preprocessing in the program. Normalizing the pixel values can help to ensure that the image data has a consistent range of values across different images, which can be important for comparing and analyzing images. Additionally, normalization can help to reduce the impact of variations in lighting or exposure on the image data, making it easier to extract useful information from the image.

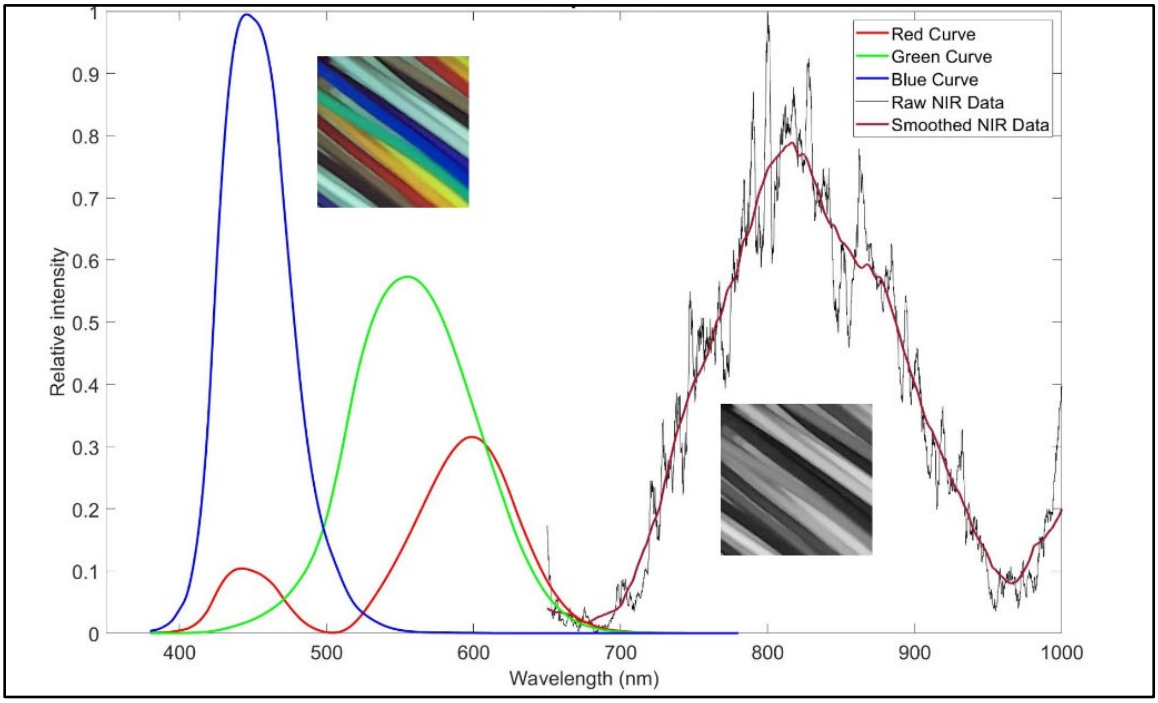


Fig. 2.51. Experimental Spectral Analysis of Dual Camera System.

The NIR region gives an estimate of the intensity of light versus wavelength in the NIR region, based on the input IR image. However, it is important to note that the

accuracy of the graph depends on the quality of the input image, as well as the accuracy of the calibration of the camera and amount of available NIR light present in the room. Additionally, the program assumes that the NIR information is contained in the red channel of the RGB image, which may not always be the case. In some cases, other color channels may contain some NIR information, or the NIR information may be contained in channels outside of the RGB color space. Therefore, the graph generated by the program can serve as a general indication of the intensity of light versus wavelength in the NIR region, but for more accurate measurements, it may be necessary to use specialized equipment designed for NIR imaging, and to perform appropriate calibration and processing of the data. The intensity of the NIR data may appear to increase at around 1000 nm due to a phenomenon called "detector saturation". In general, the intensity of the NIR radiation decreases as the wavelength increases, due to the increasing energy of the photons at shorter wavelengths. However, at very long wavelengths (above 1000 nm), the intensity may appear to increase again due to the limitations of the detector used to capture the NIR image. Many detectors have a "saturation limit", which is the maximum intensity of radiation that the detector can measure before it becomes saturated, or unable to accurately detect further increases in intensity. This limit may be reached at longer wavelengths in the NIR region, leading to a plateau or even a slight increase in the measured intensity.

The curve present in Fig. 2.51 represents the data present in a cropped image of electrical wires that has a variety of colors. The variety of colors present allow for multiple different wavelengths of light to be in the picture. It is important to note that due to the nature of the material of the wires and availability of light within an artificially lit room, the wavelengths of light around 680-700 nm is low. Also, XYZ tristimulus color functions are less sensitive to red wavelengths because human eyes are more sensitive to blue and green. Therefore, to better suit the reality of what an actual human eye would see, red is much lower in the spectral analysis. However, data is still present at these wavelengths, just small amounts. A normalized RGB spectral analysis across each channel can be seen in Fig. 2.52.

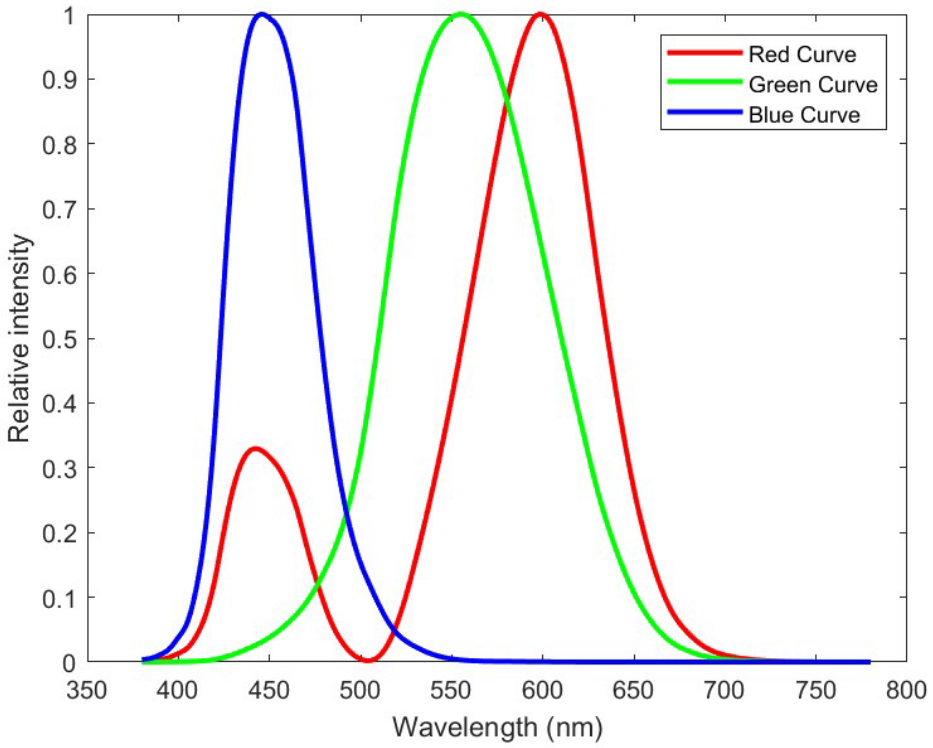


Fig. 2.52. Normalized RGB Spectral Analysis.

Other than the electrical wires, pictures of vegetation were captured to test completion and accuracy of the NDVI analysis. With the help of software, the images taken through both the IR-pass filter and the IR-cut filter were stitched together. Using MATLAB, the images were optimized and aligned, then the required data was extrapolated from the images (Appendix D). Once overlaid, the wavelengths captured in the photos were analyzed and inserted into Eq. 2.6. Based on the values returned from this process, the same area where the wavelength was taken from the photo was assigned a color. The more red the color is, the lower the NDVI value is (minimum of -1). The more green the color is, the higher the NDVI value (maximum of 1). As described earlier, the higher NDVI values indicate healthier plants. Thus, in NDVI images, the green areas show healthier plants while the red areas show unhealthy plants. An example of an NDVI image analyzed is shown in Fig. 2.53.

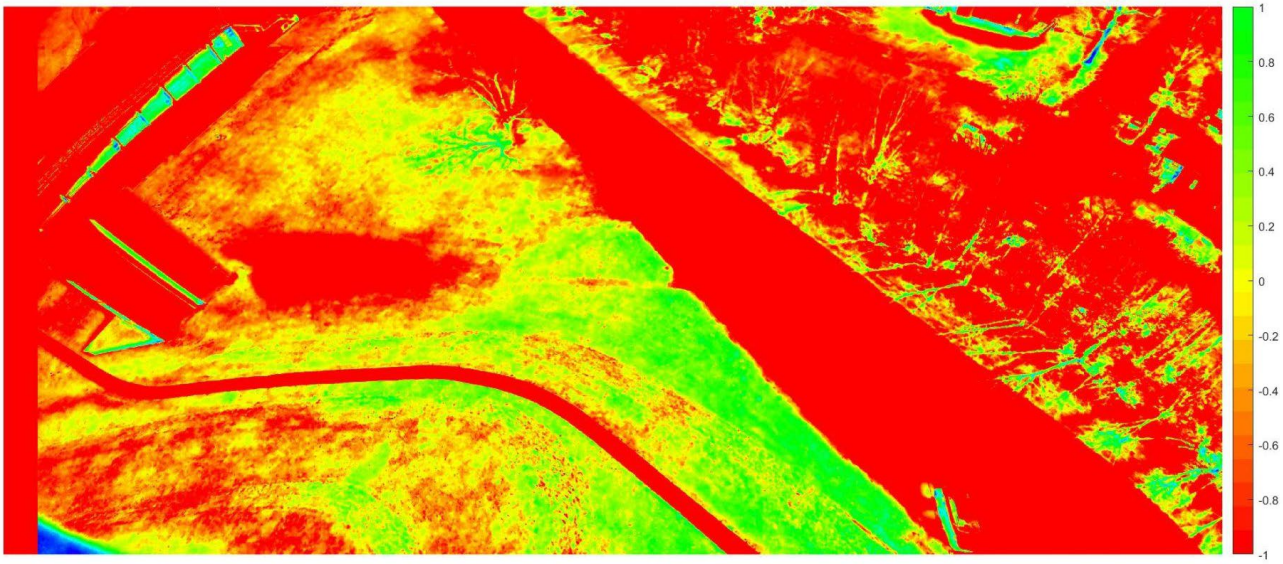


Fig. 2.53. NDVI Analysis of Grass at Ariel Park.

The green areas in Fig. 2.53 indicate areas of healthy grass found at the park. The dark red areas show either man made objects, such as buildings or roads, or areas of no vegetation. The in between colors, areas of orange and yellow, show areas of grass that are unhealthy or stressed. Blue regions are extremities that are caused by reflections and should be ignored. The original images used in this NDVI image can be seen in Appendix E. Based on a purely visual observational analysis of the original RGB photo, the NDVI image appears accurate with regards to plant health. Areas of brownish colored grass (unhealthy) appear yellow or orange in the NDVI image, which is valuable information. Shadows cause several inaccuracies in NDVI, as the change in reflectance of the visible spectrum creates areas of supposed vigor. Therefore, shadowed areas (tree branches, side of building, ravines) may appear healthier than anticipated. Overall, the camera system and NDVI analysis tests show accurate evidence in determining the health of vegetation.

Camera Case

The design and construction of a camera case to house the two NoIR 12MP cameras and the two filters (IR-Pass and IR-Block) was necessary. The camera case is a two-piece design of which includes the base with a snap on lid to make assembly simple. The base has two cylinders serving as viewing ports for the two cameras. The ports are

sized to friction fit the exterior camera filters. Four screw holes exist for each camera to be secured to the base. The top part of the base includes a slit where a screen protector is inserted. The lid includes a rim which allows for it to snap onto the base. On top of the lid sits four pegs to allow for the mounting of the Raspberry Pi. The 3D printed case with its entire assembly is pictured in Fig. 2.54.

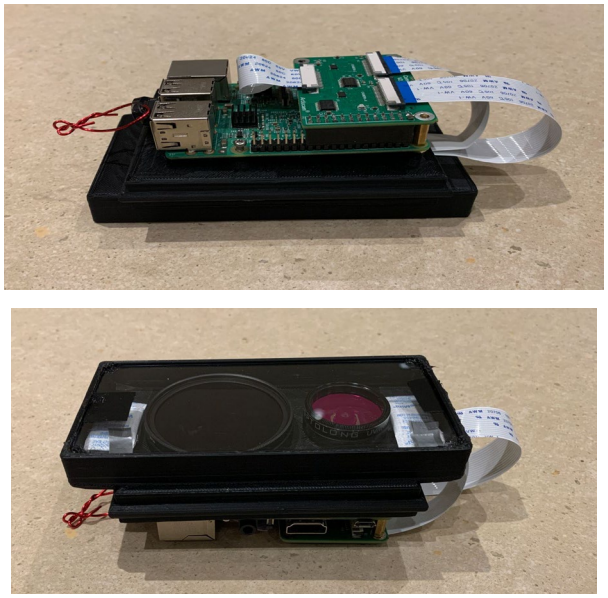


Fig. 2.54. Camera Case Assembly

Figure 2.54 includes the Raspberry Pi mounted with all components secured in the case. Considering the vulnerability of the imaging components, damage protection and moisture control were necessary. A screen protector with 9H hardness was mounted in front of the camera filters to protect from weather and physical contact. The 9H hardness falls on the Mohs hardness scale right below the diamond at 10H, which is one of the hardest things on earth. Thus, the screen protector will provide sufficient protection for the imagery components. Silica packets were secured in the case to aid in controlling moisture from accumulating on the filters and lenses. The case was also sealed with silicone paste in order to minimize exposure to outside elements. This case is inserted into the belly of the UAV, where a hole was cut out for the cameras to view the ground below.

Lessons Learned

Prototyping included the deconstruction of an Arducam 64MP camera module, which was the camera selection chosen at the end of the fall semester. It was proven difficult to disassemble and reassemble the camera without damaging its sensor. It was thought that with better tooling and a controlled atmosphere from something such as a clean box, that the disassembly of the camera filters could be successful. After multiple test runs and real runs, it was still evident that the removal of the camera filter could not be done with a high chance of success. Thus, another camera option was selected. The team learned that camera sensors are extremely sensitive and that it takes meticulous care to work on small scale imaging components.

2.3 Design and Manufacturing Team

The main goals for the design and manufacturing team were firstly to select a motor for the prototype and final UAV, secondly to set up a working prototype for testing the internal components, and thirdly to design and manufacture a fully custom UAV body.

Motor Selection & Testing

Selecting the correct motor and propeller was a key step in the process of building a functional UAV. The motor selection was based on the weight of the prototype and the estimated future weight of the final UAV. The goal was to obtain a thrust to weight ratio of over 1. This is to ensure the motor has enough thrust to lift the plane from takeoff. Once the UAV is at its intended altitude the required thrust will be significantly less as the UAV will be able to essentially glide as it follows its set path. The final motor selection was the Cobra C-2814/16. The motor specs can be seen in Table 2.4. The propeller and ESC were chosen based on the recommended data from the motor manufacturer. Cobra provided a table of propeller options based on the input voltage. The team selected the propeller recommended with the largest estimated thrust output when paired with the motor.The selected ESC was a 33A Cobra ESC. The selected propeller was an APC 9x4.5E and the propeller specifications with the combination of a 4-cell LiPo battery along with the C-2814/16 motor can be seen in Table 2.5.

Table 2.4. Motor specifications[29]

Motor	Weight	Outside Diameter	RPM Per Volt	Max Continuous Current	Max Power (4-Cell)	Motor Wind	Max Efficiency
C-2814/16	109 g	35 mm	1050	30 amps	450 Watts	16 Turn Delta	89%

Table 2.5. Propellor specifications[29]

Size	Input Voltage	RPM	Pitch Speed	Thrust	Thrust Efficiency
9x4.5E	14.8 V	12,262	52.3	1,567 g	4.54 g/W

After deciding on the motor and propeller, it was desirable to test the combination ourselves to determine the amount of thrust the motor could produce. The thrust was tested by creating an apparatus to hold the motor in a vertical position while sitting on a scale. The scale was zeroed with the motor apparatus sitting on it so any thrust produced from the motor could be measured. The testing apparatus can be seen in Fig. 2.55. Using the battery and controller, power was applied to the motor until a specific current draw to the motor was achieved. Starting at 5% of the max rated current of 26 A, the thrust was tested in intervals of 5% until 30% was reached. Due to most of the flight being below the 30% thrust level the test was given this upper limit. The data for the thrust test can be seen in Table 2.6. From the thrust test, it was found that the motor and propeller combination was actually producing more thrust than initially expected based on the data from the Cobra website. This will allow the UAV to achieve a larger thrust to weight ratio than expected which means there is a safety margin in case any extra weight needs added to the UAV.

Fig. 2.55. Thrust Test Apparatus

Table 2.6. Thrust Test Data

% of Max Rated Current	Current Draw (A)	Voltage (V)	Thrust (g)
5	1.3	2.44	84
10	2.59	3.78	182
15	3.95	5.02	322
20	5.28	5.93	450
25	6.49	6.4	520
30	7.84	7.36	674

Initial Prototype

In order to test the electronics and software components of the UAV before the final body was built, it was decided to purchase a foam plane body to use in the meantime. The body selected was a X-UAV Mini Talon which can be seen in Fig. 2.56.

Fig. 2.56. X-UAV MiniTalon[30]

This plane body is made out of EPO (Expanded Polyolefin) which makes it light and durable. The plane hull is also large enough to allow room for all the electrical equipment. After calibrating all of the servos and installing all of the needed hardware the team was ready for the test flight. During the first flight the plane was working properly along with all of the components until around 15 minutes into the flight the plane began losing communication. It is believed to be caused by the removable cover on the hull, which allows access into the compartment where all the electronics are located. Wires most likely came loose from too much airflow in the compartment. Eventually this caused a disruption in communication and the plane plummeted downward into the ground. Luckily, the foam body took most of the damage and all of the electronics and the battery remained in good condition. Pictures of the remains can be seen in Fig. 2.57.

Fig. 2.57. Test Plane Wreckage

Wing Design

In order to improve the efficiency of the final UAV, the team decided to choose the team's own airfoil shape to maximize the efficiency of the wing. This airfoil shape was used to create the ribs that form the wing. The following airfoils were selected and compared based on their common use for low-speed aircrafts and their simple designs for easier simulation and production later in the process. The airfoils compared were the NACA 1412, 2412, 4412, and 4415.

NACA 1412:

Specifications:

Max thickness 12% at 29.9% chord

Max camber 1% at 40% chord[31]

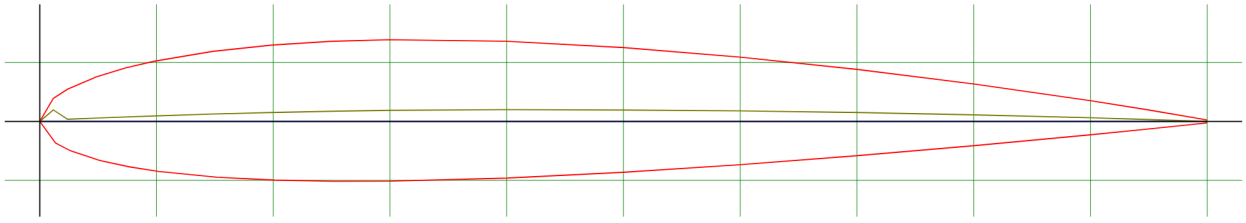


Fig. 2.58. Cross Section of NACA 1412 airfoil[31]

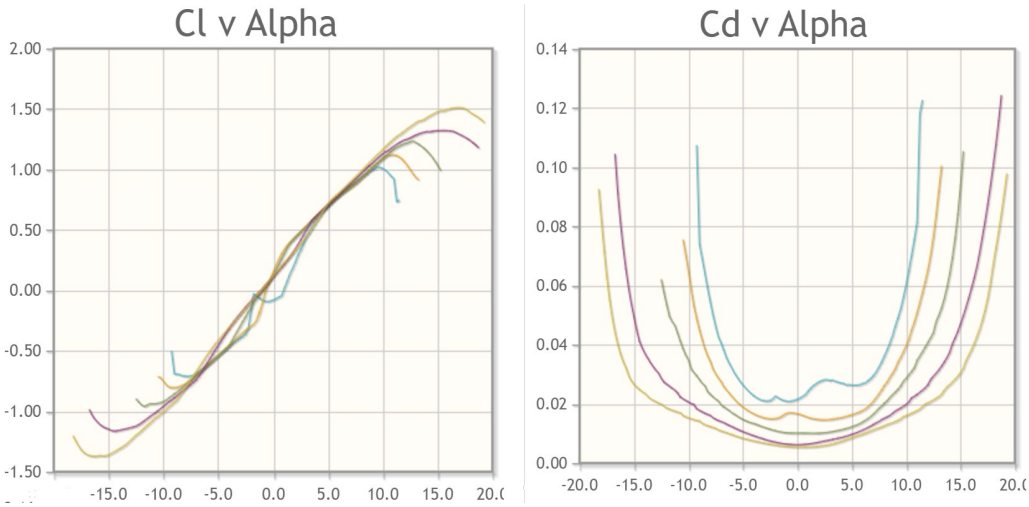


Fig. 2.59. Cl and Cd vs Alpha for 1412[31]

Plot	Airfoil	Reynolds #	Ncrit	Max Cl/Cd
☑	naca1412-il	50,000	9	29.6 at α=6.25°
☐	naca1412-il	50,000	5	31.2 at α=6°
☑	naca1412-il	100,000	9	44.1 at α=5.75°
☐	naca1412-il	100,000	5	43 at α=5.5°
☑	naca1412-il	200,000	9	57.6 at α=5.25°
☐	naca1412-il	200,000	5	53.3 at α=4.75°
☑	naca1412-il	500,000	9	73.5 at α=5°
☐	naca1412-il	500,000	5	66 at α=5.75°
☑	naca1412-il	1,000,000	9	83.8 at α=5°
☐	naca1412-il	1,000,000	5	78.6 at α=8.5°

Fig. 2.60. Legend for 1412 Graphs[31]

NACA 2412:

Specifications:

Max thickness 12% at 30% chord

Max camber 2% at 40% chord[31]

