Cotton Crop Drone Analysis

Samuel Fabricant

Luis Sanchez

Justin Whitehead

Damilola Owolabi

FINAL REPORT

REVISION – 0

27, April 2022

Cotton Crop Drone Analysis

Samuel Fabricant

Luis Sanchez

Justin Whitehead

Damilola Owolabi

CONCEPT OF OPERATIONS

REVISION – 2

3, December 2021

CONCEPT OF OPERATIONS

FOR

Cotton Crop Drone Analysis

TEAM <66>

APPROVED BY:

_____

Project Leader Date

_____

Prof. Kalafatis Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	9/19/2021	Fabricant, Sam		Revision zero
1	9/29/2021	Fabricant, Sam		FSR/ICD/Validation Revision
2	12/3/2021	Fabricant, Sam		403 Final Report Revision
3	4/28/22	Sanchez, Luis		404 Final Report Revision

Table of Contents

List of Figures

2. Executive Summary

Our team has been tasked with automating the process of data collection in cotton crops. The analysis will include information regarding estimated crop yield and harvest date. To achieve the project deliverables, we will be using a drone that has an onboard Raspberry Pi Microcontroller and camera. The drone will take off on a programmed flight path, capturing images of cotton fields as large as ten acres. These images will then be analyzed with trained machine learning (ML) code that will generate the desired information regarding the cotton plot. This analysis will then be available to the user to interact with on a web application that we will develop. The practice of introducing drones into agriculture has been on an upward trend in recent years and already has been employed by Texas A&M AgriLife researchers. Drone imaging has allowed farmers to reduce their costs by improving spray accuracy and harvest date precision. Upon completion of our project, we expect to reduce cotton farmers manual workload involved in monitoring their crops and reduce their expenses in the process. A top-of-the-line agricultural drone can cost upwards of $4,000, but we plan to achieve this task with a budget of $400, excluding the cost of the drone itself.

3. Introduction

The services that drone imaging can provide are ever expanding in this time of technological advancement. Agricultural laborers and researchers are taking advantage of this trend and using drones to gather crop information, all without stepping outside of their residence. We will be creating a drone mounted device and programming machine learning algorithms to image and analyze cotton fields as well as a web application (UI) the user can interface with. We have the goal of saving farmers and researchers both time and money by automating this data collection process.

3.1. Background

The Cotton Crop Drone Analysis project is one aspect of an ongoing agricultural research study. A team of researchers have been using a commercial grade precision agriculture drone to collect images of a 100-acre cotton field. They have been using these images to investigate the ideal variables for modeling predicted yield and harvest date. This study has been in the process for four years and has proven to be successful. This project will be completed in parallel with the ongoing research team. Our team will be given access to the four years of cotton field data to train our Artificial Intelligence (AI) model, but not granted any information on their modelling algorithms. This project aims to improve upon the work they have already begun. To do so, we plan to train a more versatile and accurate AI model than the current model. With advantage of hindsight, we will have enough information to train and test different AI models in a relatively short time. Another improvement to the ongoing research is the user interface application that our team is developing. This user interface will provide a new level of analysis with the cotton crop study with interactive displays. Upon completion of this project, we aim to lower the entry barrier for precision agriculture by producing a device that farmers can employ without vast technical knowledge or financial burden.

3.2. Overview

The drone will be programmed to perform its imaging missions with the field of interest being specified with GPS coordinates by the user in the user interface web application (UI). When prompted by the user, the drone will take off and fly to the plot of interest, where it will begin capturing images of the cotton plot. Then the drone will return to its landing location and offload the images to the user's specific google cloud storage that is generated when the user creates an account on the UI. The image data will then be analyzed with our trained AI model, producing information about crop yield and harvest time frame. Lastly, the analysis will be uploaded to our application for the user to view and interact with. One of the main goals for this project is to develop a new and more accurate ML model for cotton crops. The team will have access to four years of cotton crop images to train the AI and produce an accurate model.

3.3. Referenced Documents and Standards

- TAMU AgriLife Administrative Services. Ethics & Compliance: Unmanned Aircraft Systems https://agrilifeas.tamu.edu/ethics-compliance/research-compliance/risk-and-compliance-unmanned-aircraft-systems/
- TAMU AgriLife Drone Usage. Digital agriculture connects the dots for crop improvement https://agrilifetoday.tamu.edu/2021/01/20/digital-agriculture-connects-dots-for-crop-improvement/

4. Operating Concept

4.1. Scope

The goal of this project is to accurately predict yield and harvest dates for 10-acre cotton fields using a drone, an enclosed Raspberry Pi microcontroller (MCU) secured to the drone, and machine learning algorithms. The drone and MCU system will be designed and tested to display proof of concept for the rest of the project deliverables. The systems should be able to receive inputs such as GPS coordinates from the user via the UI application. From there the drone will be able to determine the best method of traversing and imaging the field. Lastly, the MCU will offload the image payload to google cloud storage via WIFI transfer. This is the full extent of the drone/MCU requirements, the drone images will not be directly analyzed by the machine learning algorithm, as our project scope was altered in coordination with our sponsor on 11/3/2021. The image analysis will be managed by a third-party software to produce the machine learning input data. However, the remaining components; the UI application and AI model will be designed for long term use by the sponsor and AgriLife research team we are working in coordination with. The model will be best suited to analyze images from a precision agriculture drone, being that it was trained on data collected from such a drone. Additionally, the UI application will be developed to be used by both farmers and researchers, with the aim of creating a platform that is intuitive and insightful. Should the AI model or UI application succeed in improving the current techniques, they will be adopted to some degree by the research team. To summarize, we will pursue the most accurate AI model for cotton crop yield and harvest date, develop a UI application to view the model, and design a drone/MCU device to display proof of concept and validate communication between all the associated systems.

4.2. Operational Description and Constraints

The drone's flying and imaging will be controlled by an MCU. The drone will have to be able to take clear images in a variety of situations within safe flying conditions. The power supply for the drone will allow flight for 17 minutes, and the MCU power supply will provide power for about 1.5 hours. For the system to be useful, the trained machine learning algorithm will have to be able to predict yield and harvest date. The UI interface must display useful collected data such as predicted harvest date, predicted yield, the confidence the AI has on these predictions, and the corresponding graphs. The user will be able to enter GPS coordinates that encompass the area of interest upon creating their account on the UI. Once the user has sent this information to the drone from the UI, the

only other interactions required is the charging of the drone and MCU batteries and the user running the flight mission script on the MCU via secure shell (SSH). The charging of the drone battery is a constraint to fully automating the image capture process. It requires somebody to be in the general area of the drone / field of interest throughout the duration of the image capture process. The battery capacity also incurs a constraint for the field area to be imaged. The drone can only scan fields in which both the drone and MCU battery can span and retain enough charge to return to the landing pad and offload images. The final constraint is that the user must be in WIFI range of the CC Drone to SSH into the MCU and initiate the python script the flies the drone and captures images.

4.3. System Description

This project has been broken into six subsystems that will combine to achieve the overall project goals. The subsystems, as described in the Functional System Requirements document, are as follows. Refer to Table 1 in the Functional System Requirements document to view the team member responsible for each subsystem.

Drone Flight Control / Image Capture:

- Flight control programming
- GPS coordinate input retrieval
- MCU communication to Tarot Drone flight controller
- Ground control communication with the MCU
- MCU programming of image timing
- Mission startup checklist for weather and field size considerations

Drone Enclosure / Hardware:

- Connection of microcontroller to all components
- Power distribution and electrical connections to all components
- Enclosure design and mounting to drone

Data Management/Architecture:

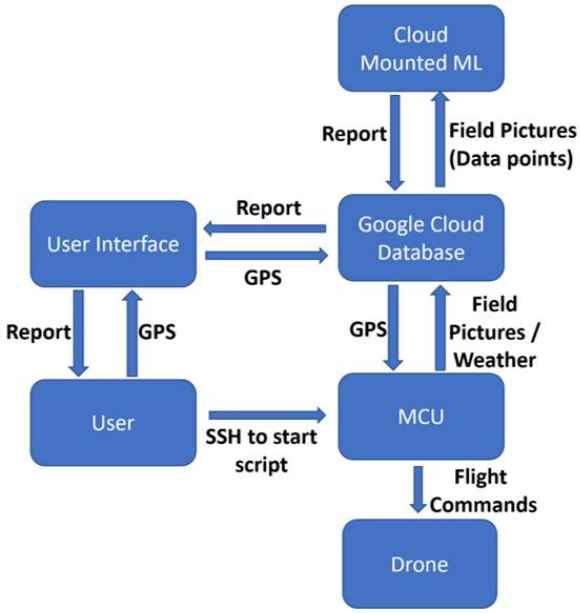
- Model data architecture
- Image data storage
- Weather data querying and storage
- Offloading of images to data management subsystem

UI Application Development:

- User access control (sending field GPS coordinates to google cloud storage)
- Display of study and scan results
- Verify User Information security
- UI application data repository

Model Development, Training, and Validation:

- Train ML on labeled cotton crop data to generate digital twin models.
- Predict harvest date and crop yield
- Generate report for display on UI



Member	Subsystem
Luis Sanchez	- Cloud Data Management / Architecture - MCU Enclosure & Power
Dami Owolabi	- User Interface
Justin Whitehead	- Machine Learning
Sam Fabricant	- Flight Control - Image Capture

Figure 1: Project Block Diagram

4.4. Modes of Operations

The only modes of operation for the CC Drone System are offline, online, and mapping mission. The CC Drone is in the offline mode when either the drone or MCU batteries are not connected and online when the batteries are connected and charged but the drone is not in flight. The CC Drone system is in the mapping mission mode when it is performing a flight mission to fly over the fields to capture images. This mode is fully automated once the user SSH's into the MCU to run the flight control python script. Once the script is initiated, the MCU will query the field GPS coordinates from the google cloud storage and fly over the field while capturing images before returning to the takeoff location and landing.