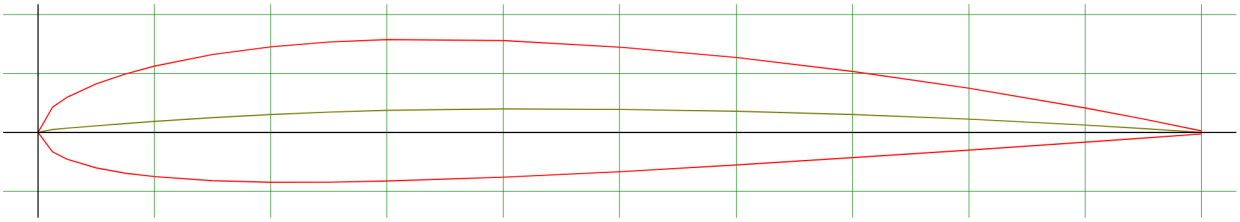


Fig. 2.61. Cross Section of NACA 2412 airfoil[31]

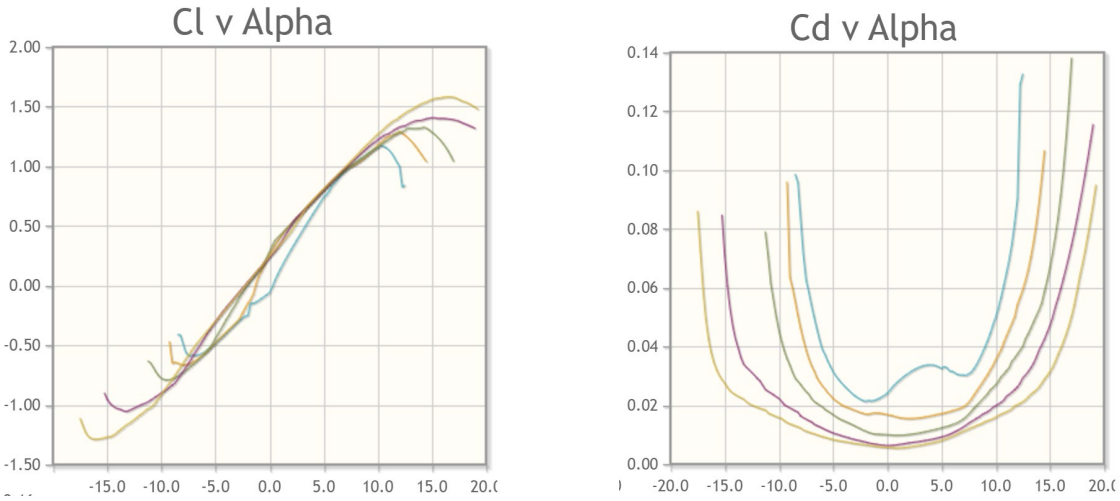


Fig. 2.62. Cl and Cd vs Alpha for 2412[31]

Plot	Airfoil	Reynolds #	Ncrit	Max Cl/Cd
☑	naca2412-il	50,000	9	32.5 at α=7.25°
☐	naca2412-il	50,000	5	34.6 at α=6.5°
☑	naca2412-il	100,000	9	50 at α=6.75°
☐	naca2412-il	100,000	5	49.4 at α=6°
☑	naca2412-il	200,000	9	66.6 at α=6°
☐	naca2412-il	200,000	5	62.6 at α=5.25°
☑	naca2412-il	500,000	9	87.3 at α=5°
☐	naca2412-il	500,000	5	78.3 at α=4°
☑	naca2412-il	1,000,000	9	101.4 at α=4.5°
☐	naca2412-il	1,000,000	5	87 at α=4.5°

Fig. 2.63. Legend for 2412 Graphs[31]

NACA 4412:

Specifications:

Max thickness 12% at 30% chord

Max camber 4% at 40% chord[31]

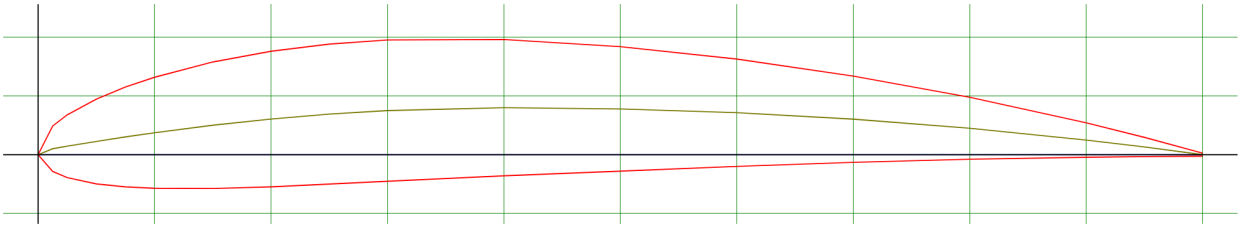


Fig. 2.64. Cross Section of NACA 4412 airfoil[31]

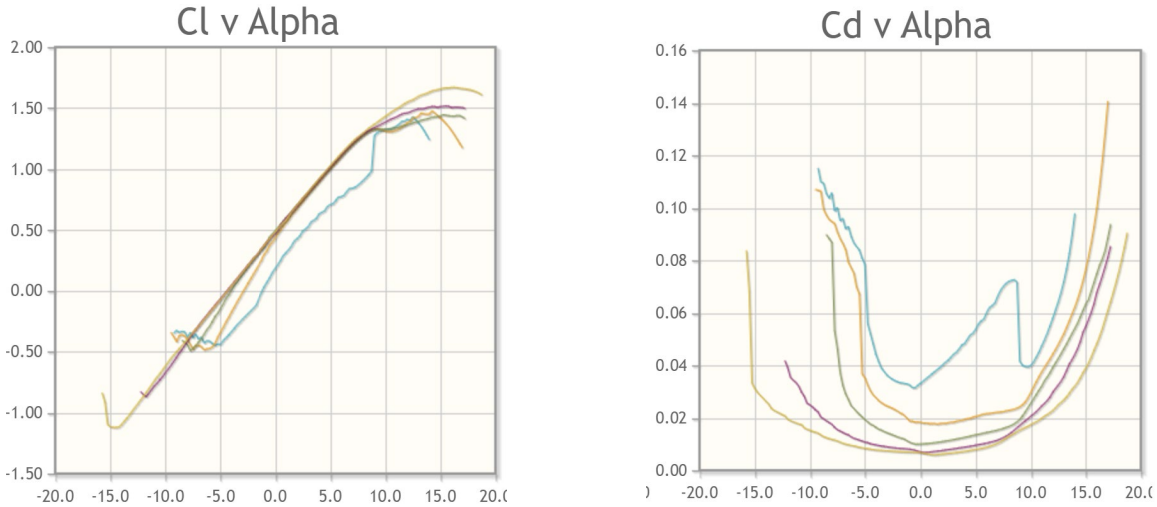


Fig. 2.65. Cl and Cd vs Alpha for 4412[31]

Plot	Airfoil	Reynolds #	Ncrit	Max Cl/Cd
☑	naca4412-il	50,000	9	33.4 at α=9.75°
☐	naca4412-il	50,000	5	36.1 at α=8.5°
☑	naca4412-il	100,000	9	56.1 at α=8.5°
☐	naca4412-il	100,000	5	57.4 at α=6.75°
☑	naca4412-il	200,000	9	78.1 at α=7°
☐	naca4412-il	200,000	5	76.6 at α=6°
☑	naca4412-il	500,000	9	107.5 at α=6°
☐	naca4412-il	500,000	5	101.1 at α=5°
☑	naca4412-il	1,000,000	9	129.4 at α=5.25°
☐	naca4412-il	1,000,000	5	119.2 at α=4.25°

Fig. 2.66. Legend for 4412 Graphs[31]

NACA 4415:

Specifications:

Max thickness 15% at 30.9% chord

Max camber 4% at 40.2% chord[31]

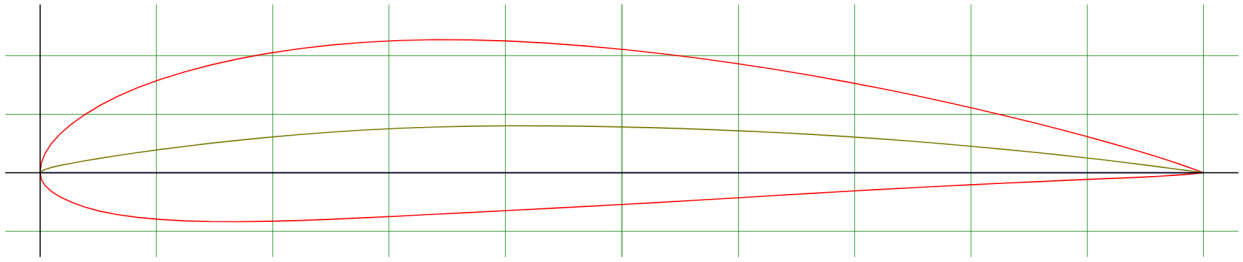


Fig. 2.67. Cross Section of NACA 4415 airfoil[31]

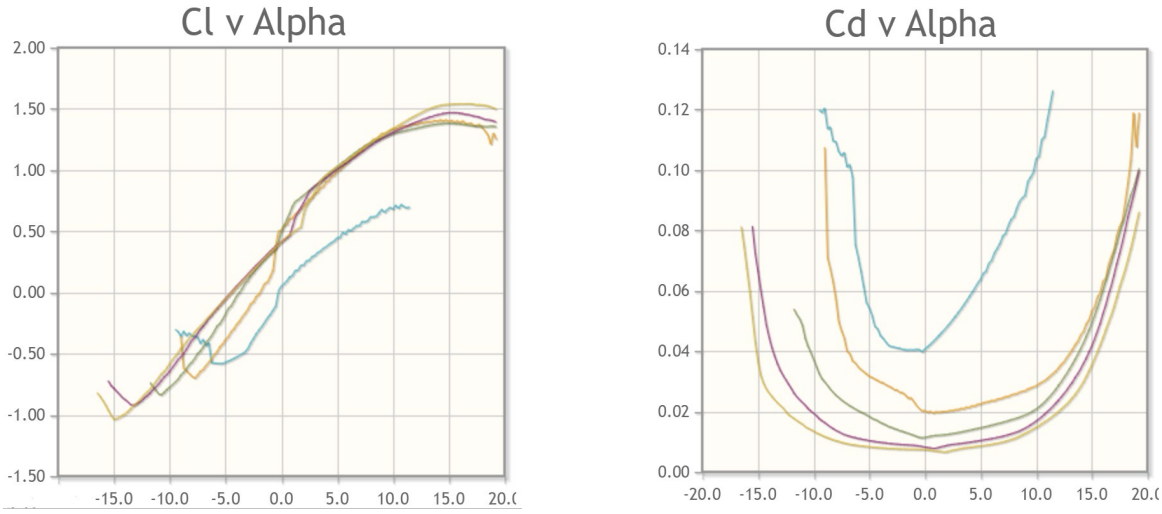


Fig. 2.68. Cl and Cd vs Alpha for 4415[31]

Plot	Airfoil	Reynolds #	Ncrit	Max Cl/Cd
☑	naca4415-il	50,000	9	7.4 at α=9°
☐	naca4415-il	50,000	5	27.4 at α=8.25°
☑	naca4415-il	100,000	9	48.4 at α=9°
☐	naca4415-il	100,000	5	50.7 at α=6.25°
☑	naca4415-il	200,000	9	71.1 at α=6.5°
☐	naca4415-il	200,000	5	68.2 at α=5.75°
☑	naca4415-il	500,000	9	97 at α=5.75°
☐	naca4415-il	500,000	5	90.4 at α=5.5°
☑	naca4415-il	1,000,000	9	119.4 at α=5.5°
☐	naca4415-il	1,000,000	5	109.1 at α=4.75°

Fig. 2.69. Legend for 4415 Graphs[31]

Cl and Cd are plotted versus alpha for each airfoil cross section listed above. Alpha in this case is the angle of attack of the airfoil. The angle of attack is the angle where the relative wind meets the airfoil. This angle is formed by the chord of the airfoil and the vector of the relative wind. An illustration of this can be seen below in Fig. 2.70.

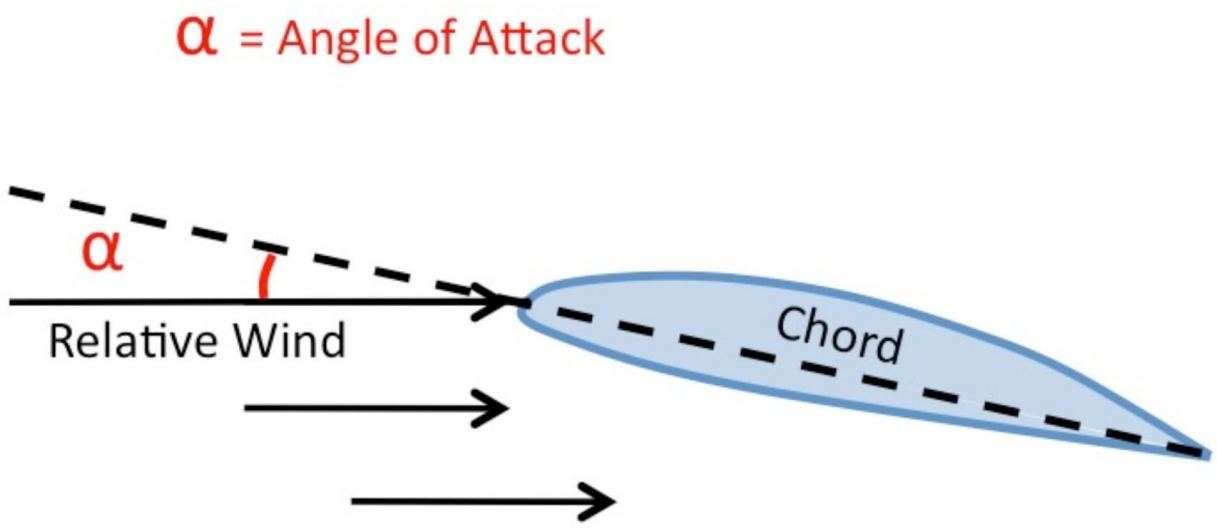


Fig. 2.70. Angle of Attack[32]

During a typical flight, once up in the air, the typical angle of attack for an airfoil is around 3 to 5 degrees. When flying at lower speeds this can safely go up to around 10 degrees. Using this information, a range of 0 to 10 degrees for alpha was used to compare the lift and drag coefficients on the airfoils. Another factor used in the comparison is the critical angle of attack, or otherwise known as stall angle. This is the angle that creates the largest lift coefficient. If the angle of attack is larger than the critical angle then the lift force begins to decrease, and the plane will eventually stall. The final comparison data between the airfoils can be seen below in Fig. 2.71.

Airfoil Comparison											
Airfoil	Angle of Attack (α, Degrees)										Critical α
	0		2.5		5		7.5		10		
	Cl	Cd	Cl	Cd	Cl	Cd	Cl	Cd	Cl	Cd	
NACA 1412	0.125	0.01	0.5	0.01	0.67	0.013	0.875	0.02	1.1	0.03	12.5
NACA 2412	0.35	0.01	0.6	0.012	0.83	0.0135	1	0.02	1.167	0.027	13.75
NACA 4412	0.5	0.01	0.75	0.013	1	0.014	1.27	0.0167	1.33	0.025	15
NACA 4415	0.5	0.013	0.83	0.014	1.07	0.015	1.22	0.016	1.3	0.023	15

	Cl/Cd			
α	NACA 1412	NACA 2412	NACA 4412	NACA 4415
0	12.5000	35.0000	50.0000	38.4615
2.5	50.0000	50.0000	57.6923	59.2857
5	51.5385	61.4815	71.4286	71.3333
7.5	43.7500	50.0000	76.0479	76.2500
10	36.6667	43.2222	53.2000	56.5217

Fig. 2.71. Airfoil Comparison Data[31]

Based on the data, it was decided to select the NACA 4412 as the profile of the wings for the final prototype. The 4412 profile was determined to have the best hypothetical efficiency based on the lift and drag coefficient data provided. The 4412 airfoil was also a very common profile for low-speed gliders and aircraft such as to be designed by the team.

After selecting the airfoil and the wing was designed, with help from the software team, a CFD analysis was completed on the wings. The CFD simulation was completed using the autodesk CFD software. A picture of the simulation along with the data can be seen in Fig. 2.72 and 2.73. Looking at the data, the Fx force is negligible, the Fy force is the lift force, and the Fz force is the drag force. Taking the Fy/Fz gives the lift/drag coefficient for the wing. The analysis was done for 30, 40, and 50 miles per hour.

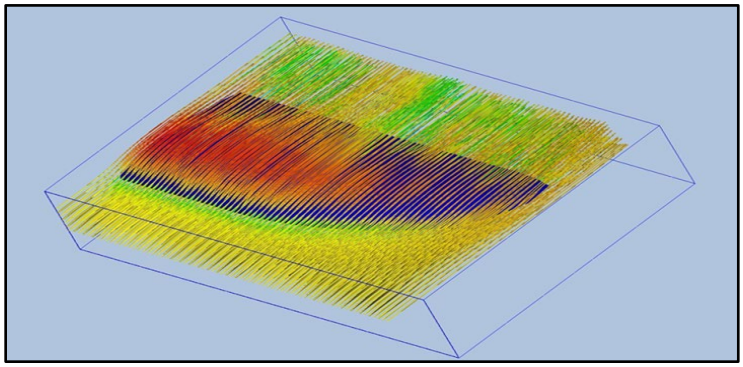


Fig. 2.72. CFD Simulation

Summary	30mph		40mph		50mph		Units

Total area	2948.98		2948.98		2948.98		cm^2
TOTAL FX	0.0179298		0.0317693		0.0471228		lbf
TOTAL FY	1.53459		2.73454		4.31351		lbf
TOTAL FZ	0.41403		0.721201		1.11777		lbf
Center of Force about X-Axis (Y-Z)	-2.51392	-4.88871	-2.52418	-4.52727	-2.60121	-5.39721	cm
Center of Force about Y-Axis (X-Z)	24.6609	16.3888	24.7079	16.3337	24.7418	16.3185	cm
Center of Force about Z-Axis (X-Y)	23.2391	-1.11584	23.2943	-1.11572	23.3132	-1.11519	cm
Lift/Drag	3.706470546		3.791647543		3.859031822		

Fig. 2.73. CFD Analysis Data

Final UAV Design

The initial UAV design was based off of some of the elements of the Mini Talon test plane. It was desirable to build a plane with similar sizes and proportions in order to make sure the motor would work with the final design as well. A similar size was also ideal due to the ability of the plane to be transported easily. The goal was to design and build a UAV that could be easily disassembled and transported to wherever it is needed, whether that was by car or in some type of pack being carried by hand. The original design sketches were drawn on paper to understand the ideal shape and size of the body that would reach further design iterations. Some of the initial sketches can be seen in Fig. 2.74.

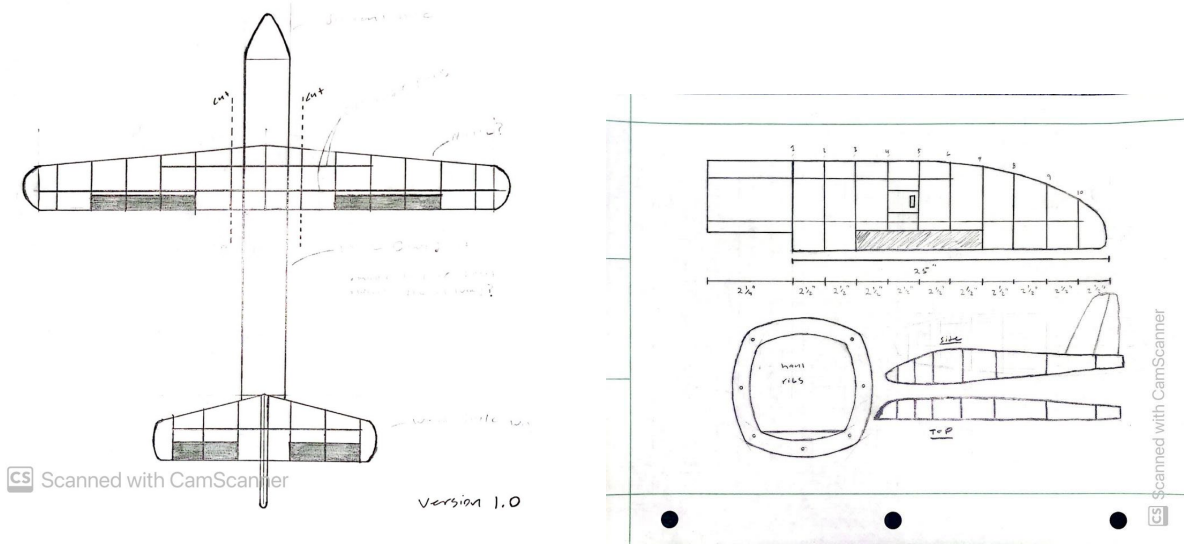


Fig. 2.74. Initial Design Sketches

The next step was drawing the rib profiles on Fusion 360. These ribs will be the main foundation of the wings and hull. As stated above, the ribs of the wings have the profile of the NACA 4412 airfoil. The wings taper down towards the outside edge of the plane to increase the efficiency by decreasing the drag force. The 3D CAD model of the wing and hull can be seen in Fig. 2.75.

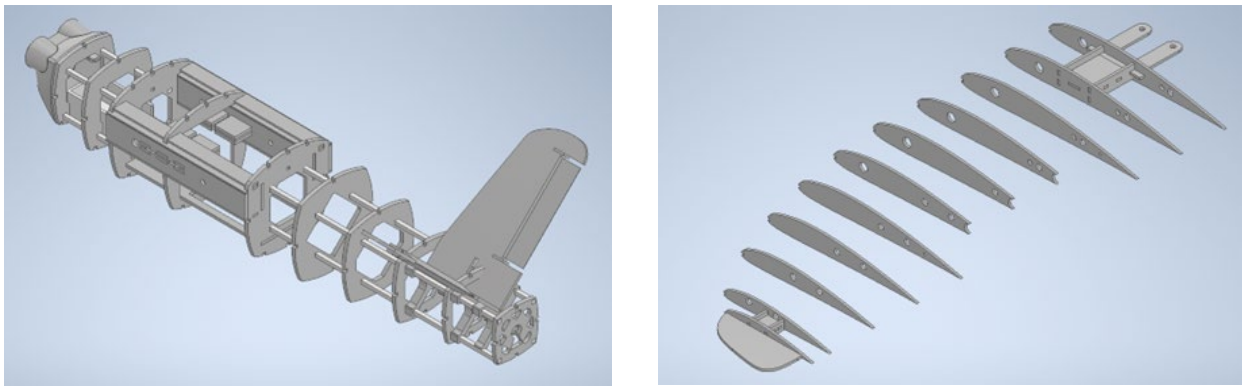


Fig. 2.75. Wing and Hull CAD Design

Once the CAD designs were finished, the process of cutting out the wing ribs and the hull ribs began. These were cut out of oak 1/8” laminate and 1/8” balsa wood depending on which parts needed to be more structurally sound. All of the wing and hull ribs were cut out using the 60-watt laser. The wing ribs were glued in place using super glue along predetermined spots on carbon fiber rods. The carbon fiber rods were used as the structure of the wings for stability and strength. The hull ribs were held together using oak square rods. The ribs were again glued to the rods in predetermined places to create the structure. The ailerons of the wings were also made using balsa wood ribs cut by the laser. The ribs were placed and glued along a metal rod which will act as a pivot for the ailerons. The ailerons were covered using a polystyrene sheet that was heated and wrapped around the ribs to provide a smooth and strong covering. The wings and hull can be seen in Fig. 2.76 along with the finished skeleton in Fig. 2.77.

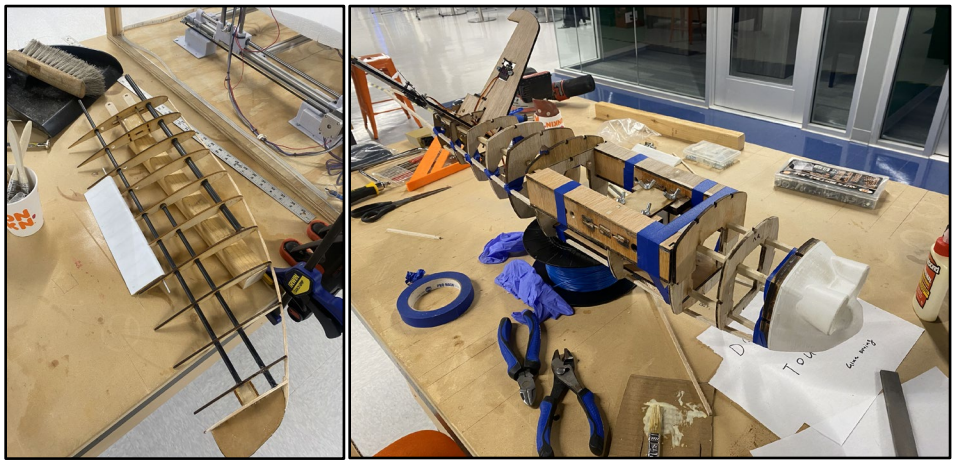


Fig. 2.76. Hull and Wing

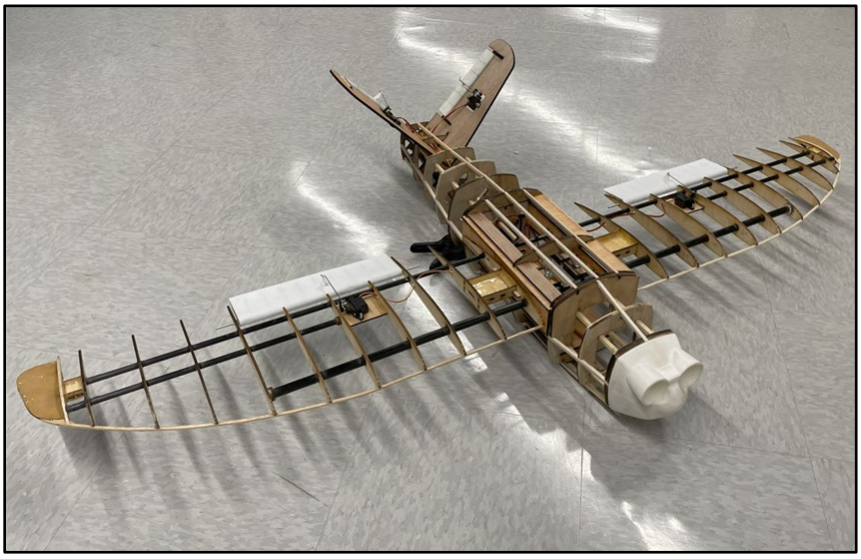


Fig. 2.77. Completed UAV Skeleton

The nosepiece of the hull was designed on Fusion 360 and 3D printed. There are two openings in the nosepiece to allow for air to flow through the inside of the hull in order to keep the electrical components and the battery cool during flights in the tropical climate. The wings were designed to be detachable when transporting the UAV. After completing the skeletal structure of the UAV, it was wrapped with foam sheets. Scoring one side of the sheet allows it to be bent around the outside of the frame. The completed UAV can be seen in Fig. 2.78.

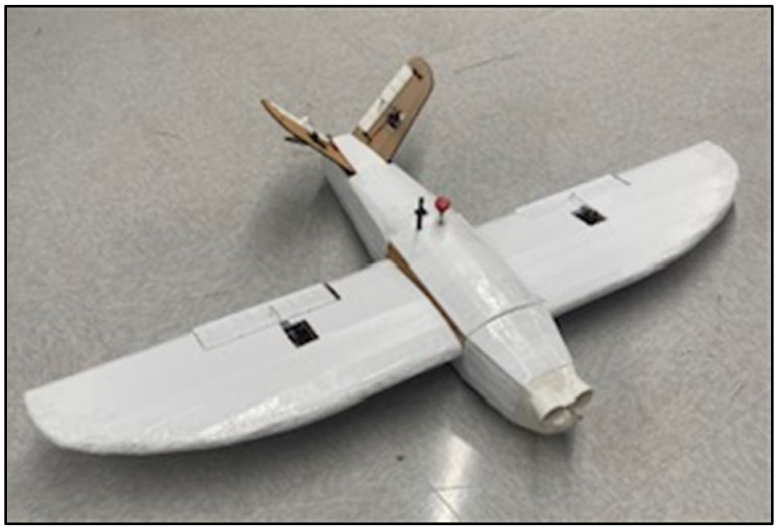


Fig. 2.78. Completed UAV

2.4 Software Team

Deliverables

Software team's deliverables were to create a system that collects photographical data from the terrain below the plane, stores that data on a storage media, and source software that processes all obtained data off-board of the drone. In order to achieve these goals, the software team worked backwards to find a solution.

Software Selection and Initial Testing

Soon after the start of the project, Open Drone Maps (ODM) was selected as the software suite to process the imagery data. It was chosen in October of 2022 due to its versatility and open-source code. As the client requested many different vegetation indexes (VI), it was decided that a software that can be programmed for any current or future index would be ideal. Further, ODM, unlike its paid competitors, has no monthly subscription cost to use the software. Instead, a paid Windows installer–included in the bill of materials (BOM)–is a one-time fee. Further, ODM has its Lightning Network that allows users to upload their imagery and pay servers to process the data remotely for a small fee. This allows for data processing when hardware requirements are too high for some jobs.

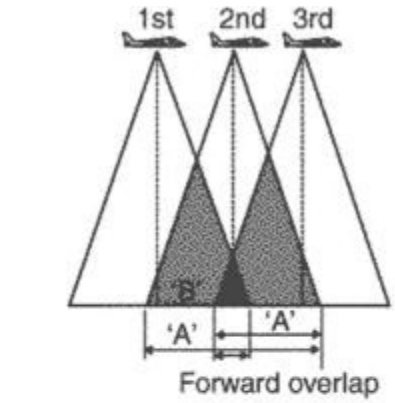
ODM requires the following to operate: a processing server and imagery data. For the server, the minimum amount of random-access memory (RAM) is proportional to the number of photos that must be stitched together. This requirement can be found in Table 2.7 below.

Table 2.7. Open Drone Maps Requirements[33]

Number of images	RAM or RAM + Swap
40	4
250	16

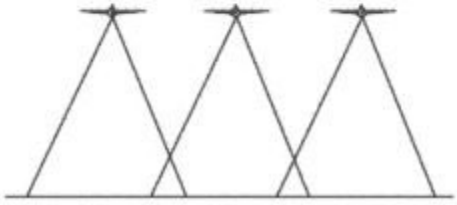
500	32
1500	64
2500	128
3500	192
5000	256

Essentially, ODM operates by stitching photographs together using interpolation algorithms. As such, there is an ideal amount of overlap between photos as it stitches them together. Based on the research done by the software team, this was found to be a 68% overlap and sidelap, making each photo share 83% of its area with its neighbors.[34] As the Imagery Team calculated in Table 2.7, the cameras are capable of taking a photo of about 21 acres at 200 feet.



Forward overlap

A. 60% overlap between successive prints
B. 20% overlap between alternate prints



Side lap between flight lines—normally 30%

Fig. 2.79. Diagram of aerial photography passes[35]

As such, the Software Team found that it would be necessary to derive a list of basic, governing equations for the project. These can be found in the Appendix F. These equations proved to be very important to the collective work between imager and software team to create an image-taking system that met the specifications given by the client and by the ODM software suite. Equations that were especially helpful include Appendix F: Equations 1-1 and 1-7 that can be used to find the pixel density at a given altitude based on the camera module specifications and find the necessary time between photos, respectively. This proved essential to ensure the photo-taking module would shoot a photo with the right distance between takes, allowing for proper image stitching later.

ODM can read metadata in the exchangeable image file format (EXIF). In order to get higher quality results, the following metadata is used: ODM was originally tested with some images donated by a local drone enthusiast for the project's research. These photos were stitched and created the following mosaic from over 200 images. Fig. 2.80 shows the mosaic in the visible light spectrum and Fig. 2.81 for the normalized difference vegetative index as described in section 2.2.

Fig. 2.80. Red Green Blue (RGB) Orthomosaic

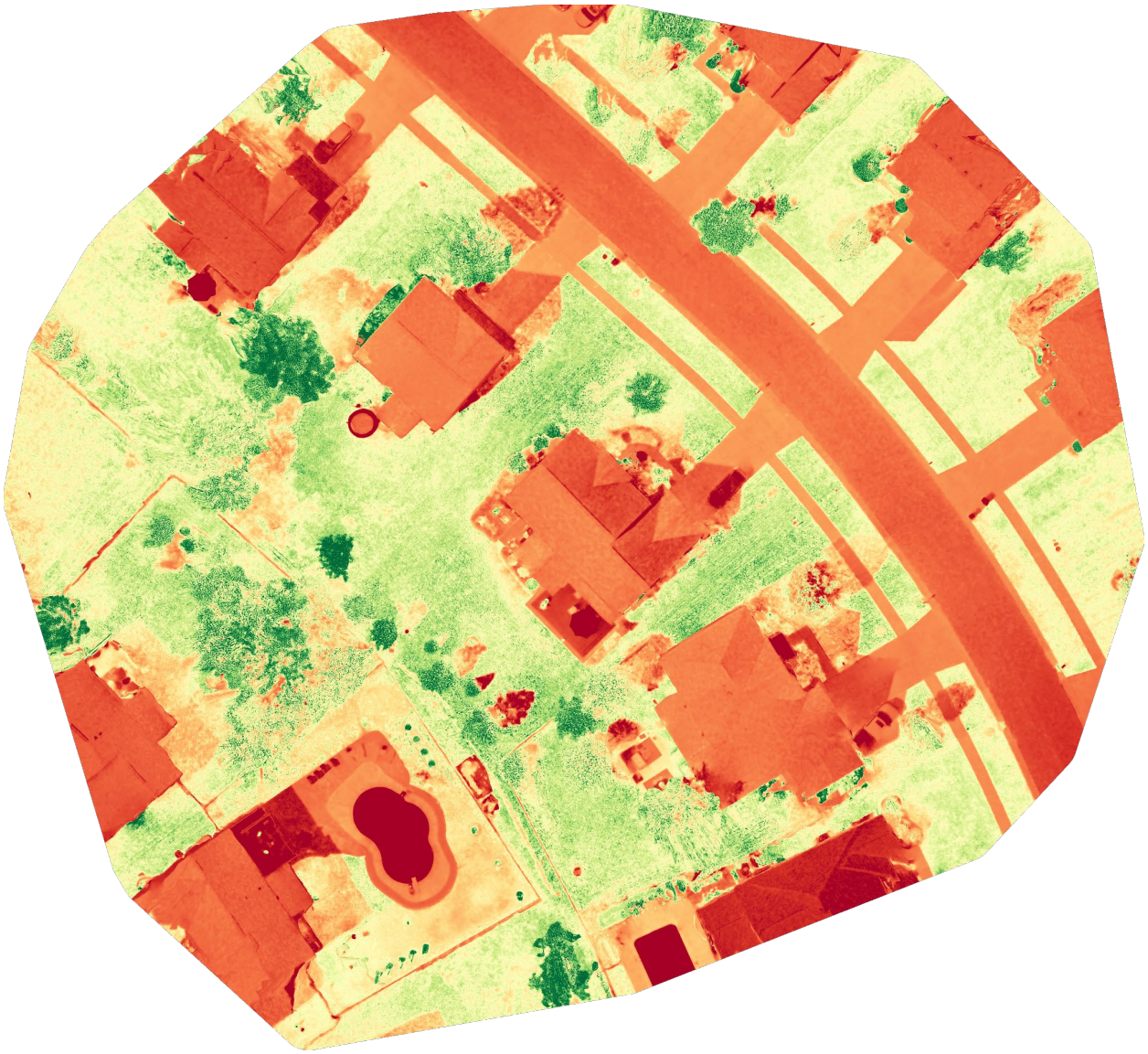


Fig. 2.81. Normalized Difference Vegetation Index (NDVI) Orthomosaic

ODM also self-generates a report of the image quality and accuracy of the interpretation. An example report of this can be found in Appendix H. ODM further can create very realistic 3D models if enough photos are taken. This can be seen in Fig. 2.82 and Fig. 2.83 below.

Fig. 2.82. 3D render of data

Fig. 2.83. Closeup of 3D render of data

Remote Technical Support and Updating

In order to provide technical support to the client remotely, the following was used: a virtual private network (VPN), secure socket layer (SSL) tunnels, and remote

desktop protocol (RDP). VPNs allow for the secure usage of a local area network (LAN) remotely. This allows for one to tunnel their traffic through an encrypted exchange to a server and act like the client is in the same LAN as the server. For the case-use in this project, the computer that was configured for the drone client contains a VPN configuration that sends its traffic to a server managed by the university. This means that the devices will show up on the LAN at the university as a local network device. Thus, a student would be able to login to the device remotely with proper credentials, essentially creating a reverse VPN tunnel. In order to control the configured device, the preconfigured laptop was set to enable RDP serving within Windows 11. This allows a student to provide technical help to the client remotely. Further, in order to be able to apply updates to the camera-taking module, a Cloudflare-based SSL tunnel was created that allows for the use of the secure-shell protocol (SSH) to remotely update and serve files to the photo-taking module's SBC. If for some reason, the system image becomes corrupted on the SBC, the Secure Digital (SD) card is accessible from the top of the drone, next to the universal serial bus (USB) drive that stores the photo data. All the client has to do to enable the remote features for the SBC is to create a hotspot or wireless access point with a username and password that is known by the client of the project, and the SBC will automatically connect if it has power, allowing for remote control. For more information regarding these protocols, see Appendix G. As can be seen in the report, camera module correction factors can be found. This correction is essential because of the wide-angle camera used in this UAV, as distortion from the bubble lens at the edges of its view can cause systematic deviation in the images.

GPS Testing

The project's client was unsure of the global position system (GPS) satellite fixing in their region of interest. Due to this, the software team readily designed, prototyped, and tested a GPS tracker that logs the quality of the signal. Below is Fig. 2.84, a photo of the completed peg-board prototype that was sent to the client's region of interest.

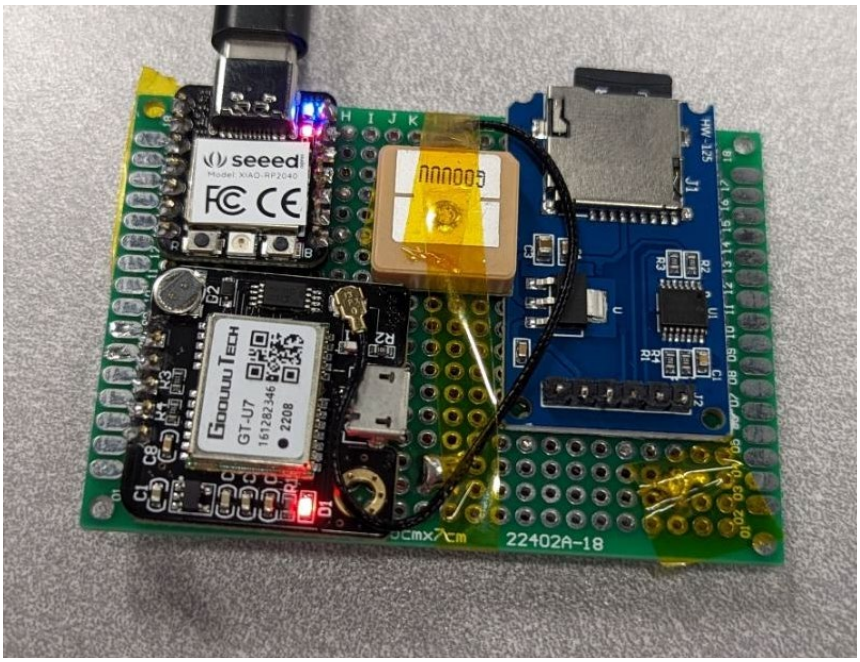


Fig. 2.84. Photo of the Completed Protoboard

The prototype uses a Seeed microcontroller with a Raspberry Pi Pico processor, a microSD card reader, and a GPS module. Power is received from a standard external battery cable and interfaces with the microcontroller's built-in USB-C port. The GPS communicates to the microcontroller via universal asynchronous receiver-transmitter (UART) protocol, and the SD card module utilizes a serial-parallel interface (SPI). The breadboard diagram and micropython code for the unit can be found in Appendix I. Appendix J shows renders of the completed case and Appendix K shows some of the collected data. The client was given a set of instructions to use the device, this can be found in Appendix L. As can be seen from the raw GPS data, the device obtained a fix from 14 different satellites. This is more than enough for very accurate GPS data. The data was mapped using a keyhole markup language file format and imported into Google Earth. The result of this is shown in Fig. 2.85 below.

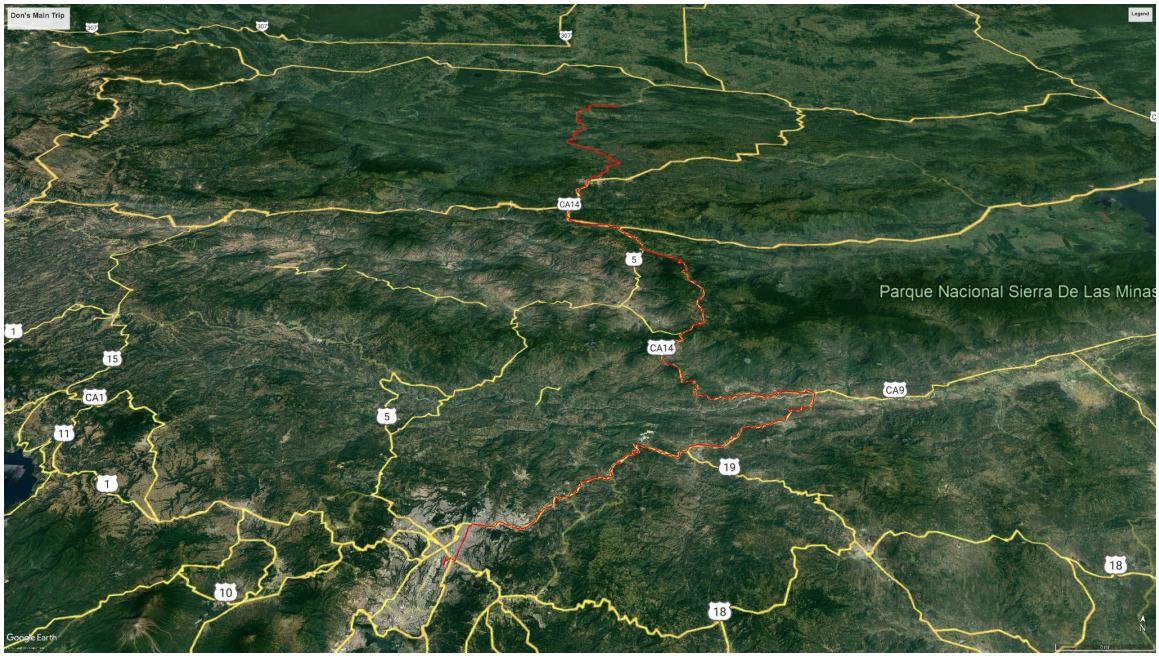


Fig. 2.85. Mapped journey from GPS data

As can be seen in the figure, the data collection has no issues with being accurate for the latitude and longitude. However, after extensive data analysis, it was found that while the coordinate data was accurate to within 10 feet, the altitude data could vary between less than 10 and over 100 feet. Thus, it was concluded that GPS data could not be used to reliably determine absolute altitude.

Single Board Computer and Camera Hardware Selection

To run the camera modules chosen for this project, the Raspberry Pi Camera V3, a device with two camera serial interfaces (CSIs) would be needed. The original choice was a Raspberry Pi (RPI) compute module 4 (CM4). Due to ongoing supply-chain issues, it was deemed impractical and expensive to obtain the RPI CMs. Instead, an alternative board was found: the SoQuartz CM. It exceeded the performance of the RPI on paper and had two CSI lanes, one for each camera. However, even installing an operating system on the single-board computer (SBC) proved to be surprisingly difficult, as the device is not well documented. In the end, a modified version of Debian 13 was found to run on the SBC but the board had trouble with the software drivers needed to run the 64 megapixel (MP) Arducam modules planned for at the time. In the end, the use of the SoQuartz board

was abandoned. Instead, a member of the software team and a member of the supervising faculty donated their RPI 4 and 3, respectively, for their permanent incorporation in this project.