4.5. Users

The target users for this system are cotton farmers and researchers but with modifications to the programming, it could extend to anyone in the agriculture industry. The system will be fully functional once it reaches the user, but some slight training is required. The user will be required to first make an account in the UI application. Then they will need to read the support documentation, found in the UI, on how to charge the CC Drone and MCU batteries. Lastly, they will need to read the support documentation to secure shell (SSH) into the MCU, to initiate the flight mission python script.

4.6. Support

An in-app step-by-step guide will be provided in the user interface for the purpose of guiding the user on how to interface with the drone.

Forms of support include:

- Drone/MCU battery charging instructions.
- MCU SSH instructions.

5. Scenario(s)

5.1. Agriculture

The primary use of our device is to predict harvest date and yield of cotton plants across ten acres of cotton field. The primary benefactors of the project will be large scale cotton farmers. The client will use the drone to scan their cotton field, then the data gathered will be sent to the google cloud storage where the UI will extract and display in the web application. Extracted data will be used by the farmer to determine the predicted crop yield of either a specific area or of the entire field. This will save the user both time and labor.

5.2. Research

In this scenario, our extracted images will be used by researchers to help aid their research. The data can be used to analyze the effect of a cotton disease across various samples of cotton plant. It can also be used to analyze the growth or ripeness of a specific specie of cotton across different samples. The drone can be used to help monitor such samples and deliver data on the discovery.

5.3. Statistical Analysis

In this scenario, the device can be used by organizations like the National Cotton Council of America, to extract statistical data on cotton. Massive numbers of the device and its operators would be sent by the council to regions across the country. The device would be used to help analyze the amount or percentage of crops affected by a particular disease. Given its speed and automation, the device should be able to scan vast acres of land per day, with minimal human interaction. The device will be impactful on the speed and effectiveness of information extraction for cotton, in a positive light.

6. Analysis

6.1. Summary of Proposed Improvements

The device will be able to automate the process of predicting yield and harvest data of cotton crops within vast acres of cotton plantations. This will save the farmer time, money and labor required to do such labor-intensive process. Because of this, farmers will be able to spend that time and money on other tasks, like planting the seeds and harvesting the cotton balls. The effectiveness and accuracy of the device is leagues ahead of the traditional method. Hence, the number of errors in spraying pesticides, predicting harvest date, and expected yield will be reduced, which will assist in critical decision making by the company/individual. The machine learning report available on the UI application will allow the user to have data at their fingertips and allow them take immediate action when issues in the crop are detected.

6.2. Disadvantages and Limitations

Despite the numerous benefits from this project, there are disadvantages and limitations associated to this project.

6.2.1. Disadvantages

- Increase in job loss among workers in the field due to the fact the device automates the entire process. Hence no need for the jobs/labor associated with those process.
- Need for a technician to operate, maintain, and repair the device. As the device is not 100% automated. The most frequent need for human interaction will be charging the drone and the on-board battery for the MCU and running the flight script.

6.2.2. Limitations

- The device is only programmed to recognize cotton plants; hence it cannot be used on other crops like tomatoes, cocoa, etc.
- The drone is limited to the climate factors like rain, hail, snow, high winds, etc. As it cannot perform effectively in those kinds of conditions.
- The use of machine learning makes it difficult, even with extensive training, to be 100% accurate. If this 1% error results in a false negative, it would be near impossible to detect without having redundancies.

6.3. Alternatives

6.3.1. Using Solar Power as a Possible Power Source

Solar energy received from direct sunlight can be a feasible alternative for powering the drone. The customized solar cell will be attached onto the ‘roof’ of the drone. A rechargeable battery (e.g., the lithium-ion battery) will be attached to the drone through a frame that is made from welding and soldering materials together. This will eliminate the need for replacement of the battery cells. It will also make the project more environmentally friendly.

6.3.2. Using Agricultural expertise and experience.

Though this goes against the entire purpose of the project, one cannot deny the viability of this alternative as a solution. This is arguably the most time consuming and effort requiring solution. It also directly contrasts our proposed solution. As the heading states, it requires a cotton plant expert to spend time and effort in carefully looking at the roots, stem, leaves and fruits of each cotton plant in the plantation. It will require extensive knowledge on the plant. This alternative on its own sounds expensive. Hence, why it is not chosen.

6.3.3. Use of Satellite Imagery

Images captured by satellites are plausible ways to analyze data. Satellite images can be extracted using tools like the ESA Sentinel-2 Satellite Constellation’ which are

usually free and open source for anyone interested in them. This is a viable supplement to the drone data (images), but it requires extensive knowledge of satellites, which might be outside our scope of view.

6.3.4. Use of High Precision GPS Receiver

High precision GPS receiver can be used as a navigation tool for the drone. This is useful because of its improved accuracy and efficiency compared to the compass system already connected to the drone. Unfortunately, devices like that are ridiculously expensive, as buyers must be ready to shell out thousands of dollars, which is over our budget of $400. Though there are news of cheaper and more affordable GPS receivers that are still in the works. There is also little to no data out there on their accuracy and efficiency in being able to deliver precise data (position), which will be used to navigate the drone.

6.4. Impact

The impacts of our project will be the reduction of manual labor and time needed by farmers to monitor the cotton crops throughout the crop cycle. Employing our device will increase the farmers' yield and profits by informing the user to harvest at the optimal time. Implementing the cotton yield and ripeness detector in plantations and farms may improve the environmental conditions by phasing out routine herbicide, pesticide, and fungicide spraying. Lastly, this device may incur economic hardship on some greenhouse workers. By automating this data collection process, cotton laborers may be laid off to reduce the operating cost of growing the crops.

Cotton Crop Drone Analysis

Sam Fabricant

Justin Whitehead

Luis Sanchez

Damilola Owolabi

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 2

27 April 2022

FUNCTIONAL SYSTEM REQUIREMENTS

FOR

Cotton Crop Drone Analysis

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	9/28/2021	Fabricant, Sam		Revision zero
1	12/3/2021	Whitehead, Justin		Revision one, edited scope of project changes
2	4/27/2022	Fabricant, Sam		Revision for Final Report, updating requirements
3	4/29/2022	Sanchez, Luis		Revision three, proof reading and consistency in wording

Table of Contents

List of Tables

List of Figures

7. Introduction

7.1. Purpose and Scope

The CC Drone is a precision agriculture device, which serves to provide clients in depth analysis of their cotton crops, without having to enter the field. This device is intended to be employed by cotton farmers to better model their crops and generate analysis that is too tedious for humans to obtain themselves. It combines drone imaging with artificial intelligence to predict crop yield and harvest date. The project consists of a Tarot 650 Pro Drone with an MCU, and camera secured to the underbody to scan the cotton field. The client will set the CC Drone study parameters in an UI application. The UI application will also contain the report generated by each scan in the study. This document will outline the systems that comprise the CC Drone Project and describe the requirements for the overall system.