The Raspberry Pi foundation does a very good job at documenting and standardizing their ARM-based SBC. The pinout between their many models of Pi's allow for programs to run on any of their Pi's. As such, any programs written for the RPI3 can run on the RPI4 or even CM. This made the software team's job much easier, as only one program would have to be written for all the expected devices. (As the project would ideally use CMs in the future.) Due to the RPI3 and 4 only having one CSI, the use of a multiplexer was necessary. Thus, the Arducam 4-channel multiplexer was selected, purchased, and interfaced. It operates by using an I2C interface and general purpose input-output (GPIO) pins to control which camera module is connected to the Pi's CSI. An image of the Arducam can be seen below in Fig. 2.86.

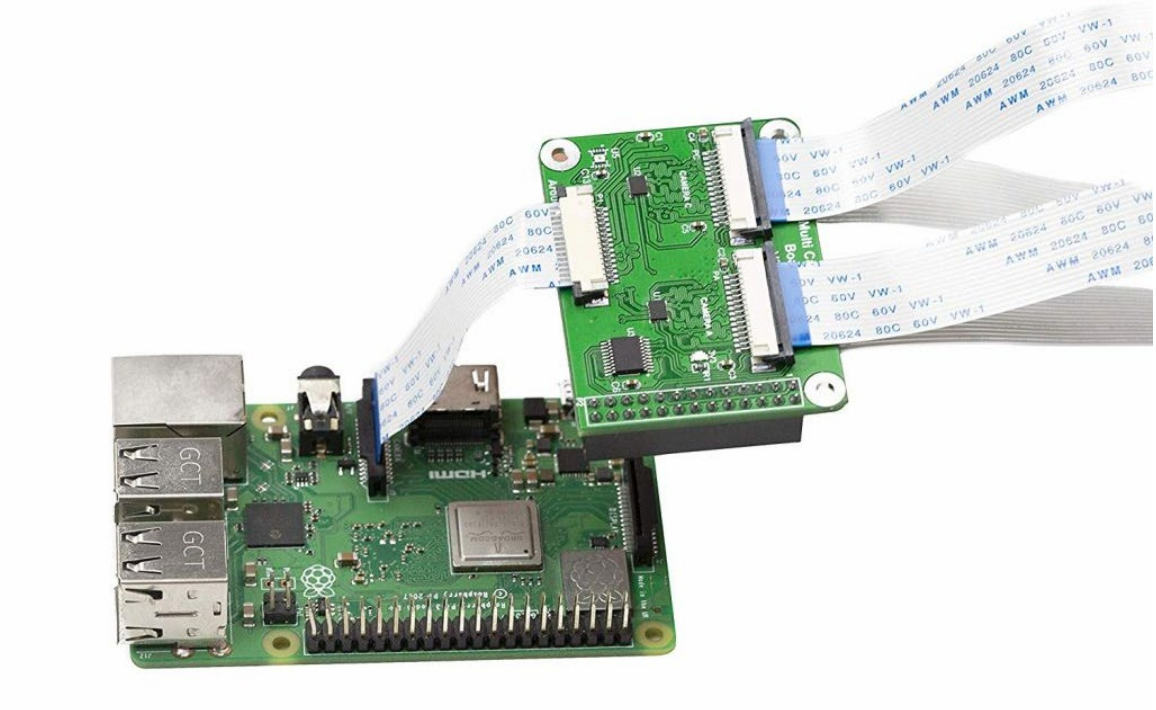


Fig. 2.86. Photo of Arducam 4-channel camera multiplexer[36]

The multiplexer has five CSIs: one to connect to the Pi upstream and four for each camera, labeled 'A', 'B', 'C', and 'D'. In order to select a camera, Table 2.8 below

shows the combination of the I2C interface that is selected and the state of the GPIO pins used. For this implementation of the multiplexer, cameras ‘A’ and ‘C’ were used.

Table 2.8. Pin state for camera module selection

	GPIO Pin Number					
Camera	7	11	12	I2C Address		
A	0	0	1	0x70	0x00	0x04
B	1	0	1	0x70	0x00	0x05
C	0	1	0	0x70	0x00	0x06
D	1	1	0	0x70	0x00	0x07

The pinout of the standardized 40-pin RPI layout can be seen below in Fig. 2.87.

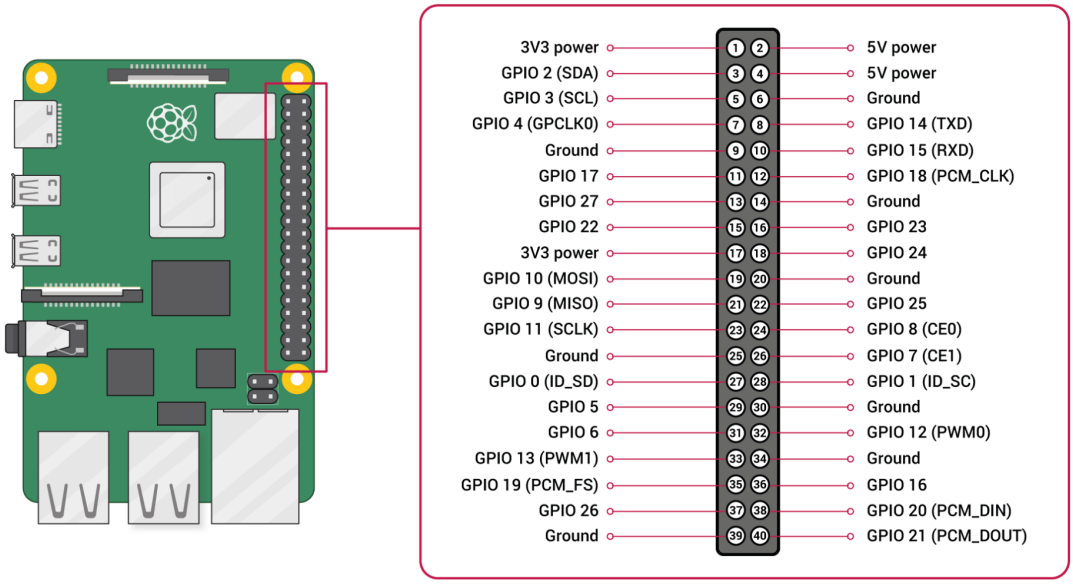


Fig. 2.87. Diagram of 40-pin Raspberry Pi Pinout[37]

Software Overview and Hardware Interfacing

In order to get higher quality stitching from ODM that uses less resources, geotagging is necessary. By adding the latitude and longitude to the metadata, in EXIF, each photo can be directly compared to one another based on geographical location. In order to add this tag, GPS data must be read, interpreted, and appended to the image metadata. This is done using the `gpsd` linux daemon on the Pi. Using `cgps`, a graphical interpreter of the daemon, one can easily read out the data from `gpsd`. An example of this data can be seen in Fig. 2.88 below.

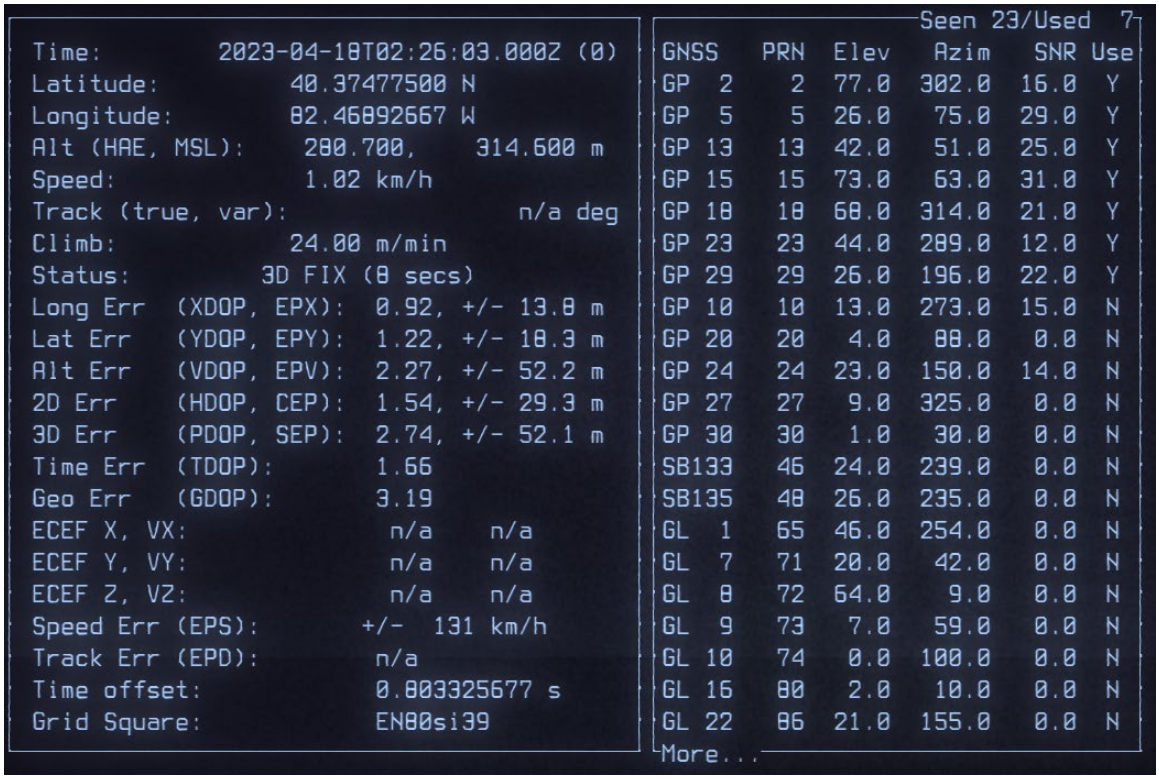


Fig. 2.88. Cgps Library Testing on Raspberry Pi

The GPS location from the test above was searched and found on Google Maps. The location can be seen in Fig. 2.89 below. This location was within ten feet of the antenna during the test.

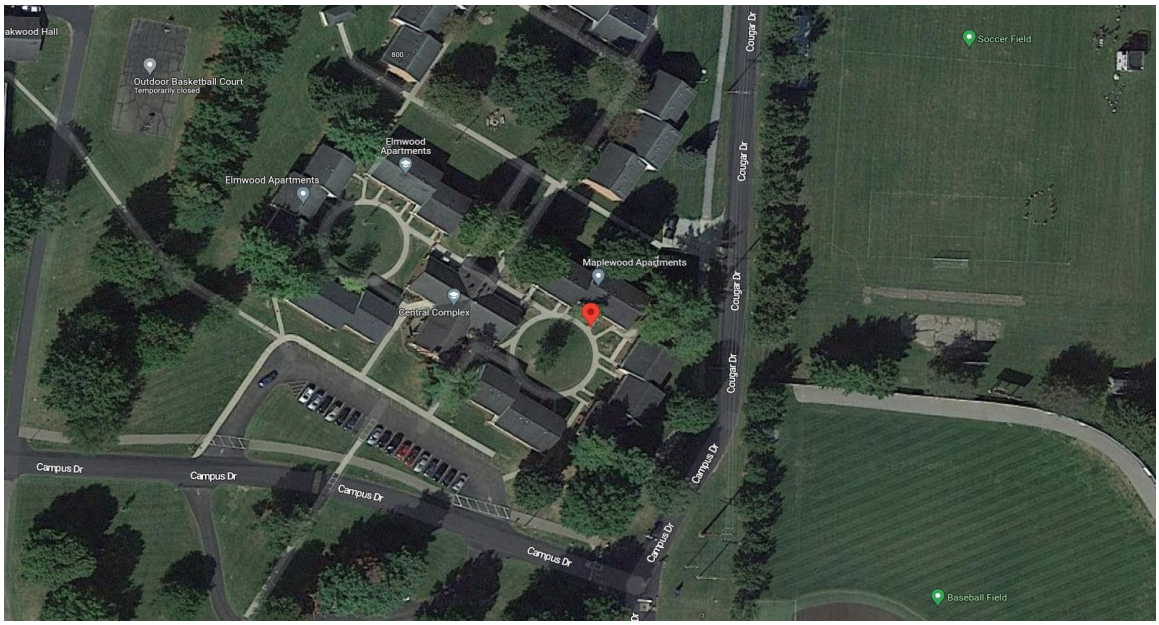


Fig. 2.89. Location shown by GPS data in test

The GPS module used in this project is the Matek M8Q-5883 as imaged in Fig. 2.90 below. This module has two identical UART and I2C interfaces. This means that the flight controller and the Pi can both update their GPS location in real-time. For this module, the I2C is used to update the magnetic compass heading, and the UART pins. Thus, the Pi only receives data from the UART interface, as the heading is not necessary for the EXIF tag.

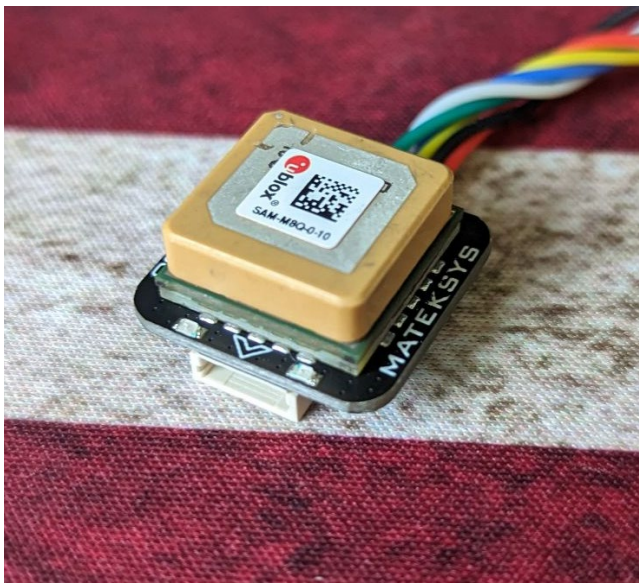


Fig. 2.90. Image of the GPS module

Unfortunately, the RPI 3 and 4 only have one UART or serial interface. Thus, an adapter is needed to translate the UART data to USB to take advantage of the Pi's four USB ports. The wiring for this can be seen in Fig. 2.91 below.

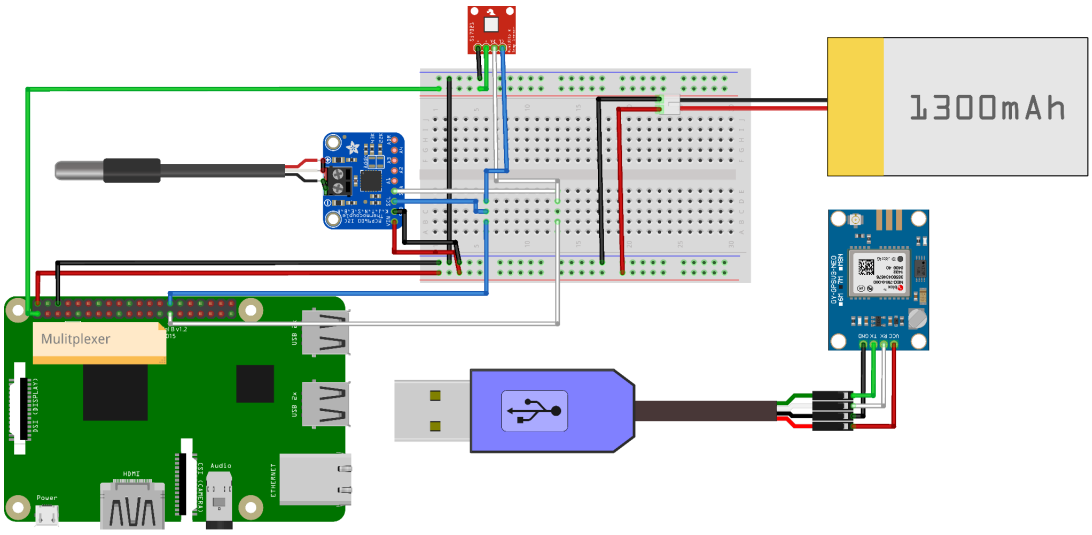


Fig. 2.91. Breadboard wiring for the photo system

Due to the multiplexer covering pins 1-26, jumper wires were soldered to the multiplexer to allow for 3.3V, 5V, and ground access for the other components. Also pictured in Fig. 2.91 above is the array of sensors used to monitor the battery temperature (the MCP9600 module) and the ambient temperature and humidity (SI7021). These sensors were added to the SBC due to the failure of the custom PCB project undergone by the electrical team. These modules are used to log temperature and humidity data to be able to find trends regarding temperature and battery capacity. Pictured in the diagram is a battery that represents the 5V output of the flight controller. The 5V pin of the RPI is bidirectional, allowing for the Pi to power other components or itself through pins 2 and 4. The USB attached to the GPS module supplies both power and data transfer. Overall the power consumption of the system can theoretically reach up to 15 watts under full load, limited by the flight controller's current output.

The program that runs the cameras is based on the `picamera2` and `libcamera` libraries for python. These libraries allow for the capturing of video and imagery from CSI cameras. This includes resolution, frame rate, exposure, etc. The program runs as an enabled, persistent service that runs at boot, meaning that the program is running whenever the Pi receives power. However, photos are not taken until a high signal is received on pin 36 (GPIO 16). When the high signal is received, photos are taken, switching alternating between camera A and C (RGB and NIR). The time delay between photos is determined by finding the distance between photos. This distance is calculated from Eqs. 1-1 and 2-2 in Appendix F. As it would be ideal to know the relative altitude above the ground, the program has a parameter that allows for the altering of the relative altitude. By default, this is 200 feet. Thus, for the Raspberry Pi Camera Module V3 NoIR Wide Angle with a resolution of 4608x2592 and a field of view of 120 degrees, the equation to find the distance between photos can be found below in Eq. 2.7 below, where d is the relative altitude and x is the overlap factor between subsequent photos.

$$\text{Photo Distance} = 1.6983d(1 - x) \qquad \text{(Eq. 2.7)}$$

Thus, by using 200 feet for the altitude and 68% for the overlap factor. Thus, the camera module will need to take a photo with both cameras every 108.7 feet traverse in the air. For the program, the capture method is run once the GPS location is greater than 108.7 feet from the location of its previous photo. This ensures that the data has the correct overlap to allow ODM to run accurately and efficiently. This python program can be seen in Appendix L. Based on testing, the code is able to take photos fast enough if the UAV cruises at a speed under 40 miles-per-hour. Photos taken from this program can be seen in Fig. 2.92 below. Note: these photos were taken while moving. Images were not taken from exactly the same perspective.

Fig. 2.92. RGB and IR photos

As can be seen from the photos above, a higher content in chlorophyll results in a higher reflectivity of infrared light coupled with a higher reflectivity of the green spectrum of visible light.

Software Conclusion

The software is designed for a rolling-release system, as parameters such as exposure and focal distance will likely need to be tuned according to the client's intentions in the region of interest. As such, the Pi has the ability to be reprogrammed remotely as covered previously. It is planned that a member of the team will provide continued technical support for the project to the client for at least six months. Further, as two identical drones were created for the client and for the university, any upgrades and

configurations changed on the drone at the university can be implemented by the client and programmed remotely. This ensures the repeatability and modularity of the UAV. ODM is a very robust software with many available plugins. As the client explores the capabilities of the software, it is expected that support will be needed to use ODM properly as well. In all, the software team has designed, programmed, and implemented a camera taking module that is mostly independent of the rest of the drone.

As can be seen from the previous subsections, all of the software team's deliverables were met as a system that collects photographical data from the terrain below the plane, stores that data on a storage media, and source software that processes all obtained data off-board of the drone was created, tested, and implemented.

3 UAV Iterations

3.1 The Goose (Prototype)

The prototype, nicknamed the Goose, was created to test the inner components of the UAV. The team had decided to buy a prebuilt body in order to see if the components ordered worked together. The tested components included the flight controller, GPS, motor, propellers, RC controller, receivers, battery, antennas, and telemetry system. Thus, all of the components were put into the X-UAV Mini Talon body, the prebuilt body, and were wired accordingly. The result of this is pictured below.

Fig. 3.1. The Goose

On March 15th, the Goose was taken for a test flight and was successful in flying. However, one of the problems noticed was that the ailerons did not bend as much as expected. This caused the Goose to take wide turns as well as not being able to ascend or descend as quickly as it should have. Fortunately, the flight of the Goose proved that the inner components used for flight worked together. Thus, the only things left to test were the custom body of the UAV, the FPV system, and the imagery team's work.

Unfortunately, after about fifteen minutes of flying the Goose, the fuselage cover came loose and flew off. This caused wind to get into the inner components of the Goose and move the wires enough to disconnect one. Due to this, Aaron Aude, who was flying the Goose, lost control and it ended up crashing. After the Goose had been recovered, it was found that the X-UAV Mini Talon body was broken, but nothing else was harmed. This was good news as this body was not the final design: only the internal components would be transferred to the final hull.

3.2 The Mule (Custom Body)

After the test flight of the Goose, the Design and Manufacturing team finished the design of the customized wings and hull. These were made of balsa wood with foam over top of them. The design can be seen below. The design was transferred to the laser cutting machine that the team had access to. Each of the parts were made and then glued together to make the skeleton of the Mule, the nickname of the team's first UAV. This UAV used the NACA 4412 airfoil and was inspired by the design of the X-UAV Mini Talon.

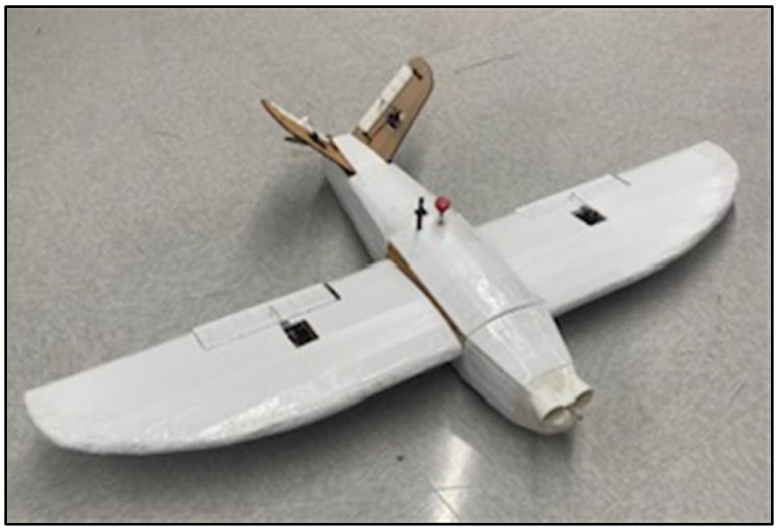


Fig. 3.2. The Mule

Unfortunately, the Mule ended up being too heavy to fly. This is due to a combination of two things. One, after the flight test of the Goose, which ended up destroying the body, the team wanted the customized hull to be sturdy. This would allow the UAV to last longer as it could be fixed much easier if it were to crash. However, the tradeoff for making it sturdy was that it would add weight. Secondly, UAVs must have their center of gravity in about the middle of the wings. At first, the Mule was back-heavy. This caused the team to add weight to the front to balance it out. The team ended up adding about 700 grams to the nose. This caused the UAV to weigh approximately 3000 grams. The motor and propeller combination used was rated to move about 1500 grams, but the motor thrust test revealed it could carry about 1800 grams. Even without

the weights added, the sturdiness of the UAV caused it to be at least 500 grams too heavy.

There ended up being two options to get the Mule to fly. One would be to find a different motor and propeller combination with more thrust output. The other option would be to redesign the wings and hull to make it less sturdy, but lighter. The problem with the first solution is that it would require a much larger propeller. Increasing the propeller size would inevitably lead to the propeller breaking during landing, not to mention the danger of a longer propeller producing enough thrust to carry 3000 grams. The other solution sounds like a better idea. However, this has its own problems. The biggest problem is time. Frankly, it takes too much time to completely redesign the UAV. Even after that is done, the UAV still has to be built again. The fastest turnaround for this would likely be about two months. Unfortunately, this means that the Mule would never be able to fly.

3.3 The Gander (UAV 1)

The team still needed to produce a couple of UAVs, which meant they would have to pursue a different solution. After the team determined that the customized UAV would be unable to fly, they set to work with recreating the Goose. This UAV, nicknamed the Gander, would use another X-UAV Mini Talon body with the chosen components inside of it. However, the difference between the Goose and the Gander is that the Gander would hold more components. The Goose only held what was necessary to fly it. The Gander had to hold the cameras, Raspberry Pi, airspeed sensor, as well as everything else the Goose had.

This solution ended up being easy to implement since the team already knew the body was able to fly. The team also knew that the imagery system works and is able to stitch photos together, as well as run NDVI calculations. Thus, the only thing left to do was put everything into the Gander and test to see if the imagery system worked in flight.

Fig. 3.3. The Gander

3.4 The Third Bird (UAV 2)

After the Gander was built, the team had to build one more UAV. This UAV, named the Third Bird, would be identical to the Gander. Since one of the requirements of the project is manufacturability, the team had to prove that the product could be repeatedly made and manufactured. As of the time of writing this report, this is still a work in progress. The goal is to finish this by May 13th, or prior to going to Guatemala.

4 Conclusions and Future Work

The main goal of this project was to "design and test a UAV used to determine the health of plants in Guatemala." Although the customized UAV body was unable to fly, the team was able to deliver two UAVs capable of flight and ascertaining the health of crops. The team was even able to put together a portable weather station in order to determine whether flight can be achieved on any given day based upon the conditions.

The overall BOM can be seen in the Appendices section. This ended up costing $3,081.90. However, this cost includes the cost of both UAVs as well as research and development costs. Going more into detail, making the prototype Goose and even the materials of the Mule pushed this price upwards. It was also expected that the original cameras would be able to do what was wanted, but that ended up not being the case. Also, some components had to be bought multiple times due to them breaking. These are just a few examples of things that made the project more expensive. However, the BOM and estimated cost for making one UAV can be seen below. This shows a cost of $1,186.50. This much lower price is what the team would expect to pay if they had to make one more UAV. Other UAVs tend to cost around $1000, so this falls close to that.

Table 4.1. BOM and Cost of Making a Single UAV

Item	Price per Unit	Quantity	Sub-Total
X-UAV Mini Talon	$ 69.99	1	$ 69.99
Foam Glue	$ 11.50	1	$ 11.50
Cobra C-2814/16 Brushless Motor	$ 37.99	1	$ 37.99
Cobra 33A ESC with 3A Switching BEC	$ 29.99	1	$ 29.99
Uxcell RC Propellers CW 9x4.5 Inch 2-Vane	$ 13.49	1	$ 13.49
Seamuing 6Pcs MG90S Micro	$ 19.99	1	$ 19.99

Item	Price per Unit	Quantity	Sub-Total
Servo			
Ltvystore 10Pcs Adjustable Pushrod Connector	$ 11.99	1	$ 11.99
4 Pcs LED Aircraft Strobe Lights	$ 12.99	1	$ 12.99
Matek H743-Wing V3 Flight Controller	$ 109.99	1	$ 109.99
SiK Telemetry Radio V3	$ 58.99	1	$ 58.99
Emax Pagoda 3B 5.8Ghz 50mm Antenna	$ 5.99	2	$ 11.98
Lumenier SM-25 25mW Micro VTX	$ 11.99	1	$ 11.99
Caddx Ant FPV Camera	$ 20.99	1	$ 20.99
Matek Digital Airspeed Sensor ASPD-4525	$ 47.99	1	$ 47.99
Spektrum SRXL2 DSMX Serial Micro Receiver	$ 31.99	1	$ 31.99
MATEKSYS M8Q-5883 GPS Module	$ 35.99	1	$ 35.99
Turnigy 5000mAh 4S 25C Lipo Pack	$ 41.97	1	$ 41.97
LiPo Charger Lipo Battery Balance Charger RC	$ 56.99	1	$ 56.99
FPV Monitor 5.8GHz	$ 106.59	1	$ 106.59

Item	Price per Unit	Quantity	Sub-Total
Realacc Triple Feed Patch-1 5.8GHz Antenna	$ 16.99	1	$ 16.99
RC Controller	$ 114.99	1	$ 114.99
Raspberry Pi 3*	$ 35.00	1	$ 35.00
Raspberry Pi Camera Module 3	$ 34.99	1	$ 34.99
Arducam Multi Camera Adapter Module V2.2	$ 49.99	1	$ 49.99
Green Infrared Filter	$ 19.99	1	$ 19.99
OPTOLONG 1.25" UV/IR Cut Filter	$ 44.00	1	$ 44.00
iSOUL [4 Pack] Screen Protector	$ 4.99	1	$ 4.99
Wisesorb Silica Gel Packets	$ 8.49	1	$ 8.49
SanDisk 256GB Ultra Fit USB 3.1	$ 19.99	1	$ 19.99
20 Pairs Mini Micro 6 Pin JST SH 1.0mm Cable	$ 9.49	1	$ 9.49
DHT Electronics 2PCS coaxial Coax Adapter	$ 5.80	1	$ 5.80
FLY RC 2 Pack XT90 Charging Cable XT90 to 4.0mm Banana Connector	$ 8.99	1	$ 8.99
XT90 Connector Male Female	$ 10.99	1	$ 10.99

Item	Price per Unit	Quantity	Sub-Total
Adapter for Battery ESC			
DTTRA 20 Pairs 20 AWG JST Plug Connector 2 Pin	$ 4.99	1	$ 4.99
yueton Rc 1-8s Lipo Battery Tester	$ 5.49	1	$ 5.49
BTF-LIGHTING 20 Pairs JST SM 3 Pin Connectors	$ 9.99	1	$ 9.99
AINOPE USB Extension Cable 1.5FT	$ 4.99	1	$ 4.99
3pin FPV silicone cable for RunCam	$ 2.99	1	$ 2.99
Atnsinc 3Pcs CP2102 USB 2.0 to TTL 5Pin	$ 9.99	1	$ 9.99
Dorhea 2PCS 24.4inch Micro SD to SD Card Extension	$ 10.99	1	$ 10.99
Micro Center 32GB Class 10 Micro SDHC Flash Memory Card	$ 8.99	1	$ 8.99
Total		42	$ 1,186.50

For future work on this project, a flying, customized UAV and a working custom PCB is desired.The team believes they would have been able to accomplish both of these tasks if given more time. The process of making the customized UAV fly would have involved a complete redesign and build. This would have likely taken another two months. The team determined the problem with the customized PCB as well, which

would have required redesigning it and having it fabricated again. This could have been done in one month.

It appears that the reason the team ran out of time had a few factors involved. One is that the team is inexperienced. The team has never worked on a project of this scale, and so was not able to properly schedule throughout the entire duration. Another factor came with supply chain issues. There were multiple components that arrived much later than expected, which pushed back time for testing and troubleshooting. Finally, the team should have had smaller, more frequent meetings to check the progress of each team. This would have allowed for each team to know exactly what to focus on at any given point and the expected time frame for it.

Overall, the team is pleased with their work on all of the UAVs. At the end of the day, the team was able to accomplish its original goal. Although not everything worked as or when expected, the team was able to learn what it is like to be full-fledged engineers and solve problems.

References

[1] Sr_Design_01Sept2022_Preview. Presentation.

[2] Fleddermann, C. B. (2014). Engineering ethics. Pearson Education.

[3] *Flight Controller H743-wing V2 & V3*. Matek Systems. (n.d.). Retrieved May 1, 2023, from http://www.mateksys.com/?portfolio=h743-wing-v2

[4] *STM32CubeProg*. STMicroelectronics. (n.d.). Retrieved May 1, 2023, from https://www.st.com/en/development-tools/stm32cubeprog.html

[5] V. Monebhurrun, "Revision of IEEE Standard 145-2013: IEEE Standard for Definitions of Terms for Antennas [Stand on Standards]," in *IEEE Antennas and Propagation Magazine*, vol. 62, no. 3, pp. 117-117, June 2020, doi: 10.1109/MAP.2020.2983956.

[6] Benjie. (2022, October 10). *Wireless Antenna Characteristics explained*. Study CCNP. Retrieved May 1, 2023, from https://study-ccnp.com/wireless-antenna-characteristics-explained/

[7] Adminjem. (2023, April 5). *Intro to antenna polarization - JEM engineering blog*. JEM Engineering. Retrieved May 1, 2023, from https://jemengineering.com/blog-intro-to-antenna-polarization/

[8] FCC Online Table of Frequency Allocations, 7 C.F.R. § 2.106 (July 1, 2022).

[9] *GPS & Compass Module M8Q-5883*. Matek Systems. (n.d.). Retrieved May 1, 2023, from http://www.mateksys.com/?portfolio=m8q-5883

[10] *DXS transmitter only*. Spektrum. (n.d.). Retrieved May 1, 2023, from https://www.spektrumrc.com/product/dxs-transmitter-only/SPMR1010.html

[11] *SRXL2 DSMX Serial Micro receiver*. Spektrum SRXL2 DSMX Serial Micro Receiver | Horizon Hobby. (n.d.). Retrieved May 1, 2023, from https://www.horizonhobby.com/product/srxl2-dsmx-serial-micro-receiver/SPM4650.html

[12] *Amazon.com : Caddx Ant FPV camera 1200TVL global WDR OSD 1.8mm ultra ...* (n.d.). Retrieved May 1, 2023, from https://www.amazon.com/Caddx-Camera-1200TVL-Global-Aspect/dp/B088TDTL1M

[13] *Lumenier SM-25 25MW Micro VTX - U.FL SMA*. www.getfpv.com. (n.d.). Retrieved May 1, 2023, from https://www.getfpv.com/lumenier-sm-25-25mw-micro-vtx-u-fl-sma.html

[14] *Emax pagoda 3B 5.8ghz 50mm antenna (SMA male)*. HeliPal.com. (n.d.). Retrieved May 1, 2023, from https://www.helipal.com/products/emax-pagoda-3b-5-8ghz-50mm-antenna-sma-male?variant=21503497338940¤cy=USD&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic&gclid=CjwKCAjwo7iiBhAEEiwAsIxQEVs-3JXKKvtXAFalMbebZjMSRVg2JJlAi8d3Y783DN5vyQ9HMHaN1RoChh0QAvD_BwE

[15] Banggood.com. (n.d.). *Realacc triple feed patch-1 5.8ghz 9.4DBI directional circular polarized FPV pagoda antenna for Fatshark Dji eachine goggles*. www.banggood.com. Retrieved May 1, 2023, from https://usa.banggood.com/Realacc-Triple-Feed-Patch-1-5_8GHz-9_4dBi-Directional-Circular-Polarized-FPV-Pagoda-Antenna-for-Fatshark-DJI-Eachine-Goggles-p-1195261.html

[16] *Amazon.com: FPV Monitor 5.8GHz 40channels 7inch LCD screen monitor ...* (n.d.). Retrieved May 1, 2023, from https://www.amazon.com/40Channels-Receiver-Quadcopter-Automatic-Switching/dp/B07NMJ2ZV5

[17] *Sik Telemetry Radio V3*. Holybro. (n.d.). Retrieved May 1, 2023, from https://holybro.com/products/sik-telemetry-radio-v3

[18] *Battery Life Calculator | DigiKey Electronics*. (2023). Digikey.com. https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-battery-life

[19] *Turnigy 5000mah 4s 25c Lipo Pack W/XT-90*. Hobbyking. (n.d.). Retrieved April 28, 2023, from https://hobbyking.com/en_us/turnigy-battery-5000mah-4s-25c-lipo-pack-xt-90.html

[20] Team, M. (2019, August 2). *6 Important Parameters for the Design-In of Lithium Polymer Batteries – Jauch Blog-Seite*. Jauch Blog-Seite. https://www.jauch.com/blog/en/6-important-parameters-for-the-design-in-of-lithium-polymer-batteries/#:~:text=By%20default%2C%20lithium%20polymer%20cells,prevail%20when%20charging%20the%20cells

[21] #847080, M., Smerk, Squirrel, R., #439268, M., #30085, M., #1070593, M., #91888, M., #873709, M., #57306, M., #1493453, M., Richmund, & Ewf. (n.d.). *SparkFun humidity and temperature sensor breakout - SI7021*. SEN-13763 - SparkFun Electronics. Retrieved April 28, 2023, from https://www.sparkfun.com/products/13763

[22] Industries, A. (n.d.). *Adafruit MCP9600 I2C thermocouple amplifier*. adafruit industries blog RSS. Retrieved April 28, 2023, from https://www.adafruit.com/product/4101

[23] Industries, A. (n.d.). *Thermocouple amplifier MAX31855 Breakout Board (MAX6675 upgrade)*. adafruit industries blog RSS. Retrieved April 28, 2023, from https://www.adafruit.com/product/269

[24] Wikimedia Foundation. (2023, February 28). *Reflow soldering*. Wikipedia. Retrieved April 28, 2023, from https://en.wikipedia.org/wiki/Reflow_soldering

[25] MicaSense Knowledge Base. (2022, September 30). Overview of agricultural indices . MicaSense. Retrieved November 28, 2022, from https://support.micasense.com/hc/en-us/articles/227837307-Overview-of-Agricultural-Indices

[26] Maschke, T. (2004). Digitale Kameratechnik: Technik digitaler Kameras in Theorie und Praxis. Springer Berlin Heidelberg.

[27] *Choosing the Right Imagery: Best Management Practices for Color, NIR, and NDVI Imagery | Integrated Crop Management*. (2016). Iastate.edu. https://crops.extension.iastate.edu/cropnews/2016/05/choosing-right-imagery-best-management-practices-color-nir-and-ndvi-imagery

[28] *IDB - Index DataBase*. (2023). Indexdatabase.de. https://www.indexdatabase.de/

[29] (2023). Innov8tivedesigns.com. https://www.innov8tivedesigns.com/images/specs/Cobra_2814-16_Specs.htm

[30] *X-uav Mini Talon EPO 1300mm Wingspan V-tail FPV RC Model Radio Remote*. (2023). RCDrone. https://rcdrone.top/products/x-uav-mini-talon-rc-kit

[31] *Airfoil database list*. (2023). Airfoiltools.com. http://airfoiltools.com/search/airfoils?m=a

[32] *Angle of Attack (AOA) | SKYbrary Aviation Safety*. (2015). Skybrary.aero. https://www.skybrary.aero/articles/angle-attack-aoa

[33] *Installation and Getting Started — OpenDroneMap 3.1.3 documentation*. (2020). Opendronemap.org. https://docs.opendronemap.org/installation/#id4

[34] *Tutorials — OpenDroneMap 3.1.3 documentation*. (2014). Opendronemap.org. https://docs.opendronemap.org/tutorials/

[35] *sidelap*. (2023). TheFreeDictionary.com. https://encyclopedia2.thefreedictionary.com/sidelap

[36] *Camera Multiplexer for Raspberry Pi 4: Arducam Solutions for Your Applications, and the Coming Surprises We Prepare for You - Arducam*. (2019, August 7). Arducam. https://www.arducam.com/arducam-multi-camera-multiplexer-raspberry-pi-4-application/

[37] *Raspberry Pi Documentation - Raspberry Pi hardware*. (2014). Raspberrypi.com. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

Appendices

Appendix A

RGB and IR Image to Spectral Analysis 350nm - 1000nm:

```
% Luke Shoen  Mount Vernon Nazarene University 4/10/2023
% This program requires two jpg files, one RGB and one IR. Also, the CIE
% 1931 Data is needed for RGB color analysis.
% Red subtractive difference method to find NIR Data
rgb_img = imread('IR_TEST1.jpg');
gray_img = rgb2gray(rgb_img);
red_channel = rgb_img(:,:,1);
nir_img = gray_img - red_channel;
wavelengths = linspace(650, 1000, size(nir_img, 2));
intensity = mean(nir_img, 1);
intensity = intensity/ max(intensity);
smoothed_intensity = smoothdata(intensity, 'movmean', 500);
p1 = plot(wavelengths, intensity, 'LineWidth', 0.5, 'Color', 'k', "LineStyle","-");
hold on
p2 = plot(wavelengths, smoothed_intensity,'Color','#A2142F','LineWidth', 2, "LineStyle"
,'-');
hold on
xlim([350 1000])
title('NIR Spectrum', 'FontSize',18);
%
% RGB to XYZ Tristimulus with Spectral Power Distribution Graph over
% visible spectrum range. This program needs a small .jpg file and an
% accompanying cie1931.mat color matching raw data.
%
rgbImage = imread('RGB_TEST1.jpg');
croppedImage = rgbImage(200:400, 200:400, :);
imwrite(croppedImage, 'cropped_image.jpg');
% Now need to convert RGB to XYZ tristimulus
% sRGB to XYZ conversion matrix
M = [0.4124564 0.3575761 0.1804375;
    0.2126729 0.7151522 0.0721750;
    0.0193339 0.1191920 0.9503041];
% Normalize RGB values to 0-1 range (dividing by 255)
croppedImage = double(croppedImage) ./ 255;
% Apply sRGB to XYZ conversion matrix (mutplies RGB values by 3x3 XYZ
% matrix, reshape is used to reshape RGB image data into 2D matrix with 3
% columns before multiplication
xyzImage = reshape(croppedImage, [], 3) * M;
% Reshape back to original image size
xyzImage = reshape(xyzImage, size(croppedImage));
```

```
% Convert XYZ to spectral data using CIE 1931 color matching functions
% Load CIE 1931 for cie1931.mat, 4 column matrix with conversion values for
% each wavlength.
load('cie1931.mat');
% Interpolate the color matching functions to match the image resolution.
% Each row of a column in selected using cie1931(:,x) and compared to the
% wavelength (column 1) to generate a [3x401] matrix cmf_interp descibing
% the cmf values for each wavelength within range (400nm total)
range = 380:1:780;
cmf_interp = [interp1(cie1931(:,1), cie1931(:,2), range, 'linear', 'extrap');
   interp1(cie1931(:,1), cie1931(:,3), range, 'linear', 'extrap');
   interp1(cie1931(:,1), cie1931(:,4), range, 'linear', 'extrap')];
% Normalize the color matching functions to a maximum of 1
cmf_interp = cmf_interp ./ max(cmf_interp(:));
% Calculate the spectral data using tristimulus values and color matching functions
% element wise multiplication of xyzImage and cmf_interp matricies,
% creating a 4th dimensional matrix called spectralData (row, column, RGB
% channels, and wavelength range)
spectralData = bsxfun(@times, xyzImage, reshape(cmf_interp, [1, 1, 3, numel(range)]));
spectralData = (sum(spectralData, [1, 2]))/2/10000;
% Plots the spectral curve, squeeze is used remove 2 dimensions of
% spectralData, making it 2 dimensional, which can be plotted as a function
h = plot(range, squeeze(spectralData), 'LineWidth', 2);
hold on
h(1).Color = 'r';
h(2).Color = 'g';
h(3).Color = 'b';
op = [h(1), h(2), h(3), p1, p2];
% Add legend with updated line colors
hold on
legend(op, 'Red Curve', 'Green Curve', 'Blue Curve','Raw NIR Data','Smoothed NIR Data',
'Location', 'northeast')
ax=gca;
ax.FontSize=18;
xlabel('Wavelength (nm)', 'FontSize', 18);
ylabel('Relative intensity','FontSize',18);
```