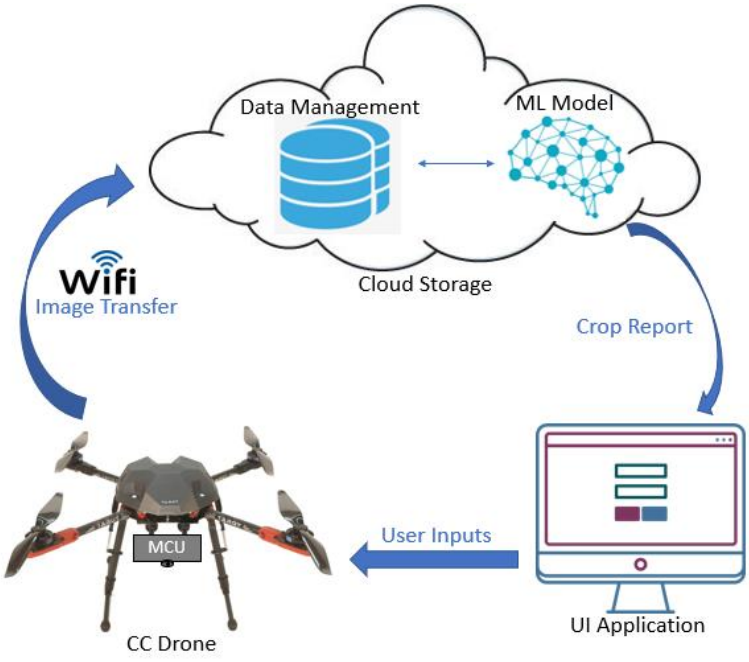


Figure 2. CC Drone Project Concept

Figure 1 shows the conceptual flow of the CC Drone Project. User inputs from the UI application will instruct the CC Drone on its scan/study parameters. The drone enters the field, images the cotton crop, returns to base, and offloads the images via WIFI transfer. The images are stored in the cloud where they are analyzed with a machine learning model. Then the report generated by the ML model is uploaded to be viewed on the UI application.

7.2. Responsibility and Change Authority

The team leader, Luis Sanchez, will be responsible for ensuring the requirements in this document are sufficiently met. In addition, Luis and the team sponsor, Dr. Nowka, have the authority to change the stated requirements when in consensus.

8. Applicable and Reference Documents

8.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title
FAA-2015-0150	June 28, 2016	PART 107 - SMALL UNMANNED AIRCRAFT SYSTEMS

8.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Reference/Citation Number	Revision/Release Date	Document Title
1	2009	Aircrew Operator's and Maintenance Manual: Tarot 650 - University of Nevada AirCTEMPs
2	2020	How to fly your Agricultural Drone to get accurate Plot Data
3	2021	Communicating with Raspberry Pi via MAVLink – ArduPilot Dev Team
4	Not Specified	DC-DC Converter Specification
5	2021	Average Humidity Levels for Texas – Current Results

8.3. Order of Precedence

In the event of a conflict between the text of this specification and a document cited herein, the text of this document takes precedence without exception.

All specifications, standards, exhibits, drawings, or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

9. Requirements

9.1. System Definition

The CC Drone project consists of using a drone, MCU, camera, machine learning algorithms, and a UI app to predict and display yield and harvest date of cotton crops. Cotton field images will be gathered and analyzed, extracting four data values to model future yield and harvest date. These data values of interest are canopy cover, excess green index, canopy height, and canopy volume of the cotton crop. The project is broken into the following five subsystems.

Drone Flight Control / Image Capture:

- Flight control programming
- GPS coordinate input retrieval
- MCU communication to Tarot Drone flight controller
- Ground control communication with the MCU
- MCU programming of image timing
- Mission startup checklist for weather and field size considerations

Drone Enclosure / Hardware:

- Connection of microcontroller to all components
- Power distribution and electrical connections to all components
- Enclosure design, printing, and mounting to drone

Data Management/Architecture:

- Model data architecture
- Image data storage
- Weather data querying and storage
- Offloading of images to data management subsystem

UI Application Development:

- User access control (sending field GPS coordinates to MCU)
- Display of study and scan results
- Interface to model and analysis
- UI application data repository

Model Development, Training, and Validation:

- Train ML on labeled cotton crop data to generate digital twin models.
- Predict harvest date and crop yield
- Generate report for display on UI app

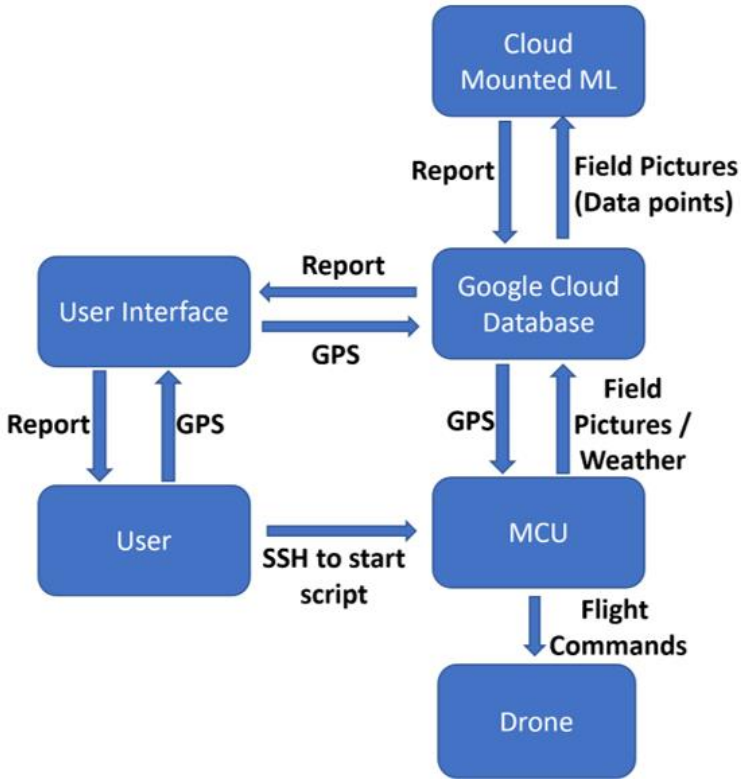
Note: Since this project is a proof of concept of analyzing crop data, the image analysis to pull data from the crop images was moved out of scope by our sponsor in November and any mention of it in this report should be assumed to be the use of third-party software.

*See Figure 1 and Figure 2 below for more information on how these systems interact. The ICD will further specify the information transfer processes between the subsystems.

Subsystem	Responsible Member
Drone Flight Control / Image Capture	Sam Fabricant
Drone Enclosure / Hardware	Luis Sanchez
UI Application Development	Damilola Owolabi
Model Development, Training, and Validation	Justin Whitehead
Data Management/Architecture	Luis Sanchez

Table 1: Member Subsystem Responsibility

The five subsystems are data management/architecture, machine learning model development, image capture, Image offloading, flight control, enclosure, and user interface. The block diagram below shows how the subsystems interact to create the full study report. Note that, training the ML model has been left out of this diagram since it is only used in the creation of the ML model, and will not be present in the final product.



Member	Subsystem
Luis Sanchez	- Cloud Data Management / Architecture - MCU Enclosure & Power
Dami Owolabi	- User Interface
Justin Whitehead	- Machine Learning
Sam Fabricant	- Flight Control - Image Capture

Figure 3. CC Drone Block Diagram

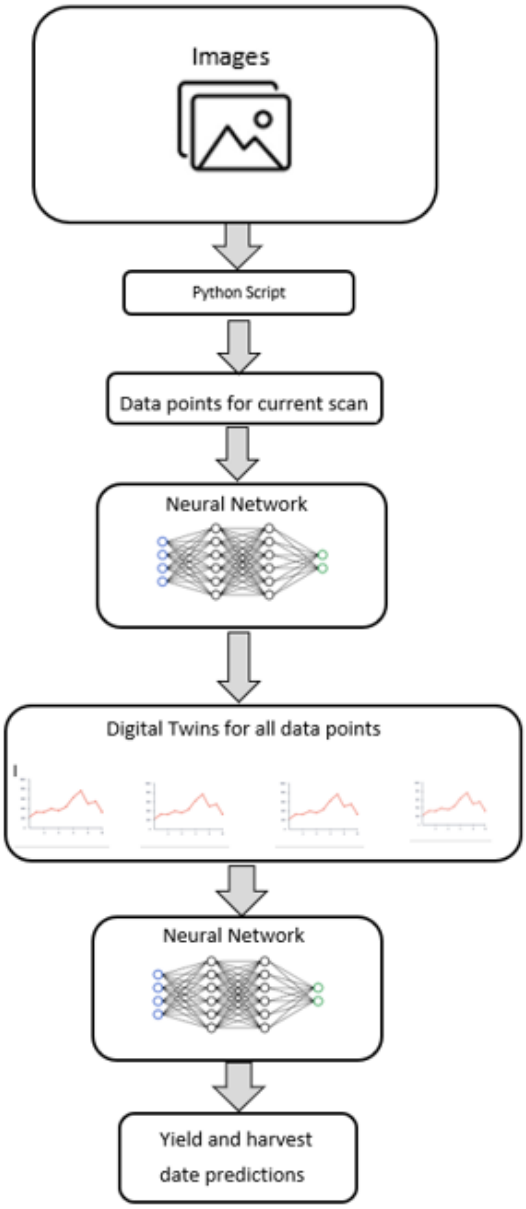


Figure 3. Machine learning block diagram

Figure 3 shows a simple block diagram of the steps the machine learning model uses to determine the final yield. The digital twins will be updated with every scan.

9.2. Characteristics

9.2.1. Functional / Performance Requirements.

9.2.1.1. Yield Prediction Accuracy

The CC drone should have a yield prediction accuracy of above 60% in the testing phase, and a harvest date prediction accuracy of 7 days.