Appendix B

Custom PCB Code (main.c):

```
/* USER CODE BEGIN Header */
/**
Author: Owen Paulus - Mount Vernon Nazarene University
******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "fatfs.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
```

```
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 I2C_HandleTypeDef hi2c1;

SPI_HandleTypeDef hspi3;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
static const uint8_t Si7021_ADDR = 0x40<<1;   // 7-bit I2C address for Si7021
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_SPI3_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
```

```
 * @retval int
 */
int main(void)
{
 /* USER CODE BEGIN 1 */

        uint8_t buffer[12]; // buffer for data transfer between master and slave
        uint8_t buffer2[4];
        uint16_t code;      // raw data for temperature/humidity measurements
        float humidity;     // humidity measurement value
        float temperature;  // temperature measurement value
        double btemp;                   // battery temperature measurement value

 /* USER CODE END 1 */

 /* MCU Configuration--------------------------------------------------------*/

 /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
 HAL_Init();

 /* USER CODE BEGIN Init */

 /* USER CODE END Init */

 /* Configure the system clock */
 SystemClock_Config();

 /* USER CODE BEGIN SysInit */

 /* USER CODE END SysInit */

 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_USART2_UART_Init();
 MX_I2C1_Init();
 MX_FATFS_Init();
 MX_SPI3_Init();
```

```
  /* USER CODE BEGIN 2 */


//********SUBROUTINES********//




 //------------------- Functions ----------------//

  float Max31855_Read_Temp(uint8_t buffer[4]){
  uint8_t Error=0;                          // Thermocouple Connection acknowledge Flag
  uint32_t sign=0;                                                                  // Sign bit
  //uint8_t DATARX[4];
  int Temp=0;                               // Temperature Variable
  Error = buffer[3]&0x07;                                                           // Error Detection
  sign = (buffer[0]&(0x80))>>7;                                                     // Sign Bit
calculation

  if(buffer[3] & 0x07)                                                              // Returns
Error Number
  return(-1*(buffer[3] & 0x07));

  else if(sign==1){                                                                 //
Negative Temperature
  Temp = (buffer[0] << 6) | (buffer[1] >> 2);
  Temp&=0b01111111111111;
  Temp^=0b01111111111111;
  return((double)-Temp/4);
            }

  else
// Positive Temperature
  {
                    Temp = (buffer[0] << 6) | (buffer[1] >> 2);
                    return((double)Temp / 4);
  }
```

```
 }



 // Subroutine for conversion from raw relative humidity data to readable data per the Si7021 datasheet
(%RH)
 float process_humi_code(uint16_t humi_code)
          {
           float value = (float)(((125.0 * humi_code) / 65536.0) - 6.0);

           if(value < 0)
             return 0;
           else if(value > 100)
             return 100;
           else
             return (float)value;
          }

 // Subroutine for conversion from raw temperature data to readable data per the Si7021 datasheet (Celsius)
  float process_temp_code(uint16_t temp_code)
          {
           return (float)(((175.72 * temp_code) / 65536.0) - 46.85);
          }



 // Subroutine to convert uint8 to uint16
  uint16_t convert_to_uint16(uint8_t bytes1[])
 {
   return (uint16_t)((bytes1[0]<<8) | bytes1[1]);
 }


 while (1)
 {
```

```
//********Saving to MicroSD********
                        //********RH Measurement********//
                // Defining "cmd" as the command code to perform a RH measurement per the datasheet
                uint8_t cmd = 0xE5;

                // MCU sends command to the Si7021 I2C address to perform a RH measurement
            if(HAL_OK != HAL_I2C_Master_Transmit(&hi2c1, Si7021_ADDR, &cmd, 1, 10000))
              return -1;

            // MCU asks to receive the RH measurement data from the Si7021 and stores it in the buffer
            if(HAL_OK != HAL_I2C_Master_Receive(&hi2c1, Si7021_ADDR, buffer, 2, 10000))
              return -1;

            // the data within the buffer is converted to an unsigned integer ranging from 0 to 65535
            code = convert_to_uint16(buffer);
            // the unsigned integer (raw data) is passed through the humidity processing subroutine for
readable conversion
            humidity = process_humi_code(code);

            // The converted RH measurement is prepared to be displayed on the serial monitor
            sprintf((char*)buffer, "%u.%u RH\r\n",((unsigned int)humidity / 100),((unsigned int)humidity
% 100));

            // The converted data is sent to serial monitor display
            HAL_UART_Transmit(&huart2, buffer, strlen((char*)buffer), HAL_MAX_DELAY);
                HAL_Delay(500);




          //********Temperature Measurement********//
            // Defining "cmd1" as the command code to perform a temperature measurement after an RH
measurement per the datasheet
            uint8_t cmd1 = 0xE0;

            // MCU sends command to the Si7021 I2C address to perform a temperature measurement
```

```
            if(HAL_OK != HAL_I2C_Master_Transmit(&hi2c1, Si7021_ADDR, &cmd1, 1,
HAL_MAX_DELAY))
              return -1;

            // MCU asks to receive the temperature measurement data from the Si7021 and stores it in the
buffer
            if(HAL_OK != HAL_I2C_Master_Receive(&hi2c1, Si7021_ADDR, buffer, 2,
HAL_MAX_DELAY))
              return -1;

            // the data within the buffer is converted to an unsigned integer ranging from 0 to 65535
            code = convert_to_uint16(buffer);
            // the unsigned integer (raw data) is passed through the temperature processing subroutine for
readable conversion
            temperature = process_temp_code(code);
            // the converted temperature value is multiplied by 100 to display an accurate decimal reading in
degrees Celsius
            temperature *= 100;

            // The converted temperature measurement is prepared to be displayed on the serial monitor
            sprintf((char*)buffer, "%u.%u C Ambient\r\n",((unsigned int)temperature / 100),
                  ((unsigned int)temperature % 100));

            // The converted data is sent to serial monitor display
            HAL_UART_Transmit(&huart2, buffer, strlen((char*)buffer), HAL_MAX_DELAY);
                  HAL_Delay(500);




          //********Battery Temperature Measurement********//
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);       // Low State for SPI
Communication
                  HAL_Delay(250);
            HAL_SPI_Receive(&hspi3,buffer2,4,1000);              // DATA Transfer
                  HAL_Delay(250);
```

```
            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);       // High State for SPI
Communication

        btemp = Max31855_Read_Temp(buffer2);                    // Conversion to readable temperature
data in Celsius
        btemp *= 100;

        // The converted temperature measurement is prepared to be displayed on the serial monitor
        sprintf((char*)buffer2, "%u.%u C Battery\r\n",((unsigned int)btemp / 100),
                                        ((unsigned int)btemp % 100));

        // The converted data is sent to serial monitor display
        HAL_UART_Transmit(&huart2, buffer2, strlen((char*)buffer2), HAL_MAX_DELAY);
                HAL_Delay(5000);



 }
  /* USER CODE END WHILE */
```

Appendix C

Weather Station Arduino Code:

Author: Owen Paulus, Mount Vernon Nazarene University

```
#include <Adafruit_Si7021.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 64  // OLED display height, in pixels
#define OLED_RESET    4  // Reset pin # (or -1 if sharing reset pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

float RH = 0;  // Relative Humidity Value
float Temp = 0; // Temperature (C) Value
float TempF = 0; // Temperature (F) Value
float Windms = 0; // Wind Speed (m/s) Value
float Windkmh = 0; // Wind Speed (km/h) Value
float Windkt = 0; // Wind Speed (knots) Value
float Windmph = 0; // Wind Speed (mph) Value



Adafruit_Si7021 sensor = Adafruit_Si7021();

void setup() {
  // put your setup code here, to run once:
sensor.begin();

Serial.begin(9600);

  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {  // I2C Address 0x3C
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
```

```
  delay(1000);

  display.clearDisplay();  // Clear OLED Screen
  display.setTextSize(2);  // Set Text Size
  display.setTextColor(WHITE); // Set Text Color
}

void loop() {
  // put your main code here, to run repeatedly:
                                        //*****Analog Values*****//
int pot = analogRead(A1); // Potentiometer Value
int anemometer = analogRead(A0); // Anemometer Analog Voltage

// m/s = (a-82)/204.0*32.4   204=1V
                                        //*****Unit Conversions*****//
Temp = sensor.readTemperature();  // Read Temperature in C
RH = sensor.readHumidity();       // Read Relative Humidity
TempF = (Temp*1.8)+32;            // Temperature in F
Windms = (anemometer-82)/204.0*32.4; // Wind Speed in m/s
Windkmh = Windms*3600/1000;         // Wind Speed in km/h
Windkt = Windkmh/1.852;            // Wind Speed in knots
Windmph= Windkmh/1.609;              // Wind Speed in mph
                                        //*****Temperature*****//
  display.clearDisplay();
  display.setCursor(0,0);
  display.print("T=");
  display.setCursor(26,0);

  if (pot>100){
  display.print(Temp);
  display.setCursor(90,0);
  display.print("C");

  } else {

  display.print(TempF);
  display.setCursor(90,0);
```

```
  display.print("F");
}

                                        //*****Relative Humidity*****//
  display.setCursor(0,24);
  display.print("RH=");
  display.setCursor(37,24);
  display.print(RH);
  display.setCursor(100,24);
  display.print("%");


                                        //*****Wind Speed*****//
  display.setCursor(0,48);
  display.print("WS=");
  display.setCursor(37,48);

  if (pot>640){
  display.print(Windms,1);
  display.setCursor(90,48);
  display.print("m/s");
  }

else if (pot>380&pot<640){
  display.print(Windkmh,1);
  display.setCursor(90,48);
  display.print("kmh");
}

else if (pot>100&pot<380){
  display.print(Windkt,1);
  display.setCursor(90,48);
  display.print("kt");
}
else {
  display.print(Windmph,1);
  display.setCursor(90,48);
```

```
  display.print("mph");
}


  display.display();
  delay(500);
}
```

Appendix D

RGB and IR image to NDVI MATLAB Test Program

```
%Luke Shoen  Mount Vernon Nazarene University 4/27/2023
% INFRARED AND RGB IMAGE TO NDVI IMAGE IN HSV COLORMAP
rgb_img = imread('rgb1.jpg');
nir_img = imread('nir1.jpg');
% will have to change trans val manually
nirtrans = imtranslate(nir_img, [130, 210]);
rgb_img_cropped = im2double(rgb_img);
nir_img_cropped = im2double(nirtrans);
ndvi = (nir_img_cropped - rgb_img_cropped) ./ (nir_img_cropped + rgb_img_cropped);
ndvi_gray = rgb2gray(ndvi);
imshow(ndvi_gray)
imwrite(ndvi_gray, 'ndvigray.jpg')
img = imread('ndvigray.jpg');
img_rgb = ind2rgb(img, hsv);
img_size = size(img_rgb);
crop_height = round(img_size(1)*0.80);
img_rgb_cropped = img_rgb(ceil((img_size(1)-
crop_height)/2):floor((img_size(1)+crop_height)/2), :, :);
imshow(img_rgb_cropped);
imwrite(img_rgb_cropped,'NDVIfin.jpg')
caxis([-1 1])
colormap_subset = hsv(64); % Use 64 colors
colormap_subset = colormap_subset(1:23,:); % Use only a subset of the colormap
colormap(colormap_subset) % Set the colormap for the colorbar
Colorbar
```

Appendix E

Original Sample RGB and IR Photos for NDVI Analysis

Appendix F

Derived Governing Equations

UAV Optimization Governing Equations

Nathaniel T. Dersom

Department of Engineering, Mount Vernon Nazarene University

Senior Design I

Dr. Oommen, Dr. Majors, Dr. Meng, and Dr. Sob

October 17, 2022

AIM

To derive equations that can be used to optimize the design of an unmanned aerial vehicle (UAV).

THEORY

Part 1: Relating vehicle speed to necessary frame rate of imagery instrument

As a UAV flies over a field, a photo taken encompasses a certain ground size dependent on the altitude of the vehicle and the specifications of the camera sensor. This is imaged in figure 1 below.

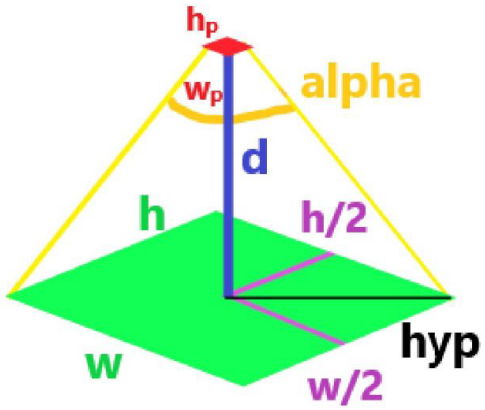


Fig. 1. Geometric variables of aerial photography

As such, the following equations are found, (See Appendix A for detailed derivation)

$$\rho_p = \frac{{h_p}^2 + {w_p}^2}{\left[2d \tan\left(\frac{\alpha}{2}\right)\right]^2} \quad \text{(Eq. 1.1)}$$

$$h = \frac{h_p}{\sqrt{\rho_p}} \quad \text{(Eq. 1.2)}$$

$$w = \frac{w_p}{\sqrt{\rho_p}} \quad \text{(Eq. 1.3)}$$

where:

ρ_p :	Pixel density	$\left[\frac{\text{Pixels}}{\text{m}^2}\right]$ | $\left[\frac{\text{Pixels}}{\text{ft}^2}\right]$
h_p :	Camera sensor height	[Pixels]
w_p :	Camera sensor width	[Pixels]
h :	Ground height	[m] | [ft]
w :	Ground width	[m] | [ft]
d :	Altitude	[m] | [ft]
α :	Camera angle of view	[degrees] | [radians]

In order for the photos taken from the drone to be stitched properly into one photo, an overlap factor of x is used to determine the distance between photo locations. Using this factor, a relationship between frames-per-second (fps) and ground velocity can be determined. However, the orientation of the camera sensor (ex: width-leading or height-leading) affects the number of photos needed per distance. This yields the following equations: (detailed derivations can be found in Appendix A)

For width-leading:

$$V_{ground} = \frac{h_p f_r (1 - x)}{\sqrt{\rho_p}} \quad \text{(Eq. 1-4)}$$

$$f_r = \frac{V_{ground}\sqrt{\rho_p}}{h_p(1-x)} \quad \text{(Eq. 1-5)}$$

For height-leading:

$$V_{ground} = \frac{w_p f_r (1-x)}{\sqrt{\rho_p}} \quad \text{(Eq. 1-6)}$$

$$f_r = \frac{V_{ground}\sqrt{\rho_p}}{w_p(1-x)} \quad \text{(Eq. 1-7)}$$

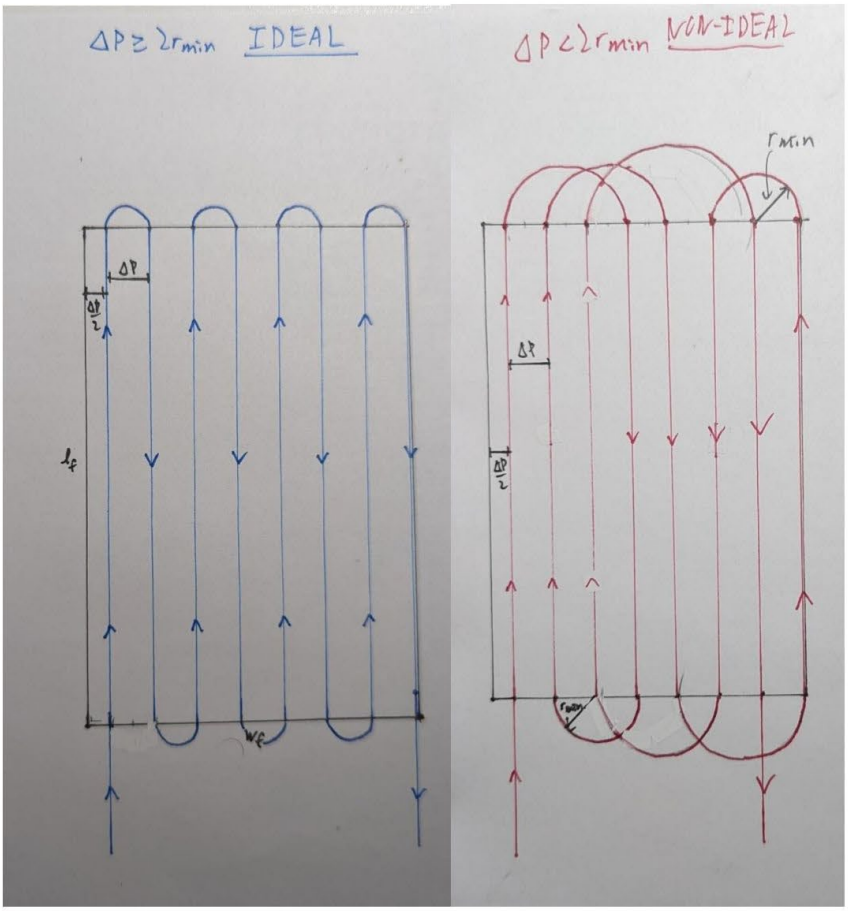
where:

V_{ground} :	Ground speed	$\left[\frac{\text{m}}{\text{s}}\right]$ | $\left[\frac{\text{ft}}{\text{s}}\right]$
f_r :	Framerate	$\left[\frac{\text{frames}}{\text{s}}\right]$

Thus, the maximum velocity of the plane can be optimized based on the maximum framerate of the camera. This is when $V_{ground} = V_{ground,max}$ and $f_r = f_{r,max}$, found by using Eq. 1-6 or 1-4 based on the orientation of the camera sensor.

Part 2: Flight path equations

There are two conditions for the ideal flight path: ideal and non-ideal. The definition of an ideal path is when the distance between each pass is greater than two times the minimum turning radius of the UAV, r_{min}. Otherwise, the path is non-ideal. These two concepts are visualized in Figs. 2 and 3 below.



Figs. 2 and 3 Diagrams of ideal and non-ideal survey patterns

Non-ideal paths are very difficult to plot paths for without human intervention. Later research

will need to be done to provide an ideal path when the distance between passes is less than the minimum turning diameter of the UAV.

The pass distance is given below.

$$\text{Width-leading: } \Delta P = \frac{w_p(1-x)}{\sqrt{\rho_p}} \quad \text{(Eq. 2-1)}$$

$$\text{Height-leading: } \Delta P = \frac{h_p(1-x)}{\sqrt{\rho_p}} \quad \text{(Eq. 2-2)}$$

where:

ΔP:	Distance between passes	[m] | [ft]
hp:	Pixel height of camera	[Pixels]
w_p:	Pixel width of camera	[Pixels]
x:	Overlap factor	[unitless]
ρ_p:	Pixel density	$\left[\frac{\text{Pixels}}{\text{m}^2}\right]$ | $\left[\frac{\text{Pixels}}{\text{ft}^2}\right]$

The minimum turning radius is given by Eq. 2-3[1], below.

$$r_{min} = \frac{{V_{air,min}}^2}{g \tan\theta} \quad \text{(Eq. 2-3)}$$

where:

r_{min}:	Minimum turning radius	[m] | [ft]
g:	Acceleration due to gravity	$\left[\frac{\text{m}}{\text{s}^2}\right]$
θ:	Angle of banking	[degrees] | [radians]
$V_{air,min}$:	Stalling airspeed	$\left[\text{ft}/\text{s}\right]$ | $[\text{m}/\text{s}]$

The stalling speed can also be found using Equation 2-4 below.[2]

$$V_{air.min} = \sqrt{\frac{2L}{Cl \cdot \rho \cdot S}} \quad \text{(Eq. 2-4)}$$

where:

L:	Lift force	[N] | [lb]
Cl:	Coefficient of lift	

[1] https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/phak/media/06_phak_ch4.pdf
[2] https://www.grc.nasa.gov/www/k-12/rocket/lifteq.html

ρ: Density of air [Pa] | [psi]
S: Surface area of wings [m^2] | [ft^2]

Thus, if it is assumed that the minimum required lift force is equal to the force of gravity on the plane, the stalling speed of the UAV can be found below.

$$V_{air.min} = \sqrt{\frac{2mg}{Cl \cdot \rho \cdot S}} \quad \text{(Eq. 2-5)}$$

Combining Eq. 2-3 and 2-5 yields the following.

$$r_{min} = \frac{2 \cdot m}{Cl \cdot \rho \cdot S \cdot \tan\theta} \quad \text{(Eq. 2-6)}$$

Below is the path condition. If true, the path is ideal.

$$\Delta P \geq 2r_{min} \quad \text{(Eq. 2-7)}$$

The number of passes that needs to be made for a given rectangular field is given below.

$$n_p = ceiling\left(\frac{w_f}{\Delta P}\right) \quad \text{(Eq. 2-8)}$$

where:

n_p: Minimum number of passes
w_f: Shorter side of field [m] | [ft]

If ideal, the turning radius, r, of each pass is just half of the pass distance, ΔP.

Thus, the total flight distance for a field of known dimensions is given below.

$$L_f = straight_d + turning_d = (l_f n_p) + (\pi r n_p - \pi r) \quad \text{(Eq. 2-9)}$$

where:

L_f: Total flight distance for field [m] | [ft]
l_f: Longer side of field [m] | [ft]

Part 3: Flight time and power consumption

Total flight time for a given field can simply be calculated by finding the sum of the total time turning and the total time going straight on passes. This is given by Eq. 3-1 below.

$$F_t = t_{straight} + t_{turning} = \frac{straight_d}{V_{avg,straight}} + \frac{turning_d}{V_{avg,turning}} \quad \text{(Eq. 3-1)}$$

Further, energy consumption can be found by using Eq. 3-2 below.

$$E_f = \frac{P_s t_s + P_T t_T}{3600} \quad \text{(Eq. 3-2)}$$

where:

P_s:	Power consumption going straight	[W]
t_s:	Total time going straight	[s]
P_T:	Power consumption while turning	[W]
t_T:	Total time spent turning	[s]
E_f:	Energy spent for a field	[Wh]

APPENDIX A: Derivation of Equations 1-1 to 1-7

Equation A-1 is found by applying Pythagorean's theorem to Fig. A-1.

$$hyp = \sqrt{\left(\frac{w}{2}\right)^2 + \left(\frac{h}{2}\right)^2} \quad \text{(Eq. A-1)}$$

$$hyp = d \tan\left(\frac{\alpha}{2}\right) \quad \text{(Eq. A-2)}$$

where:

hyp:	Hypotenuse shown in Fig. A-1	[m] | [ft]
w:	Width of frame	[m] | [ft]
h:	Height of frame	[m] | [ft]
d:	Altitude of camera	[m] | [ft]
α:	Angle of view of camera sensor	[m] | [ft]

Thus, equations A-1 and A-2 can be combined to yield the following.

$$d \tan\left(\frac{\alpha}{2}\right) = \sqrt{\left(\frac{w}{2}\right)^2 + \left(\frac{h}{2}\right)^2} \quad \text{(Eq. A-3)}$$

The definition of pixel density is given below in Eq. A-4.

$$\rho_p = \frac{A_p}{A} = \frac{w_p h_p}{wh} \quad \text{(Eq. A-4)}$$

where:

A:	Ground area of the frame	[m^2] | [ft^2]
A_p:	Number of pixels in frame	[Pixels]
w_p:	Pixel width of frame	[Pixels]
h_p:	Pixel height of frame	[Pixels]

It is known that the ratio between the height and width of the frame is the same in pixels and in square area because the projection of a smaller rectangle is similar to a larger one, seen in Fig. A-1.