> *Rationale: The original data used to train the ML model has inaccuracies stemming from errors is drone imaging and analysis of the Ortho-mosaic. Because of this the training data is noisy and makes it difficult to train a Machine model. Additionally, there are many factors such as crop gene, soil quality, etc. That may affect crop growth without showing in the data set. This leads to oversimplification of a complex problem which results in low accuracies.*

9.2.1.2. Scan Specifications

The CC Drone shall support a scan altitude from 50m(164.04ft) to 60m(196.85ft). The CC Drone will support a maximum scan flight speed of 3 m/s (31.32 mph). Operating within the ranges, the drone shall capture clear images.

> *Rationale: The CC Drone System utilizes a Tarot 650 Pro quadcopter as its vehicle. The drone operates with a functioning altitude up to 122 meters (400.26 ft) and maximum velocity of 15 m/s (33.55 mph), however for clear images, a slower speed of 3m/s (6.7 mph) is needed. The specified height and velocity will allow for imaging of fields up to ten acres on a fully charged battery.*

9.2.1.3. MCU File Query and Offload

The MCU connected to the drone, whilst supplied with WIFI, shall be able to query text files from the Cloud Storage database in less than five seconds. Additionally, the MCU shall be able to offload images and text files to the Cloud Storage database at a maximum rate of 3 seconds per file.

> *Rationale: To be efficient for the user, the MCU must be able connect to WIFI and query/offload the necessary files before/after it flies a data collection flight.*

9.2.1.4. Imaging Flight Programming Requirements

The flight command programming must be able to effectively takeoff, fly to waypoints, and land the drone while connected to the MCU via Rx/Tx serial communication wires. The flight path mapping algorithm ran on the MCU before flights must be able to generate a flight path that will efficiently allow imaging of at least 85% of the field at 50 meters of altitude. At 50 meters of altitude, the drone shall image fields at the rate of at least 0.5 acres / minute.

> *Rationale: To safely and image fields in a timely manner, the flight path algorithm, and drone flight command scripts must work to a high level of efficiency. It is important to effectively operate at 50 meters to allow imaging of fields as large as ten acres.*

9.2.2. Physical Characteristics

9.2.2.1. Mass

The weight of the CC Drone shall not exceed 7.3 lbs. due to the drone weighing 4.8 lbs. and its recommended payload is 2.5 lbs.

Rationale: This is the limitation of the Tarot 650 Pro drone specifications. See reference document one.

9.2.2.2. Volume Envelope

The volume envelope of the MCU including the necessary components within the enclosure shall not exceed a length of 6.7 inches, a width of 3.2 inches, and a height of 3.2 inches.

Rationale: This is a requirement based on the dimensions of the components being used in the enclosure and the physical dimensions of the Tarot drone underbody.

9.2.2.3. Mounting

The enclosure, which will contain the MCU, battery, camera, DC/DC buck converter, and the SD card, will be mounted on the underside of the CC Drone. The drone provides fifteen inches of ground clearance, which will be enough to mount the enclosure. Secure mounting of the enclosure is required for accurate imaging.

Rationale: Since the images will be taken in an aerial view, mounting the enclosure on the underside of the drone would be optimal. To achieve clear images, the enclosure and MCU components must be tightly secured to the drone.

9.2.3. Electrical Characteristics

9.2.3.1. Inputs

The presence or absence of any input signals in accordance with ICD specifications applied shall not incur damage to the CC Drone System or the operator.

Rationale: This is a requirement to ensure safe operation and preserve the life expectancy of the CC Drone.

9.2.3.2. Power Consumption

The maximum peak power of the enclosed system, MCU along with the components mentioned in section 3.2.2.3, shall not exceed fifteen watts.

Rationale: The DC/DC buck converter has a rating of 5V/3A which will provide ample power to the MCU, and the components connected within the enclosure system.

9.2.3.3. Input Voltage Level

The battery within the enclosed system will provide +14.8(+/- 15%) VDC which will be stepped down to 5 (+/- 1.5 %) VDC by a DC/DC buck-converter. The 5 VDC will be the input voltage for the MCU.

Rationale: Raspberry Pi runs on 5V/ 2A. When adding additional components and a desired runtime of around an hour, a large battery is desired. Since the battery being used delivers 14.8 VDC, a DC/DC buck converter is necessary to step down the voltage, deliver the power, and run time needed.

9.2.3.4. Input Noise and Ripple

The input noise and ripple voltage from the 5V/ 3A DC-DC buck converter may reach a 30mV peak, according to the data sheet. The steady-state voltage input along with this ripple from the converter will be fed into the MCU to power our enclosed system for the CC Drone.

Rationale: The specifications for the WG-12S0503 DC-DC converter lists the output ripple and noise to be 30mV.

9.2.3.5. External Commands

The CC Drone will receive external commands from the UI. External commands include appropriate GPS coordinates for the area of interest.

Rationale: The CC Drone will receive commands from the user and will modify its scans accordingly to the specifications of the user.

9.2.3.0. Battery Range

The CC Drone MCU battery will be able to power the onboard components for a maximum period 1.25 hours at peak power consumption. The Tarot drone onboard battery will provide power for up to 17 minutes of flight time.

Rational: The Drone MCU receives power from a 4500 milliamp-hour lithium polymer battery. According to the battery specifications, this is enough power to supply the enclosed electrical system for greater than 60 minutes.

9.2.4. Outputs

9.2.4.1. Images Output

The CC Drone MCU shall offload all images taken during a scan to the cloud-based data management system, where it will be stored. This will be achieved over short range (less than 10 feet) Wi-Fi transfer or with an ethernet cord that shares Wi-Fi from a laptop.

Rationale: The CC drone shall store all images taken for completeness and as data for future training sets and analysis. WIFI transfer is the most practical method for offloading a large image payload to be stored in the cloud.

9.2.4.2. Report Generation Output

The CC Drone Project shall use machine learning to process the images into digital twin models and predict the yield and harvest date. The digital twin graphs and predicted yield will be condensed into a report and directed through the data management sub-system where it will be accessible via the UI application.

Rationale: Provides the user with predictive graphs of data points with the final predicted yield and harvest date for their use.

9.2.5. Environmental Requirements/Flying Conditions

The CC Drone shall be designed to withstand and operate in the environmental conditions specified in the following section.

9.2.5.1. Pressure (Altitude)

The CC Drone System shall be designed to withstand and operate in air pressures of altitudes from sea level (0ft) to 5,000 ft. The system should be able to function efficiently in the weather conditions listed above.

Rationale: This is a requirement specified due to the location the customers will be using the system at (TX). The altitude in which cotton is grown in Texas ranges from 2,000 ft to 4,700 ft. We also considered the maximum height needed for a drone to capture an image without affecting the image quality, which is 60m(196.85ft), as well as a 100 ft threshold.

9.2.5.2. Thermal

The CC drone shall be designed to withstand and operate within the temperature range of 20-degree Fahrenheit to 115-degree Fahrenheit.

Rationale: This range includes the range of the region where the customers will be using the system. The temperature range in which the drone can withstand is between 14-degree Fahrenheit to 122-degree Fahrenheit. Which is within our determined operating range.

9.2.5.3. Humidity

The CC Drone shall be designed to withstand and operate in humidity levels ranging from 26% to 90%.

Rationale: This range depicts the average historical data on the weather conditions of the location where the system is designed for use (Texas).

9.2.5.4. Lighting

The CC drone system will be designed to operate under natural light conditions from the daily time schedule of 11am CST to 5pm CST.

Rationale: This range include the planned scheduled time the drone is set to be operation. This is also the set time the image can be captured without its quality being affected.

9.2.5.5. Windspeed

The CC Drone system shall be designed to withstand and operate in windspeed levels ranging from 0 mph to twenty-four mph.

Rationale: This range was decided off the max tilt of the Tarot 650 Sport Pro drone. High wind speeds would also reduce the image quality, which would impact the final report.

9.2.5.6. Precipitation

The CC Drone system will not operate in any level of precipitation.

Rationale: Imaging quality will be decreased in rainy environments. Additionally, the machine learning model will be trained to operate accurately with precipitation free images, due to the provided data set for training the ML. Lastly, the drone-MCU device is designed as a proof of concept, thus extensive measures to waterproof the drone-MCU device will not be taken.

9.2.6. Failure Propagation

9.2.6.1. Overfitting prevention

The data set shall contain 7,000 labeled data sets for the ML code to train on. This large data shall include a wide variety of cases which will allow the ML code to be prepared for a variety of scenarios.

Rationale: If the data set is too small, overfitting will occur. This will cause the ML code to give poor predictions due to uncertainty.

9.2.6.2. Noisy Images

The training data set will include slightly noisy images to prevent a loss of accuracy when presented with noisy images in the field. Thus, the CC Drone Image Capture and Machine Learning subsystems are required to function with slightly noisy images.

Rationale: In the field, blurry/noisy images are bound to occur. The machine learning model should be trained to recognize a noisy picture.

9.2.6.3. User Input Verification

The UI application will not allow unachievable instructions to be sent to the CC Drone MCU. Unachievable inputs include GPS coordinates that exceed the 100-acre maximum scan area, scan frequency of greater than one scan per day, and scan timing that do not follow the environmental requirements listed above.

Rationale: This is a requirement to prevent the CC Drone embarking on scans that exceed its functional requirements described in this document.

9.2.6.4. Battery Life Considerations

The CC Drone System shall not embark on a scan without ample battery life to complete the scan and return to base. To achieve this, the flight script runs a checklist before taking off, calculating the area of the field to scan, and the distance to the field from the takeoff location. The drone will not embark to scan fields that are greater than one thousand meters away or greater than ten acres in area. As a failsafe, once embarking on a mapping mission, the drone will only fly for a maximum of 15 minutes, before the script will command the drone to return to the takeoff location and land. This is to ensure that the drone does not lose power while in the air.

Rationale: This is a requirement to avoid the CC Drone running out of power while in the air or offloading images.

10. Support Requirements

The user will be provided with support documentation for the UI app. The support documentation will explain how the user can initiate a new study with parameters of their choosing. Additionally, the documentation will provide information on how the user can view scan / study results. There will also be a short tutorial on initiating the flight mission script on the MCU. Lastly, the documentation will include battery maintenance instructions and troubleshooting tips.

11.References Works Cited

1. https://ctemps.org/sites/ctemps.org/files/09_aircrew_operator_and_maintenance_manual_tarot650.pdf
2. https://delair.aero/agriculture/how-to-fly-your-drone-to-get-accurate-plot-data/
3. https://ardupilot.org/dev/docs/raspberry-pi-via-mavlink.html
4. https://cdn.sparkfun.com/assets/5/4/2/5/7/WG-12S0503-8V.pdf
5. https://www.currentresults.com/Weather/Texas/humidity-annual.php

Appendix A: Acronyms and Abbreviations

CC Drone	Cotton Capture Drone
UI	User Interface
MCU	Microcontroller Unit
AI	Artificial Intelligence
ML	Machine Learning
MPH	Miles Per Hour
GPS	Global Positioning System
CST	Central Standard Time
VDC	Volts Direct Current
DC	Direct Current
m/s	Meters Per Second
ft	Feet/Foot
mV	millivolt

Appendix B: Definition of Terms

CC Drone – The combination of the Tarot Drone, Microcontroller, camera, and any other components attached to the drone.

Scan – The process of the CC Drone traversing the field of interest, taking photos that completely map the field.

Study – A collection of scans that are used to model the field of interest.

Drone Enclosure – The apparatus that secures the hardware (MCU, Camera, Power Supply) in place, and to the drone.

Overfitting - When a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

Digital twin – A model that predicts future values of data points based on previous recorded values. Updates after every new data point is recorded.

MCU – Defined as Microcontroller Unit. In the CC Drone Project documentation, MCU refers to the microcontroller we are using, the Raspberry Pi model 3B, as well as the components that are connected to it.

Base – In context with the CC Drone System, base refers to the location that the drone departs to scan from, returns to after completion of a scan, and where it offloads images via WIFI transfer.

Ortho-mosaic – collection of separate images with overlaps that are fitted together to create a single image.

Cotton Crop Drone Analysis

Sam Fabricant

Luis Sanchez

Justin Whitehead

Damilola Owolabi

INTERFACE CONTROL DOCUMENT

REVISION – 1

3 December 2021

INTERFACE CONTROL DOCUMENT

FOR

Cotton Crop Drone Analysis

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher II, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	10/3/2021	Fabricant, Sam		Revision zero
1	12/3/2021	Sanchez, Luis		Revision one, edited scope changes
2	4/29/2022	Sanchez, Luis		Revision two, proof reading and consistency in wording

Table of Contents

List of Tables

List of Figures

No table of figures entries found.

1. Overview

The purpose of this document is to provide detail on how the MCU interfaces with the other components in the CC Drone System. Information will be provided on how the Raspberry Pi communicates with the camera, the Tarot Drone flight controller, and the UI application. Additionally, there will be information regarding the physical interface of the MCU enclosure and the Tarot Drone.

2. References and Definitions

2.1. References

IEEE 802.11 LAN/MAC/WIFI Standards
December 2020

MAVSec: Securing the MAVLink Protocol for Ardupilot/PX4 Unmanned Aerial Systems (arXiv:1905.00265v2)
4 May 2021

2.2. Definitions

mAh	Milliamp Hours
g	Grams
cm	Centimeters
MCU	Micro controller unit
V	volts
A	Amps
mm	millimeter
DC	Direct Current
CC Drone	Cotton Capture Drone (name of our system)
Li-Po	Lithium Polymer
UI	User Interface
SD	Standard Disk
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver-Transmitter
MAV	Micro Air Vehicle
RX	Receive
TX	Transmit
GB	Gigabyte
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ORM	Object-Relational Mapping
lb	Pounds

3. Physical Interface

3.1. Weight

Component	Weight	Number of Items	Total Weight
Tarot 650 Drone	2180 g	1	2180 g
Raspberry Pi 3 B	42 g	1	42 g
4500mAh Lithium-Polymer Battery	496 g	1	496 g
Pi Camera v2	3 g	1	3 g
16 GB micro-SD card	5.1 g		5.1 g
5V/3A DC/DC Buck Converter	35 g	1	35 g
Wi-Fi USB Adapter	45 g	1	45 g

Table 2: Component Weight

Table 1 shows the individual weights of all the components of the CC drone. The combined weight of the entire system is 2806.1g or 6.19 lbs.

3.2. Dimensions

Component	Length	Width	Height
Tarot 650 Drone	47 cm	30 cm	16 cm
Enclosed System	17 cm	8 cm	8 cm

Table 3: Component Dimensions

Table 2 shows the dimensions of the system. Since the enclosed system will be within the Tarot 650 Drone, the total length, width, and height of the system is 47cm, 30cm, and 16cm, respectively. The total volume of the system is 0.2645m^2 or 2.847ft^2.

3.3. Mounting Locations

The enclosed system will be mounted on the underside of the Tarot 650 drone. The CC Drone, Tarot 650, and enclosed system will fly its path with the camera pointing down towards the crops from above.

4. Thermal Interface

4.1. Heat Sink

The Raspberry Pi will have a heat sink inside of the enclosure attached to the integrated circuit. This will reduce the chance of efficiency reduction or overheating during periods of high computational stress on the device.

4.2. Ventilation

The enclosure for the Raspberry Pi will have ventilation holes in the sides. This will allow better circulation of air through the system as the CC Drone flies and cool the enclosed system.

5. Electrical Interface

5.1. Primary Input Power

The primary input power will be the 4500 mAh Li-Po battery. This will be used along with the 5V/3A DC-DC Buck converter to power the MCU, camera, and Wi-Fi adapter for image capture and connectivity to the cloud.

5.2. Battery Charging

After each flight, the CC Drone will dock, and the user must connect the Tarot 650 to its appropriate charger as well as the battery that resides inside the enclosed system. The enclosed system will have to be manually opened to connect the battery inside to the designated charger.

6. Communications / Device Interface Protocols

6.1. Internet Communications (Wi-Fi and ethernet)

The Raspberry Pi 3B microcontroller has a built-in Wi-Fi module using IEEE 802.11 g/b/n standards. This connection can be used to send the final report to the cloud, as well as receive user inputs from the webpage UI. During our flight tests, we are sharing Wi-Fi from a laptop to the Raspberry Pi via an ethernet cable for faster connectivity pre-flight, to run the fight scripts, and post flight, to upload the images taken during flight.

6.2. Host Device

The host device will be the Raspberry Pi 3 B. Everything within the enclosed system will be connected to the host device, directly and indirectly. The Li-Po battery will connect to the DC converter, which will be connected to the Host device. The camera will connect directly to the host device along with the micro-SD card and the Wi-Fi adapter.

6.3. Camera Interface

The Raspberry Pi camera V2 will connect to the Raspberry Pi with a 15-Way 150mm flat cable.

6.4. Device Peripheral Interface

The Raspberry Pi will be connected to the Tarot Drone flight controller through a serial connection to communicate using the MAVLink Protocol, namely the flight controllers UART TX and RX pins. Additionally, the Raspberry Pi / Flight Controller pair will be connected to the ground control application (Mission Planner or DroneKit) by running the MAVProxy application on the Raspberry Pi.

7. User Experience / User Interface

7.1. Application Webpage

The UI will be developed using Django. A python web framework that facilitates quick development of a clean and efficient web design. This tool allows us to create a project file in our file directory. The project file will be the location where we store frontend (like) and backend code for the webpage development. The specified database for this project will be the Google Cloud database. A service which allows for unlimited storage from their “cloud storage” and access to a vast array of API and services. Said server will be where data collected from the drone and micro controller unit are stored. Data from the server will be accessed using Django specific python backend functions and JavaScript files.