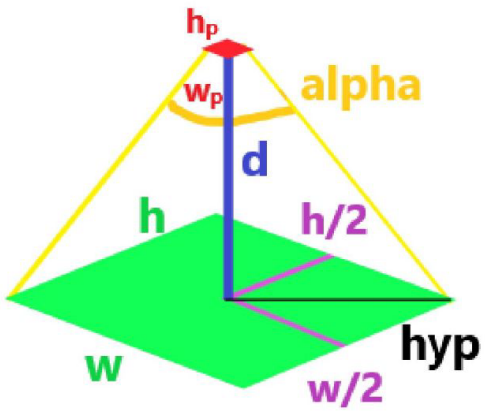


Fig. A-1. Diagram of camera sensor geometry

This assumption is shown in Eq. A-5 below.

$$\frac{w_p}{h_p} = \frac{w}{h} \quad \text{(Eq. A-5)}$$

Thus, Eq. A-5 can be rewritten into the following equations.

$$w = \frac{hw_p}{h_p} \quad \text{(Eq. A-6)}$$

$$h = \frac{wh_p}{w_p} \quad \text{(Eq. A-7)}$$

Equation A-4 can be rewritten to isolated *w*, as given below.

$$w = \frac{w_p h_p}{h\rho_p} \quad \text{(Eq. A-8)}$$

Thus, Eq. A-6 and A-8 can be combined to yield the following.

$$\frac{hw_p}{h_p} = \frac{w_p h_p}{h\rho_p} \quad \text{(Eq. A-9)}$$

Equation A-9 can be simplified to yield Eq. A-10.

$$h = \frac{h_p}{\sqrt{\rho_p}} \quad \text{(Eq. 1-2)}$$

Similarly, A-11 can be derived.

$$w = \frac{w_p}{\sqrt{\rho_p}} \quad \text{(Eq. 1-3)}$$

Thus, Eq. 1-2 and 1-3 can be substituted back into Eq. A-3.

$$d\tan\left(\frac{\alpha}{2}\right) = \sqrt{\left(\frac{\left(\frac{w_p}{\sqrt{\rho_p}}\right)}{2}\right)^2 + \left(\frac{\left(\frac{h_p}{\sqrt{\rho_p}}\right)}{2}\right)^2} \quad \text{(Eq. A-12)}$$

Equation A-12 can be simplified to yield Eq. A-13 below.

$$\rho_p = \frac{{h_p}^2 + {w_p}^2}{\left[2d\tan\left(\frac{\alpha}{2}\right)\right]^2} \quad \text{(Eq. 1-1)}$$

The frame rate of the camera is given by the equation below.

$$f_r = \frac{V_{ground}}{\Delta x} \quad \text{(Eq. A-14)}$$

where:

V_{ground}:	Ground velocity of UAV	$\left[{}^{m}/_{s}\right] \mid \left[{}^{ft}/_{s}\right]$
Δx:	Distance between frames	[m] | [ft]
f_r:	Framerate of camera	[fps]

For different software stitching suites, there must be a certain percentage of overlap between consecutive frames. This is used to calculate the distance between frames. This is given by the *x*

variable. Depending on whether the camera's orientation, two different sets of equations will be derived. Eq. A-15 assumes width-facing and Eq. A-16 assumes height-facing motion.

$$\text{Width-facing:} \quad \Delta x = h(1-x) \qquad \text{(Eq. A-15)}$$
$$\text{Height-facing:} \quad \Delta x = w(1-x) \qquad \text{(Eq. A-16)}$$

Thus, equations A-15 and A-16 can be substituted into equation A-14 to yield the following equations.

$$\text{Width-facing:} \quad f_r = \frac{V_{ground}}{h(1-x)} \qquad \text{(Eq. A-17)}$$
$$\text{Height-facing:} \quad f_r = \frac{V_{ground}}{w(1-x)} \qquad \text{(Eq. A-18)}$$

Then Eq. A-10 and Eq. A-11 can be substituted in Eq. A-17 and A-18.

$$\text{Width-facing:} \quad f_r = \frac{V_{ground}}{\left(\frac{h_p}{\sqrt{\rho_p}}\right)(1-x)} \qquad \text{(Eq. A-17)}$$
$$\text{Height-facing:} \quad f_r = \frac{V_{ground}}{\left(\frac{w_p}{\sqrt{\rho_p}}\right)(1-x)} \qquad \text{(Eq. A-18)}$$

Equations A-17 and A-18 can be simplified to yield the following

$$\text{Width-facing:} \quad f_r = \frac{V_{ground}\sqrt{\rho_p}}{h_p(1-x)} \qquad \text{(Eq. 1-5)}$$
$$\text{Height-facing:} \quad f_r = \frac{V_{ground}\sqrt{\rho_p}}{w_p(1-x)} \qquad \text{(Eq. 1-7)}$$

Equations A-19 and A-20 can be solved for V_{ground} to yield the following.

$$\text{Width-facing:} \quad V_{ground} = \frac{h_p f_r(1-x)}{\sqrt{\rho_p}} \qquad \text{(Eq. 1-4)}$$
$$\text{Height-facing:} \quad V_{ground} = \frac{w_p f_r(1-x)}{\sqrt{\rho_p}} \qquad \text{(Eq.1-6)}$$

Appendix G

Explanation of Networking Protocols

A Virtual Private Network (VPN) is a network technology that allows a user to create a secure and encrypted connection over the internet to another network, such as a company network, a university network, or a public Wi-Fi hotspot.

When a user connects to a VPN, their device (the client) creates a secure and encrypted tunnel to the VPN server. This is done by encapsulating the data packets in an additional layer of encryption, which makes it difficult for anyone to intercept or view the data being transmitted over the internet.

The encryption used in a VPN is typically based on the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol. This encryption protocol establishes a secure and encrypted connection between the client and the VPN server, which is used to transmit data securely over the internet.

Once the VPN connection is established, the user's device appears as if it is on the same local network as the VPN server, even if it is located in a different geographic location. This allows the user to access resources that are only available on that network, such as files, applications, and services.

A remote desktop session utilizing Microsoft’s proprietary Remote Desktop Protocol (RDP) allows for the remote-control of a desktop environment. When a user initiates an RDP session, their device (the client) sends a request to the target device (the server) to establish a remote desktop session. The server responds by setting up a virtual display screen and sending that display to the client.

The client then renders the display on the user's screen, which enables them to see and interact with the target device's desktop environment. This allows the user to remotely control the target device as if they were physically present at that device.

Combining these two protocols, VPN and RDP, a client can control when they would like to be able to be controlled by a remote device by using the tunnel in reverse. Thus, a client on the same VPN server can initialize an RDP session to control the other client. If the original client does not want to be able to be controlled remotely, they can disable the VPN session, cutting off the RDP session. This serves as both a safety and privacy feature.

Being a client on a VPN does not require any port forwarding. However, if the VPN does not connect, there is no ability for the remote user to connect to the device. This is not ideal for devices that are running headless, having no input or output devices connected. Instead, a secure-socket layer (SSL) tunnel is more ideal. VPNs utilize this tunneling method, but using the protocol directly allows for better use in daemons. Cloudflare has programmed a very good SSL daemon called `cloudflared` and is readily available on almost all linux distributions. The user can simply choose what services they would like forwarded to a domain, and these services can be accessed from anywhere without port forwarding. This is all part of Cloudflare's Zero Trust security suite included in their domain name system (DNS) hosting subscription.

Appendix H

Open Drone Maps Example Report

ODM Quality Report

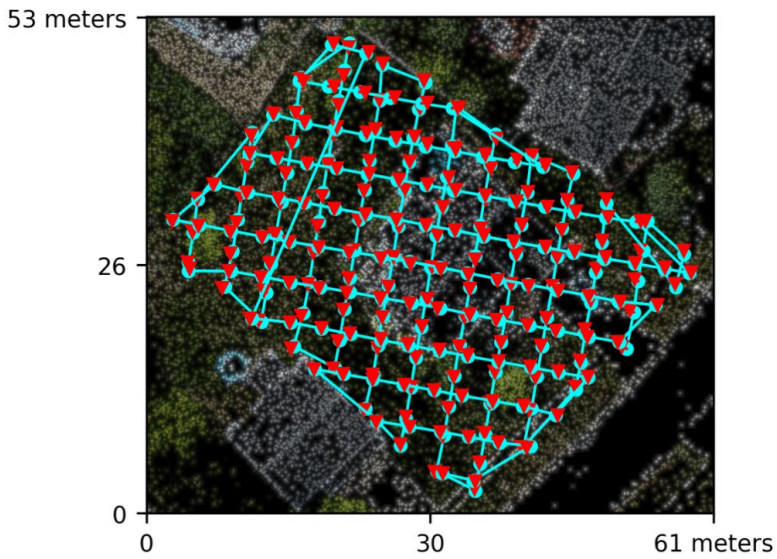
Processed with ODM version 3.1.1

Dataset Summary

Date	28/04/2023 at 05:20:18
Area Covered	0.002648 km²
Processing Time	34.0m:17.0s
Capture Start	01/09/2022 at 16:03:24
Capture End	01/09/2022 at 16:12:37

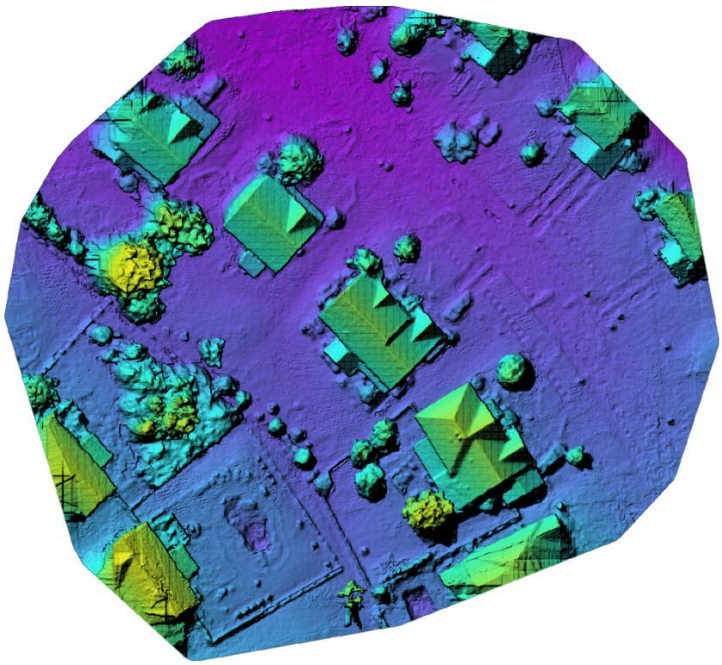
Processing Summary

Reconstructed Images	222 over 222 shots (100.0%)
Reconstructed Points (Sparse)	306517 over 312661 points (98.0%)
Reconstructed Points (Dense)	19,422,297 points
Average Ground Sampling Distance (GSD)	2.0 cm
Detected Features	13,116 features
Reconstructed Features	7,059 features
Geographic Reference	GPS
GPS errors	0.36 meters



Previews

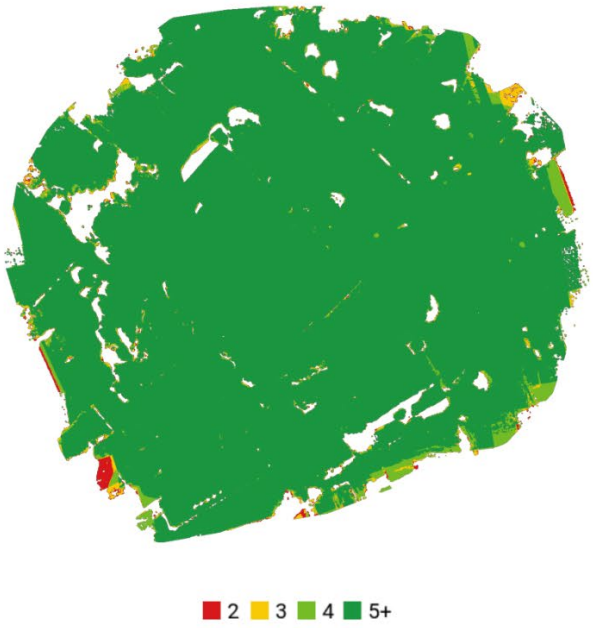
Orthophoto



Digital Surface Model

187.03m 202.89m

Survey Data



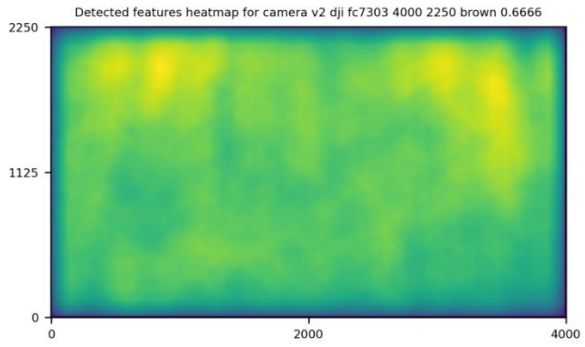
GPS/GCP/3D Errors Details

GPS	Mean	Sigma	RMS Error
X Error (meters)	0.001	0.178	0.178
Y Error (meters)	-0.001	0.164	0.164
Z Error (meters)	0.001	0.302	0.302
Total			0.358

3D	Mean	Sigma	RMS Error
X Error (meters)	0.027	0.092	0.096
Y Error (meters)	0.027	0.100	0.104
Z Error (meters)	0.083	0.236	0.250
Total			0.098

	Absolute	Relative
Horizontal Accuracy CE90 (meters)	0.387	0.084
Vertical Accuracy LE90 (meters)	0.471	0.159

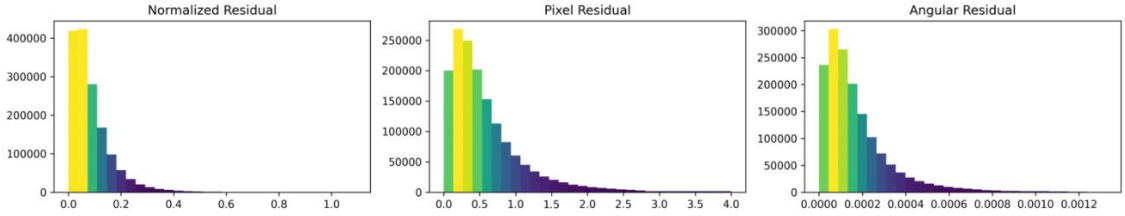
Features Details



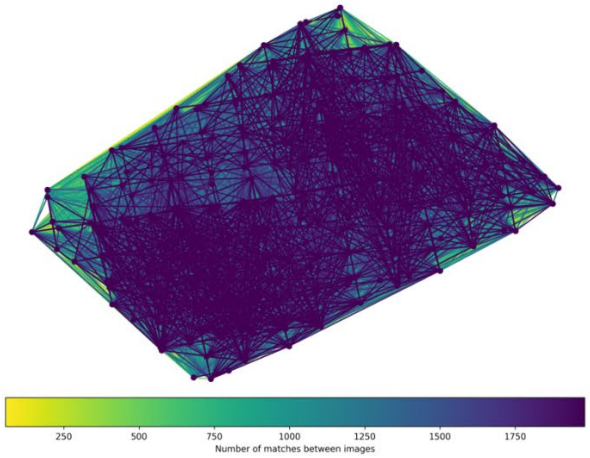
	Min.	Max.	Mean	Median
Detected	10049	19082	13451	13116
Reconstructed	3623	10459	7089	7059

Reconstruction Details

Average Reprojection Error (normalized / pixels / angular)	0.09 / 0.59 / 0.00017
Average Track Length	5.13 images
Average Track Length (> 2)	8.30 images



Tracks Details

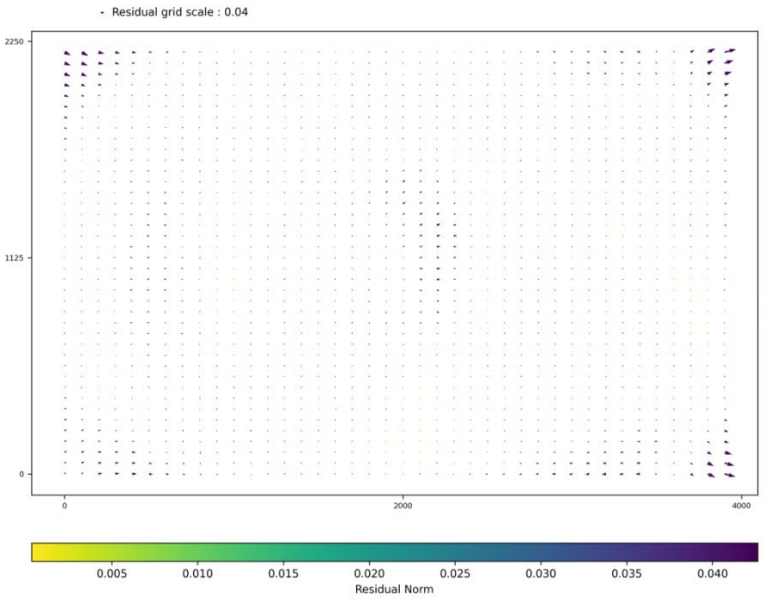


Length	2	3	4	5	6	7	8	9	10
Count	154087	53409	26076	15358	10193	7099	5272	4146	3318

Camera Models Details

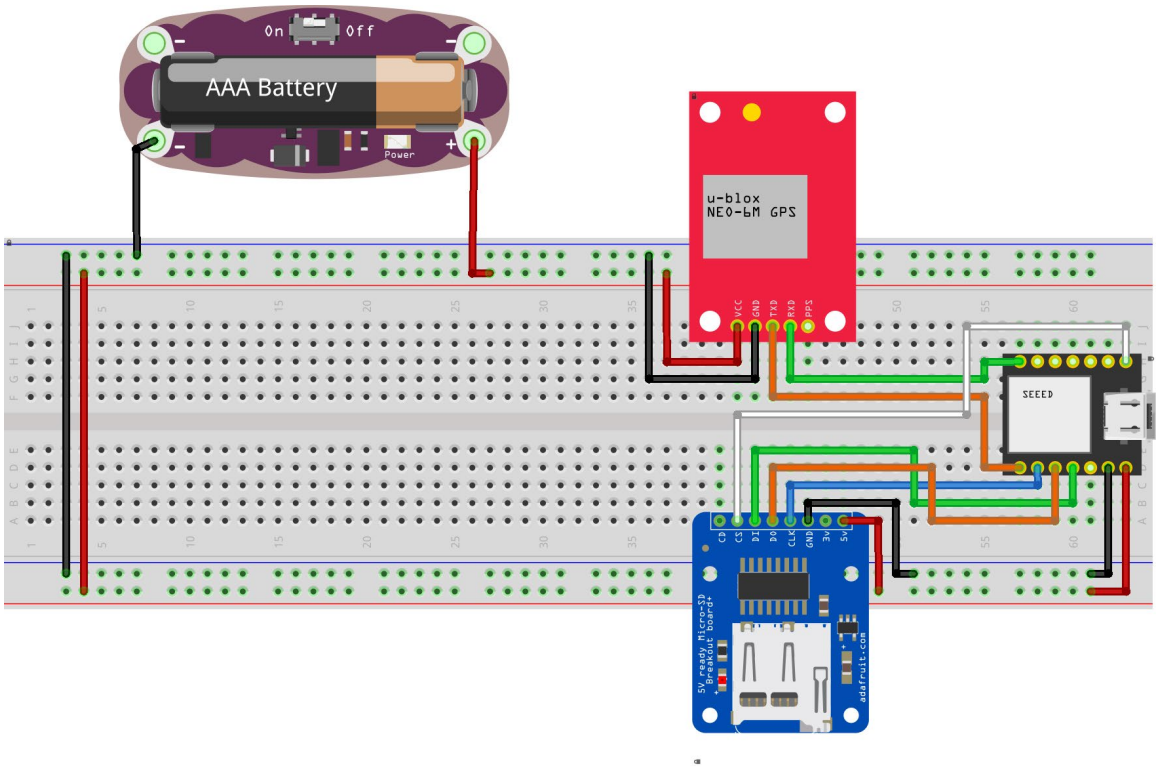
v2 dji fc7303 4000 2250 brown 0.6666

	k1	k2	k3	p1	p2	focal	aspect_ratio	cx	cy
Initial	0.0000	0.0000	0.0000	0.0000	0.0000	0.6667	1.0000	0.0000	0.0000
Optimized	-0.0025	0.0133	-0.0126	0.0013	0.0002	0.7503	1.0000	-0.0003	0.0026



Appendix I

Global Positioning System Testing Breadboard and Code




```
from board import *
from time import *
import board
import busio
import sdcardio
import storage
import time
import adafruit_gps
from digitalio import DigitalInOut, Direction

sleep(1)

spi = busio.SPI(board.D8, board.MOSI, board.MISO)
```

```
cs = board.D0
sd = sdcardio.SDCard(spi, cs)

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT

vfs = storage.VfsFat(sd)
storage.mount(vfs, '/sd')

uart = busio.UART(board.TX, board.RX, baudrate=9600,
timeout=10)
gps = adafruit_gps.GPS(uart, debug=True)
gps.send_command(b"PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0") #record standard gps data
gps.send_command(b"PMTK220,5000") #wait 5 seconds between
readings

last_print = time.monotonic()

while True:
     with open("/sd/pico.txt", "w") as file:
     led.value = True
     # Make sure to call gps.update() every loop iteration
and at least twice
     # as fast as data comes from the GPS unit (usually
every second).
     # This returns a bool that's true if it parsed new
data (you can ignore it
     # though if you don't care and instead look at the
has_fix property).
     gps.update()
```