7.2. User Interface

The Application will be accessed online via a login page, which refers them to the webpage once his login information has been authenticated. The User also can register to the site if they do not have an account yet. The user inputs will be received by the User Interface, which collects the inputs in the form of strings. The inputs will be the GPS coordinate points for the areas to be scanned. The user also has the option of viewing their input GPS coordinates on a web generated map. User inputs will be sent as text files to the cloud database, where the MCU receives the file and converts the input parameters to commands for the drone. Data received from the MCU, and drone are sent to the designated cloud server. The websites extract the crop data from the database and displays them on the website. Extracted data consists of canopy height, canopy volume, green excess index, canopy cover, and the ML generated graphs. The website also has other functions like uploading csv files and redirecting users to other sites that shows weather and drone information.

Cotton Crop Drone Analysis

Sam Fabricant

Luis Sanchez

Justin Whitehead

Damilola Owolabi

MILESTONES SCHEDULE AND VALIDATION PLAN

REVISION – 0

3 October 2021

MILESTONES SCHEDULE AND VALIDATION PLAN

FOR

Cotton Crop Drone Analysis

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	10/3/2021	Fabricant, Sam		Revision zero
1	12/3/2021	Whitehead, Justin		Revision 1 - edited scope changes
2	4/29/2022	Sanchez, Luis		Revision two, proof reading and consistency in wording

1. Milestones Schedule

The following milestones schedule outlines the time frame and check points for completing subsystem goals for the Cotton Crop Drone Analysis Project.

Work to complete:	Complete By:	Responsible Member:	Status:
Connect Raspberry Pi and test essential functions.	10/8/2021	Sam Fabricant -Flight control/image capture	Completed
Test Raspberry pi camera by taking images	10/10/2021	Sam Fabricant -Flight control/image capture	Completed
Simulate drone flight with SITL simulation	10/12/2021	Sam Fabricant -Flight control/image capture	Completed
Finalize power systems circuit design, order components	10/15/2021	Luis Sanchez -Drone enclosure/power system	Completed
Create sample webpage (UI app) using Django	10/18/2021	Dami Owolabi -User Interface	Completed
Connect Raspberry pi to drone flight controller, verify communication. A) Test drone compass B) Test flight controller communication to mission planner C) Run test scripts on RPI to arm drone motors	10/18/2021	Sam Fabricant -Flight control/image capture	A) Completed B) Completed C) Completed
Finalize design of the Raspberry Pi mount/enclosure	10/18/2021	Luis Sanchez -Drone enclosure/power system	Completed
Test Raspberry Pi WIFI range.	10/18/2021	Sam Fabricant -Flight control/image capture	Completed
Finalize image offloading process	10/19/2021	Luis Sanchez -Drone enclosure/power system	Completed

from Raspberry Pi to cloud storage			
Generate digital twins from interpolated data	10/20/2021	Justin Whitehead -Machine Learning	Completed
Test image offloading process from Raspberry Pi to cloud storage	10/22/2021	Luis Sanchez + Sam Fabricant -Data management -Flight control/image capture	Completed
Verify method of querying report data from cloud storage to UI webpage	10/22/2021	Dami Owolabi -User Interface	Completed
Connect battery to buck converter and Raspberry Pi. Test battery life/power supply requirements.	10/22/2021	Luis Sanchez + Sam Fabricant -Data management -Flight control/image capture	Completed
Verify method of querying and storing weather data	10/28/2021	Luis Sanchez -Data management	Completed
Code mission startup check. Drone will not embark if: A) field is > 1000 meters away from takeoff location B) Field area is too large to scan in battery life (>10 acres)	10/28/2021	Sam Fabricant -Flight control/image capture	A) Completed B) Completed
Code no-fly conditions on Raspberry Pi. (Rain or Winds >= 10mph)	10/28/2021	Sam Fabricant -Flight control/image capture	Completed
Create machine learning models	10/29/2021	Justin Whitehead -Machine Learning	Completed
Finish Modeling of Mount/enclosure for 3D printing	11/1/2021	Luis Sanchez -Drone enclosure/power system	Completed
Train/assess all machine learning	11/8/2021	Justin Whitehead -Machine Learning	Completed

models and choose best model			
Finalize UI web model application	11/12/2021	Dami Owolabi -User Interface	Completed
Have ML upload final report to the cloud	11/12/2021	Justin Whitehead -Machine Learning	Completed
Test flight path algorithm with GPS coordinate inputs in SITL simulation. Scan rate goal: ½ acre per minute Field image covering goal: 95% A) 3inputs B) 4 inputs C) 5 inputs	11/15/2021	Sam Fabricant -Flight control/image capture	A) Completed B) Completed C) Completed
3D print enclosure and verify secure mounting to drone	11/15/2021	Luis Sanchez -Drone enclosure/power system	Completed
Test ML model accuracy of 60% in yield/harvest data prediction	11/15/2021	Justin Whitehead -Machine Learning	Not Completed
Test invalid user input	11/20/2021	Dami Owolabi -User Interface	Completed
Determine optimal flight altitude and velocity for clear images and 95% field imaging coverage. A1) thirty meters A2) forty meters A3) fifty meters B1) 2 m/s B2) 4 m/s	11/20/2021	Sam Fabricant -Flight control/image capture	ALL: Completed
Wires from MCU to drone flight controller (through the enclosure)	2//7/22	Luis Sanchez -Drone enclosure/power system	Completed
Finish UI app backend conversion to Cloud Run	2/11/22	Dami Owolabi -User Interface	Completed
Append image offload code to drone	2/7/22	Luis Sanchez	Completed

flight program. Verify no errors and successful offload to cloud storage. (Run on RPi)		-Drone enclosure/power system Sam Fabricant -Flight control/image capture	
Add code to query GPS coordinate text file from cloud storage to drone flight program. (Run on RPi)	2/7/22	Sam Fabricant -Flight control/image capture	Completed
Insert Machine Learning AI into the Cloud	2/21/22	Justin Whitehead -Machine Learning	Completed
Have the ML offload report data to the Cloud	2/25/22	Justin Whitehead -Machine Learning	Completed
The UI allows user to download ML report as a csv file	3/7/22	Dami Owolabi -User Interface	Completed
Display the ML report outputs on the UI	3/11/22	Justin Whitehead -Machine Learning	Completed
Display the ML data point graphs on the UI	3/11/22	Justin Whitehead -Machine Learning	Completed
Simple takeoff and land test flight with enclosed MCU secured to drone.	3/11/22	Sam Fabricant -Flight control/image capture Luis Sanchez -Drone enclosure/power system	Completed
Drone flight validating flight command functions: - Connection & Takeoff functions - Go to waypoint function - Wait to arrive and image function - Land using mode RTL	3/11/22	Sam Fabricant -Flight control/image capture Luis Sanchez -Drone enclosure/power system	Completed

Integrated flight with: - GPS coordinates queried from cloud storage. - Waypoints determined by mapping algorithm - Images captured while in flight. - Images offloaded to cloud.	4/20/22	Sam Fabricant -Flight control/image capture Luis Sanchez -Drone enclosure/power system	Completed

2. Validation Plan

The following validation plan outlines the tests that will be completed to validate the function requirements listed in the Functional System Requirements document. Note that all blocks in the following validation plan table are considered fully complete, unless specified with an ***incomplete*** label.

Paragraph	Category	Test Name	Success Criteria	Methodology	Status	Person Responsible
6.3	UI	User can make account with email and password	User can make account on UI website, and log in.	Test UI ability to use Django backend functionality to make and remember accounts.	Passed	Dami
6.3	UI	User input GPS coordinates	User can enter GPS coordinates of their field; UI will upload coordinates to cloud storage as .txt file	User logs in and enters GPS coordinates. Check if GPS text file appears under user account folder in cloud storage.	Passed	Dami
6.3	UI	UI web application global deployment	UI web application can be deployed globally for use on any laptop.	Log in to website from a different computer than the device that deployed the web app.	Passed	Dami
6.3	UI	UI account folder creation in cloud storage	UI will populate a folder for each new user in cloud storage, upload GPS.txt file after user enters it.	Make a new account enter GPS coordinates. Check if new account folder was created in cloud storage with GPS text file.	Passed	Dami

6.3	UI	UI cloud storage report query	UI can access information from ML reports located in cloud storage csv file	Use the download report button on the UI app to download a report from the 'example' google cloud storage account folder	Passed	Dami
6.3	UI	View Reports	User can view the crop report for the current and previous years within the UI	The GPS will be queried via Django templates. Queried data will be collected, and written to a .txt file, which will be uploaded too the team's Google Cloud Storage, which acts as the database.	Passed	Dami
6.3	UI	View Graphs	User can view graphs of the four data points within the UI	Use the view graphs button on the UI web page to view graphs from 'example/images' google cloud folder	Passed	Dami
6.3	UI	Duplicate Accounts	UI will not allow two accounts to have the same email address	Make an account with a duplicate email of an existing account, have site give an error	Passed	Dami
6.3	UI	Information Security	UI will only allow users to view report information generated from their account / field	Log in to website and check if reports from other accounts are viewable	Passed	Dami
3.3	Drone	SiK Telemetry	Can wirelessly read telemetry from drone on MP (GPS, Flight Mode)	Test in lab, wired serial connection from RPi to flight controller with Rx and Tx wires	Passed	Sam
3.3	Drone	Dronekit Connection Function	Connect RPi to Drone Flight controller with Dronekit-Python script	Test in lab, wired serial connection from RPi to flight controller with Rx and Tx wires	Passed	Sam
3.3	Drone	Dronekit Change Mode	Change Flight mode with Dronekit-Python Script	Test in lab, wired serial connection from RPi to flight controller with Rx and Tx wires	Passed	Sam
3.3	Drone	Dronekit Takeoff and land	Dronekit-Python script can effectively takeoff and land the drone, with no damage	Test at Lick-Creek Park, Dronekit-Python Script on RPi ran via SSH with serial connection to flight controller	Passed	Sam

3.3	Drone	Dronekit Simple_goto Function test	Dronekit-Python script can effectively takeoff and fly to a waypoint	Test at Lick-Creek Park, Dronekit-Python Script on RPi ran via SSH with serial connection to flight controller	Passed	Sam
3.3	Drone	Location based waypoint arrival function	Waypoint arrival function allows drone to arrive within three meters waypoint before stopping	Test at Lick-Creek Park, Dronekit-Python Script on RPi ran via SSH with serial connection to flight controller	Passed	Sam
3.3	Drone	Waypoint arrival taking pictures function	Waypoint arrival function takes images on path to waypoint, offset with less than the calculated image spacing	Test at Lick-Creek Park, Dronekit-Python Script on RPi ran via SSH with serial connection to flight controller	Passed	Sam
3.3	Drone	Flight with mapping algorithm waypoints, imaging	Drone will fly to the determined waypoints while taking images and land within 15 minutes	Test at Rellis Campus, Dronekit-Python Script on RPi ran via SSH with serial connection to flight controller, time on stopwatch	Passed	Sam
3.3	MCU	Mission Checklist - Field & Weather consideratio ns	Flight script will block takeoff when: - Field area > 8 acres - Distance to field > 250 meters - Weather query forecasts rain - Weather query forecasts windspeeds > 10 mph	Test mission startup checklist script for all cases on RPi. Verify weather using weather.com. Verify field area / distance considerations with google maps.	Passed	Sam
3.3	MCU	Image/Weat her Data Offload	Images and weather forecast is uploaded to the google cloud	Test on RPi, run designated script, check GCS if new files are uploaded in designated folder, check if files are deleted in RPi	Passed	Sam & Luis
4.3	MCU	Wi-Fi Sharing over Ethernet	RPi can access internet while connected to computer via ethernet	Test on RPi connected to monitor, and WIFI sharing from personal device via ethernet. Open canvas on browser to check connection.	Passed	Sam & Luis
4.3	MCU	Start Script then	Script will finish running even	Test on RPi connected to monitor, start script	Passed	Sam & Luis

		Remove Ethernet	after ethernet is removed	via SSH on RPi, remove ethernet during runtime, check if script finishes		
3.3	MCU	Query Weather Information	Hourly weather forecast is obtained and stored in RPi file	Test on RPi, WIFI sharing over from personal device via ethernet, query weather using weatherbit.io API, check if weather file is created. Verify weather using weather.com	Passed	Sam & Luis
3.3	MCU	Query Files from GCS	RPi script can query files from cloud storage while connected to WIFI via ethernet	Test on RPi, WIFI sharing from personal device via ethernet, run function to query GPS coordinates from cloud, check if coordinates are retrieved	Passed	Sam
3.3	MCU	Run Mapping Algorithm	Mapping Algorithm function can run and generate accurate waypoints on RPi. Mapping algorithm will generate waypoints to image >= 85% of field area.	Test on RPi, run mapping algorithm function from queried GPS coordinate file, check mapping algorithm results. Using waypoint plotting tool to validate that the waypoints cover >= 85% of the field area.	Passed	Sam
4.3	MCU	RPi Battery Supply Requirements (5V, 3A)	Buck converter feeds 5V and 3A to RPi	Test output of MCU LiPo battery/buck converter using voltmeter. Verify 3A,5V. Connect buck to RPi, check for proper functionality.	Passed	Luis
4.3	MCU	RPi Battery Life	Power RPi for flight time of 10-15 minutes plus additional 5 minutes for data/image upload	Supply RPi with battery from MCU LiPo battery, time battery life with stopwatch while running scripts on RPi	Passed	Luis
4.3	MCU	MCU/Enclosure Weight	Weight less than the max drone payload (3.5 lbs)	Weigh fully loaded enclosure on scale, record weight to ensure it is below 3.4 lbs	Passed	Sam & Luis

2.3	ML	Data interpolation	Raw data is interpolated to 170 data points for ML training	Test in colab, upload raw data then check that length of the graph changes to 170 after interpolation using SciPy library	Passed	Justin
2.3	ML	Twin Creation	Ten unique Twins are generated from previous data	Test in colab, interpolate twin data, then compared against itself to find ten twins, calculate similarity using similarity measures library	Passed	Justin
2.3	ML	Twin Matching	Raw data is matched to most similar twin	Test in colab, upload raw data check similarity between raw data and twins using similarity measures library, select best similarity	Passed	Justin
2.3	ML	Twin and raw data combined	Raw data and twin are combined into a single graph with 170 data points	Test in colab, upload raw data and its digital twin, ensure that no real data is lost after combining, ensure length is 170	Passed	Justin
2.3	ML	ML training	Ml model predicts data with 60% accuracy	Test in colab, test ML on data that was not used in training, ensure accuracy is above 60%	Not Passed	Justin
2.3	Cloud/VM	Offload Reports	Vm instance uploads reports to shared cloud	Test in cloud, run vm instance, upload file to shared cloud, ensure that file is uploaded	Passed	Justin
2.3	Cloud/VM	Query ML outputs	ML predictions can be pulled from the cloud	Test locally, run a script to pull predictions from the cloud, test that predictions match with colab ML	Passed	Justin
2.3	Cloud/VM	Get input data	cloud vm instance can check in cloud for new data	Test in cloud, run vm instance and pull data from cloud to vm instance, check that file was pulled	Passed	Justin
2.3	Cloud/VM	Vm runs on start	vm instances runs data process code when it boots	Test in cloud, reboot vm instance, ensure data is processed and report generated automatically	Passed	Justin
2.3	Cloud/VM	No data for user	vm instance ignores users with no new data in cloud	Test in cloud, run vm with no data in a user's folder, ensure code does not exit	Passed	Justin

				and continues to check other users		
2.3	Cloud/VM	Not all data processed	if vm instance shuts down before processing all data, on reboot, it does not re-process the same users' data	Test in cloud, shutdown vm instance in the middle of processing data, reboot vm, ensure report is not reuploaded	Passed	Justin
2.3	Cloud/VM	Scheduling Data process	vm instance is run at scheduled hours (currently 1-10am)	Test in cloud, schedule vm using google scheduler, check logs to see that vm instance ran from 1-10am, and processed all data	Passed	Justin

Appendix A: Definitions

SITL	Software in the loop
UI	User interface
ML	Machine learning
DC	Direct Current
GPS	Global Positioning System
MCU	Microcontroller Unit
CC Drone	Cotton Capture Drone

Cotton Crop Drone Analysis

Sam Fabricant
Luis Sanchez
Justin Whitehead
Damilola Owolabi

SUBSYSTEM REPORT

REVISION – 1

4 April 2022

SUBSYSTEM REPORT

FOR

Cotton Crop Drone Analysis

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	12/03/21	Fabricant, Sam		Revision zero
1	4/27/2022	Fabricant, Sam		Revision one, updates after integration and validation testing.
2	4/29/2022	Sanchez, Luis		Revision two, proof reading and consistency in wording

Table of Contents

1. Introduction

The overall project is down into five different subsystems. The subsystems are machine learning, flight control / image capture, drone enclosure / hardware, data management / architecture, and lastly the user interface subsystem. The remainder of this section will provide details on each of the subsystems, the progress that has been made on them, and an evaluation to the completeness of the subsystems with respect to the validation and execution plan for the project.

Since each of the subsystems mentioned above are validated to be working correctly and fulfill all requirements, there is a clear path integrate the complete system specified in the ConOps, FSR, and ICD documents.