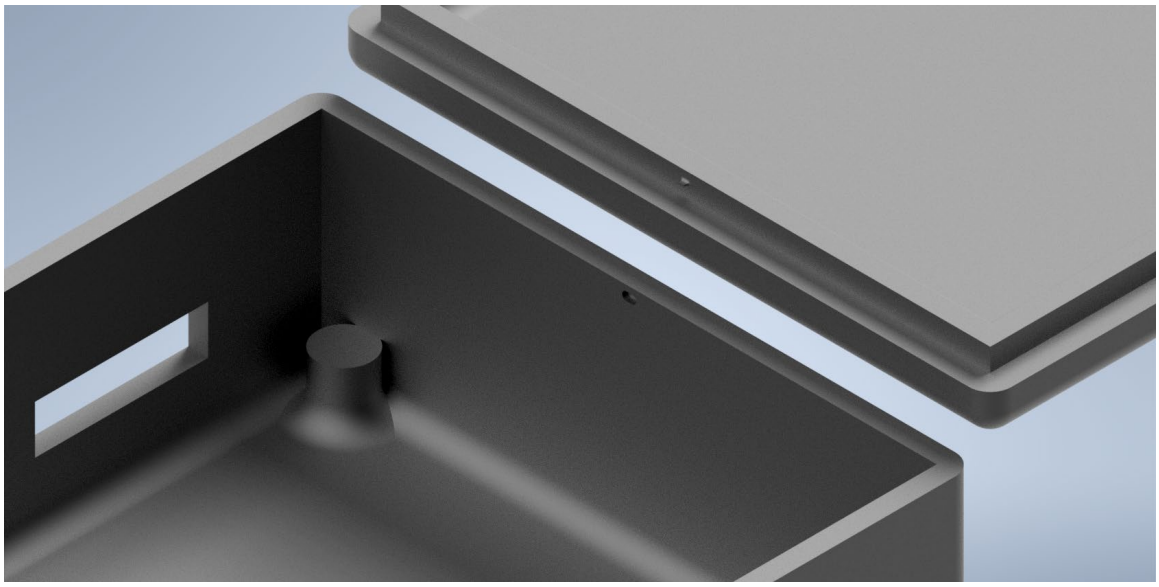
```
     # Every second print out current location details if
there's a fix.
     current = time.monotonic()
     led.value = False
     if current - last_print >= 1.0:
          last_print = current
          if not gps.has_fix:
               # Try again if we don't have a fix yet.
               print("Waiting for fix...")
               continue
          # We have a fix! (gps.has_fix is true)
          # Print out details about the fix like location,
date, etc.
          file.write("=" * 40)  # Print a separator line.
          file.write(
               "Fix timestamp: {}/{}/{}
{:02}:{:02}:{:02}".format(
                    gps.timestamp_utc.tm_mon,  # Grab parts
of the time from the
                    gps.timestamp_utc.tm_mday,  #
struct_time object that holds
                    gps.timestamp_utc.tm_year,  # the fix
time.  Note you might
                    gps.timestamp_utc.tm_hour,  # not get
all data like year, day,
                    gps.timestamp_utc.tm_min,  # month!
                    gps.timestamp_utc.tm_sec,
               )
          )
          file.write("Latitude: {0:.6f}
degrees".format(gps.latitude))
```

```
          file.write("Longitude: {0:.6f}
degrees".format(gps.longitude))
          file.write(
               "Precise Latitude: {:2.}{:2.4f}
degrees".format(
                    gps.latitude_degrees,
gps.latitude_minutes
               )
          )
          file.write(
               "Precise Longitude: {:2.}{:2.4f}
degrees".format(
                    gps.longitude_degrees,
gps.longitude_minutes
               )
          )
          file.write("Fix quality:
{}".format(gps.fix_quality))
          # Some attributes beyond latitude, longitude and
timestamp are optional
          # and might not be present.  Check if they're
None before trying to use!
          if gps.satellites is not None:
               file.write("# satellites:
{}".format(gps.satellites))
          if gps.altitude_m is not None:
               file.write("Altitude: {}
meters".format(gps.altitude_m))
          if gps.speed_knots is not None:
               file.write("Speed: {}
knots".format(gps.speed_knots))
          if gps.track_angle_deg is not None:
```

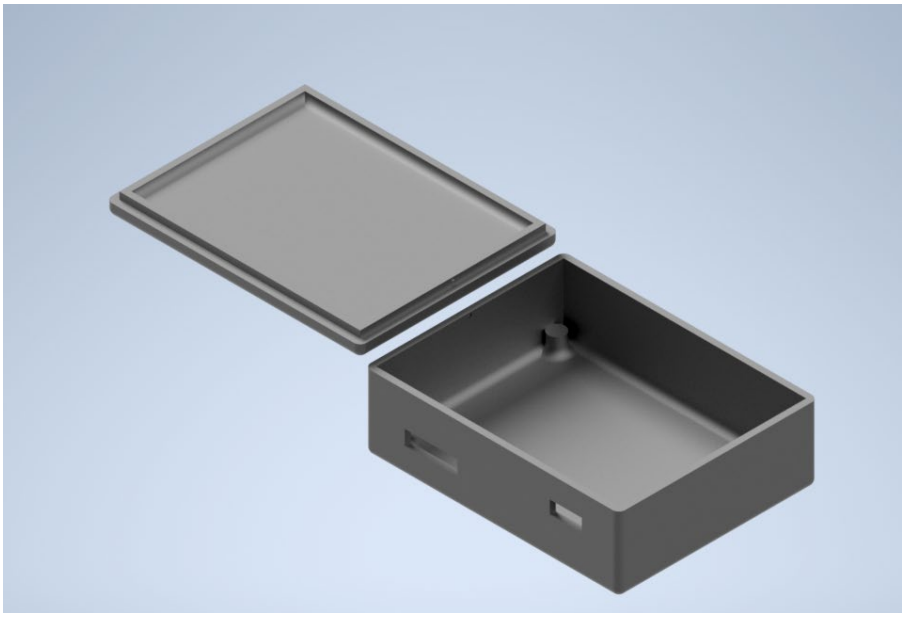
```
                file.write("Track angle: {}
degrees".format(gps.track_angle_deg))
          if gps.horizontal_dilution is not None:
                file.write("Horizontal dilution:
{}".format(gps.horizontal_dilution))
          if gps.height_geoid is not None:
                file.write("Height geoid: {}
meters".format(gps.height_geoid))
```

Appendix J

Global Positioning System Testing Case Renders



Render of the Lid Locking Close-up



Render of the Open Case

Render of the completed case

Appendix K

Global Positioning System Testing Data

```
*
Fix timestamp: 11/6/2022 00:46:15
Lat: 15.470757
Long: -90.386906
Fix qual: 2
# sats: 14, Alt: 1309.5 m, Speed: 0.047 knots, Horizontal
dilution: 1.06, Height geoid: -5.7 m
*
Fix timestamp: 11/6/2022 00:46:16
Lat: 15.470757
Long: -90.386906
Fix qual: 2
# sats: 14, Alt: 1309.6 m, Speed: 0.037 knots, Horizontal
dilution: 1.06, Height geoid: -5.7 m
*
Fix timestamp: 11/6/2022 00:46:17
Lat: 15.470761
Long: -90.386906
Fix qual: 2
# sats: 14, Alt: 1309.7 m, Speed: 0.059 knots, Horizontal
dilution: 1.06, Height geoid: -5.7 m
```

Appendix L

Global Positioning System Testing Instructions

MVNU Engineering Department

Senior Design 2022

GPS Logger Instructions

Notes:

- While the device has power, your location will be logged every second. If you want privacy, just unplug the device from power. The data will not be corrupted.
- The included battery pack should be able to last at least 12 hours based on one test. **So, p**lease charge the battery nightly with the given cable.
- The purpose of this device is to find the quality of GPS data in Guatemala. This will be done by determining the "fix quality" of recorded data. Below is an example of what data will be recorded. This data will also be cross-referenced with known map data for the region in order to find the overall accuracy of a GPS/map-based guidance system.

```
$
Fix timestamp: 10/29/2022 12:58:14
Lat: 40.374923
Long: -82.469330
Fix qual: 1
# sats: 12, Alt: 263.8 m, Speed: 2.049 knots, Horizontal dilution: 4.38, Height geoid: -33.9 m
```

- Do not remove the SD Card without unplugging the device from power. Do not connect the GPS device to a computer for the safety of our software.
- The included SD Card (32GB) has enough storage to log approximately 5.5 years of GPS data. This is because each recording is only about 180 Bytes, logged every second.
- Try to especially take this device with you when going on trips down known transportation paths (i.e. trails, roads, waterways). This will help immensely with cross-referencing for accuracy.

Instructions:

1. Connect GPS Logger (USB-C) to battery pack (USB-A). Data collection will start automatically
2. Disconnect GPS Logger from power when not travelling for long periods of time like at night
3. Recharge the battery pack overnight with the included USB-C to USB-A cable. (The USB-C end attaches to the battery pack and the USB-A end attaches to a power supply/laptop when charging
4. Each week, please remove the SD Card (with power disconnected), connect the card to your computer, and send us the "gps.txt" text file. Then return the card back into the device. We will let you know when we have enough data. Thanks!

Appendix M

Overall BOM

Item	Unit Price	Quantity	Sub-Total
2 Sets for Raspberry Pi Camera	$ 23.99	1	$ 23.99
Matek H743-Wing V3 Flight Controller	$ 109.99	1	$ 109.99
MATEKSYS M8Q-5883 GPS Module	$ 35.99	1	$ 35.99
Matek Digital Airspeed Sensor ASPD-4525	$ 47.99	1	$ 47.99
X-UAV Mini Talon	$ 69.99	1	$ 69.99
Arducam 64MP Autofocus Camera	$ 59.99	1	$ 59.99
Caddx Ant FPV Camera	$ 20.99	1	$ 20.99
Green Infrared Filter	$ 19.99	1	$ 19.99
Flash Memory Card	$ 8.99	1	$ 8.99
Mini Portable Charger 5000mAh	$ 9.89	1	$ 9.89
Seeed Studio XIAO RP2040 Microcontroller	$ 9.49	1	$ 9.49
AiTrip 3PCS Micro SC Card Module	$ 5.99	1	$ 5.99
2Pack GPS Module	$ 17.99	1	$ 17.99
Humidity and Temperature Breakout Board	$ 10.95	1	$ 10.95
Foam Glue	$ 11.50	1	$ 11.50
RC Controller	$ 114.99	1	$ 114.99

Item	Unit Price	Quantity	Sub-Total
RC Receiver	$ 19.99	1	$ 19.99
FPV Monitor 5.8GHz	$ 106.59	1	$ 106.59
TrueRC D-Pole 2.4GHz MK II Antenna	$ 7.99	1	$ 7.99
Adafruit MCP9600 I2C Thermocouple Amplifier	$ 15.95	1	$ 15.95
DHT Electronics 2PCS coaxial Coax Adapter	$ 5.80	1	$ 5.80
OPTOLONG 1.25" UV/IR Cut Filter	$ 44.00	1	$ 44.00
Emax Pagoda 3B 5.8Ghz Stubby 30mm Antenna	$ 5.99	1	$ 5.99
Emax Pagoda 3B 5.8Ghz 50mm Antenna	$ 5.99	1	$ 5.99
Realacc Triple Feed Patch-1 5.8GHz Antenna	$ 16.99	1	$ 16.99
Arducam 64MP Autofocus Camera	$ 59.99	1	$ 59.99
Lemon Rx 6 Channel Full Range Receiver	$ 17.99	1	$ 17.99
SiK Telemetry Radio V3	$ 66.00	1	$ 66.00
Lumenier SM-25 25mW Micro VTX	$ 11.99	1	$ 11.99
Cobra C-2814/16 Brushless Motor	$ 37.99	1	$ 37.99
Uxcell RC Propellers CW 9x4.5 Inch 2-Vane	$ 13.49	1	$ 13.49
Cobra 33A ESC with 3A Switching BEC	$ 29.99	1	$ 29.99
Turnigy 5000mAh 4S 25C Lipo Pack	$ 41.97	1	$ 41.97

Item	Unit Price	Quantity	Sub-Total
LiPo Charger Lipo Battery Balance Charger RC	$ 56.99	1	$ 56.99
FLY RC 2 Pack XT90 Charging Cable XT90 to 4.0mm Banana Connector	$ 8.99	1	$ 8.99
XT90 Connector Male Female Adapter for Battery ESC	$ 10.99	1	$ 10.99
Thermocouple Amplifier MAX31855 breakout board	$ 14.95	1	$ 14.95
MicroSD card breakout board+	$ 7.50	1	$ 7.50
iSOUL [4 Pack] Screen Protector	$ 4.99	1	$ 4.99
Spektrum SRXL2 DSMX Serial Micro Receiver	$ 31.99	1	$ 31.99
Seamuing 6Pcs MG90S Micro Servo	$ 19.99	1	$ 19.99
Arducam 64MP Autofocus Camera	$ 59.99	1	$ 59.99
Spektrum Ws2000 Wireless USB RC Flight Simulator Dongle	$ 43.93	1	$ 43.93
SOQUARTZ 2GB COMPUTE MODULE W	$ 34.99	1	$ 34.99
SOQUARTZ "MODEL A" BASEBOARD	$ 24.99	1	$ 24.99
NEIKO 20713A Digital Tachometer	$ 24.99	1	$ 24.99
Lumenier SM-25	$ 11.99	1	$ 11.99

Item	Unit Price	Quantity	Sub-Total
25mW Micro VTX			
Medpride Disposable Scalpel Blades	$ 6.97	1	$ 6.97
SEN-13763	$ 10.95	2	$ 21.90
Excelta Tapered Ultra-Fine Tweezers	$ 6.54	1	$ 6.54
MATEKSYS M8Q-5883 GPS Module	$ 35.99	1	$ 35.99
Matek H743-Wing V3 Flight Controller	$ 109.99	1	$ 109.99
RC Controller	$ 114.99	1	$ 114.99
Spektrum SRXL2 DSMX Serial Micro Receiver	$ 31.99	1	$ 31.99
Cobra C-2814/16 Brushless Motor	$ 37.99	1	$ 37.99
Cobra 33A ESC with 3A Switching BEC	$ 29.99	1	$ 29.99
LiPo Charger Lipo Battery Balance Charger RC	$ 56.99	1	$ 56.99
1/8 inch Balsa Wood	$ 23.99	1	$ 23.99
FPV Monitor 5.8GHz	$ 106.59	1	$ 106.59
Emax Pagoda 3B 5.8Ghz 50mm Antenna	$ 5.99	2	$ 11.98
Realacc Triple Feed Patch-1 5.8GHz Antenna	$ 16.99	1	$ 16.99
Caddx Ant FPV Camera	$ 20.99	1	$ 20.99

Item	Unit Price	Quantity	Sub-Total
SiK Telemetry Radio V3	$ 58.99	1	$ 58.99
KARBXON Carbon Fiber Tube 12x8x1000	$ 30.94	1	$ 30.94
KARBXON Carbon Fiber Tube 8x6x1000	$ 28.99	1	$ 28.99
KARBXON Carbon Fiber Tube 6x4x1000	$ 16.96	1	$ 16.96
KARBXON Carbon Fiber Tube 5x3x1000	$ 15.96	1	$ 15.96
Turnigy Heavy Duty 5000mAh 4S 60C Lipo Pack	$ 49.99	1	$ 49.99
Arducam Multi Camera Adapter Module V2.2	$ 49.99	1	$ 49.99
UCTRONICS 0.96 Inch OLED Module	$ 6.99	1	$ 6.99
Flite Test Maker Foam Board	$ 49.99	1	$ 49.99
Anemometer Wind Speed Sensor	$ 44.95	1	$ 44.95
Crazepony 400mAh 2S 7.4V 30C LiPo Battery	$ 13.99	1	$ 13.99
DTTRA 20 Pairs 20 AWG JST Plug Connector 2 Pin	$ 4.99	1	$ 4.99
Wisesorb Silica Gel Packets	$ 8.49	1	$ 8.49
Ltvystore 10Pcs Adjustable Pushrod Connector	$ 11.99	1	$ 11.99
Mini Nano V3.0 ATmega328P	$ 12.99	1	$ 12.99
yueton Rc 1-8s	$ 5.49	1	$ 5.49

Item	Unit Price	Quantity	Sub-Total
Lipo Battery Tester			
Seamuing 6Pcs MG90S Micro Servo	**$ 19.99**	**1**	**$ 19.99**
BTF-LIGHTING 20 Pairs JST SM 3 Pin Connectors	**$ 9.99**	**1**	**$ 9.99**
X-UAV Mini Talon	**$ 69.99**	**1**	**$ 69.99**
SanDisk 256GB Ultra Fit USB 3.1	**$ 19.99**	**3**	**$ 59.97**
AINOPE USB Extension Cable 1.5FT	**$ 4.99**	**2**	**$ 9.98**
Arducam Multi Camera Adapter Module V2.2	**$ 49.99**	**1**	**$ 49.99**
Matek Digital Airspeed Sensor ASPD-4525	**$ 47.99**	**1**	**$ 47.99**
3pin FPV silicone cable for RunCam	**$ 2.99**	**1**	**$ 2.99**
5.8GHz FPV Triple Feed Patch Antenna	**$ 15.89**	**1**	**$ 15.89**
4 Pcs LED Aircraft Strobe Lights	**$ 12.99**	**1**	**$ 12.99**
Atnsinc 3Pcs CP2102 USB 2.0 to TTL 5Pin	**$ 9.99**	**1**	**$ 9.99**
Dorhea 2PCS 24.4inch Micro SD to SD Card Extension	**$ 10.99**	**1**	**$ 10.99**
Micro Center 32GB Class 10 Micro SDHC Flash Memory Card	**$ 8.99**	**1**	**$ 8.99**

Item	Unit Price	Quantity	Sub-Total
Uxcell RC Propellers CW 9x4.5 Inch 2-Vane	**$ 13.49**	**3**	**$ 40.47**
Ltvystore 10Pcs Adjustable Pushrod Connector	**$ 11.99**	**2**	**$ 23.98**
20 Pairs Mini Micro 6 Pin JST SH 1.0mm Cable	**$ 9.49**	**1**	**$ 9.49**
X-UAV Mini Talon	**$ 69.99**	**2**	**$ 139.98**
Matek Digital Airspeed Sensor ASPD-4525	**$ 47.99**	**1**	**$ 47.99**
Total		**105**	**$ 3,081.90**

Appendix N

Photo-taking Module Code

```
#!/usr/bin/python3
from picamera2 import Picamera2, Preview
import RPi.GPIO as gp
import time,os
from libcamera import controls
import time, subprocess
from datetime import datetime, timezone, timedelta
import gpsd
import stat

max_width = 4608
max_height = 2592



#change this to the flying height above ground
alt = 200

#DO NOT CHANGE
DISTANCE = 1.983*alt*(1-.68)

#disables warnings for mis-configurations
gp.setwarnings(False)

#allows the board state to be alterred
gp.setmode(gp.BOARD)

#sets the gpio pins for the Arducam Multiplexer as output
gp.setup(7, gp.OUT)
gp.setup(11, gp.OUT)
```

```
gp.setup(12, gp.OUT)

#Truth table for Multiplexer module
# Pin 7:    0    1    0    1
# Pin 11:    0    0    1    1
# Pin 12:    1    1    0    0
# Selection:    A    B    C    D

#for this program, "takeRGB" uses Camera A and "takeIR"
uses Camera C

#to use different cameras, use change the "i2c = " to the
following accordingly
    # Camera A:    i2cset -y 1 0x70 0x00 0x04
    # Camera B:    i2cset -y 1 0x70 0x00 0x05
    # Camera C:    i2cset -y 1 0x70 0x00 0x06
    # Camera D:    i2cset -y 1 0x70 0x00 0x07
def start():
     gpsd.connect()
     global picam2
     picam2 = Picamera2()
     camera_config =
picam2.create_still_configuration(main={"size":
(max_width,max_height)})
     picam2.set_controls({"AfMode":
controls.AfModeEnum.Continuous,"ExposureTime": 1000})
     #picam2.exposure_mode = 'short'
     #picam2.shutter_speed = 1000
     picam2.configure(camera_config)

def takeRGB():
     #take photo and initialize camera A
```

```
        i2c = "i2cset -y 1 0x70 0x00 0x04"
        os.system(i2c)
        gp.output(7, False)
        gp.output(11, False)
        gp.output(12, True)
        capture()

def takeIR():
        #take photo and initialize camera C
        i2c = "i2cset -y 1 0x70 0x00 0x06"
        os.system(i2c)
        gp.output(7, False)
        gp.output(11, True)
        gp.output(12, False)
        capture()

def getTimestamp():
        now = datetime.now(timezone(timedelta(hours=-4)))
        return now.strftime("%Y-%m-%d_%H:%M:%S.%f_%Z")

def capture():
        start()
        picam2.start()
        file =
find_first_connected_usb+str(getTimestamp())+".jpg"
        picam2.capture_file(file)

        picam2.close()

def on_change(camera, gps):
        global last_position
        if last_position is None:
```

```
        last_position = gps.position()
        return

        distance = gps.distance_to(last_position)
        if distance > DISTANCE:
        takeRGB()
        time.sleep(0.02)
        takeIR()
        time.sleep(0.02)
        last_position = gps.position()



def find_first_connected_usb():
        # Get the list of all connected USB devices
        devices = [os.path.join("/dev/disk/by-id", d) for d in
os.listdir("/dev/disk/by-id")]

        # Find the first USB device that is formatted in FAT32
        for device in devices:
        if os.path.isdir(device) and
os.stat(device).st_fsystem == stat.S_IFMT_VFAT:
                return device

        # No USB devices found
        return None

def check_and_format_usb(usb_path):
        # Check if the USB is formatted in FAT32
        if not os.path.isdir(usb_path):
        raise ValueError("USB path is not valid")
```

```
    file_system = os.stat(usb_path).st_fsystem
    if file_system != stat.S_IFMT_VFAT:
    # Format the USB to FAT32
    print("Formatting USB to FAT32...")
    os.system("mkdosfs -F 32 {}{}".format(usb_path,
os.sep))

    return usb_path

def delete_oldest_file(usb_path):
    # Get the list of files on the USB
    files = os.listdir(usb_path)

    # Get the size of the USB
    total_size = os.statvfs(usb_path).f_bsize *
os.statvfs(usb_path).f_blocks

    # Get the size of the oldest file
    oldest_file = min(files, key=lambda f:
os.stat(os.path.join(usb_path, f)).st_size)
    oldest_file_size = os.stat(os.path.join(usb_path,
oldest_file)).st_size

    # Delete the oldest file if the USB has less than
200MB left
    if total_size - oldest_file_size < 200 * 1024 * 1024:
    print("Deleting oldest file: {}".format(oldest_file))
    os.remove(os.path.join(usb_path, oldest_file))

def main():
    # Find the first connected USB device
    usb_path = find_first_connected_usb()
```

```
     # Check if the USB is formatted in FAT32 and format it
if it is not
     usb_path = check_and_format_usb(usb_path)
     while True:
     # Delete the oldest file on the drive if the drive has
less than 200MB left
     delete_oldest_file(usb_path)

     on_change() #takes the RGB and IR photo if GPS
location is > DISTANCE
     time.sleep(0.02) #allows time to clear the buffer
     #Camera.start()
     takeIR()
     time.sleep(0.02) #allows time to clear the buffer
     with gpsd.GPSD() as gps: #updates the current gps
coordinates
          gps.stream(gpsd.WATCH_ENABLE |
gpsd.WATCH_NEWSTYLE)
          gps.on_change = on_change



main()

if __name__ == "__main__":
     main()

     gp.output(7, False)
     gp.output(11, False)
     gp.output(12, True)
```