2. Machine Learning Subsystem Report

2.1. Subsystem Introduction

In this subsystem, data collected from the drone images will be used to train a machine learning model. This model will then be used to look at future data to predict the yield and harvest date of crops to help the user made critical decisions about their field. The team member, Justin Whitehead, is responsible for this subsystem.

2.2. Subsystem Details

The first step of this subsystem is the clean the training data. These training data sets come from a field in Corpus Christi that has been imaged for three years, however, in 2019 the only data points collected were Canopy coverage and Excess green index, so that year was removed. With the 2020 and 2021 data, I first removed the outliers from the data sets by removing and graph that, at any point, was over three times the standard deviation. This removed about 25% of the usable data. As you can see in Figure 1, there is a lot of deviation in the data, but there are also errors in the data as well. In Figure 1, there are a few plots that spike downward and upward somewhat randomly. For canopy coverage, the graph should only increase until the farmers spray the crop to prepare for harvest. When the crops are prepared for harvest the leaves wilt and the canopy coverage decreases. Therefore, any graph that decreases before an appropriate harvest date is most likely an error. Removing the outliers helped this problem significantly, however it is difficult to tell if other errors persisted in the data that could confuse the model.

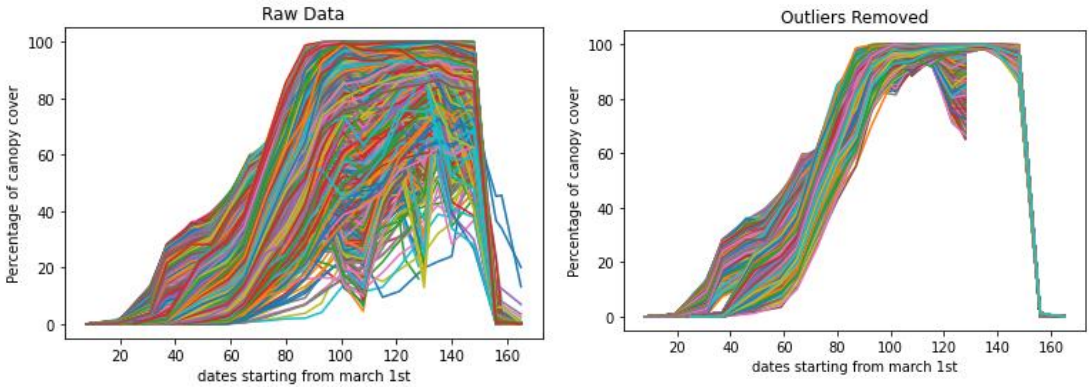


Figure 1 & 2: Figure 1 shows the raw Canopy Cover data for 2020 and 2021 and Figure 2 shows the same values after the outliers are removed

The next step for cleaning the data was to interpolate the 2020 and 2021 values. Interpolating the data ensures that I have a value for every day. This simplifies the machine learning since all inputs will be the same size. In Figure 2 above, you can see

that the 2020 data stops before the 2021 data. To have both data sets be of equal length, I set the 2020 data to hold its final value until the end of the 2021 data.

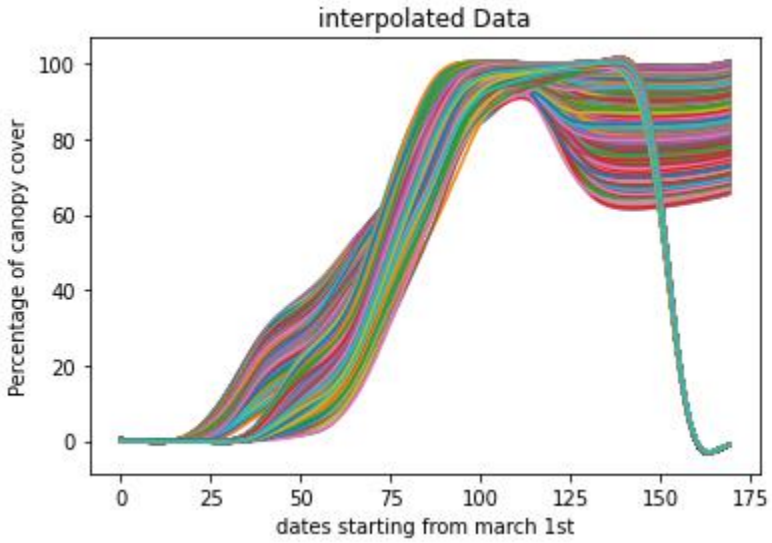


Figure 3: Interpolated data for 2020 and 2021 Canopy Coverage

After all the data was interpolated, I sorted each data set by similarity. For this I compared the first graph to each graph after using dynamic time warping (DTW), this type of comparison was used, since comparisons like area under curves does not consider when the data starts rising. For example, for most other comparisons the graphs *x* and *x+1* would be considered identical, since it is the same graph, only time shifted. Therefore, DTW was used since it considered whether the data was left or right shifted. Once the data was sorted by similarity, it was easy to select any number of dissimilar digital twins by selecting the first and last graph and ensuring that any graphs selected in between are equidistant from each other. The next step was to decide how many digital twins to use for comparison. Originally, I used one hundred twins. However, the data became overfitted as seen by the figure below. The figure shows one hundred digital twins matched to five hundred data points. As you can see, there are only seven distinct individual graphs. Because of this I lowered the number of digital twins to ten shown in Figure 5.

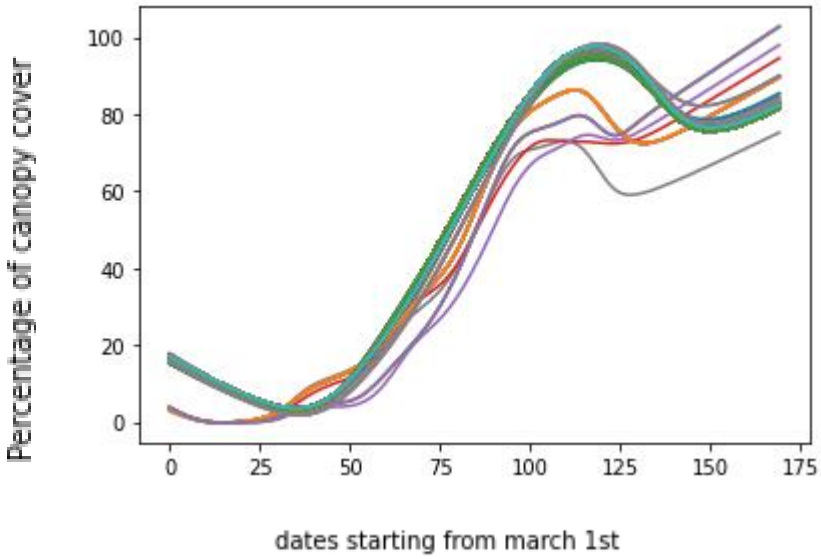


Figure 4: Shows the graphs selected from one hundred digital twins for five hundred Canopy Coverage data points

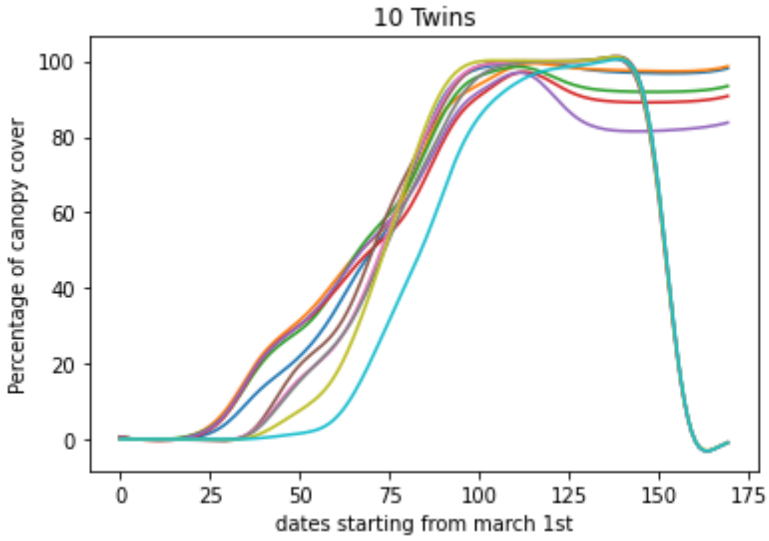


Figure 5: Shows the ten digital twins selected for Canopy Coverage

After the digital twins are selected, I built a training data set using known dates. For the training data set, the known dates are compared to the digital twins and matched, the known dates are then concatenated with the digital twin. The missing data points between the know dates are interpolated linearly, so that the array is the same size for the machine learning model. The figures below show the digital twins for canopy coverage after 5 scans, and ten scans, respectively. Note that the number of scans does not correspond to the number of days. This is because in the original drone flights were spaced out to twice a week, then weather and maintenance caused the drone to

have a non-uniform scan schedule. Therefore, the data in-between flights were interpolated, to ensure equal size of all data sets.

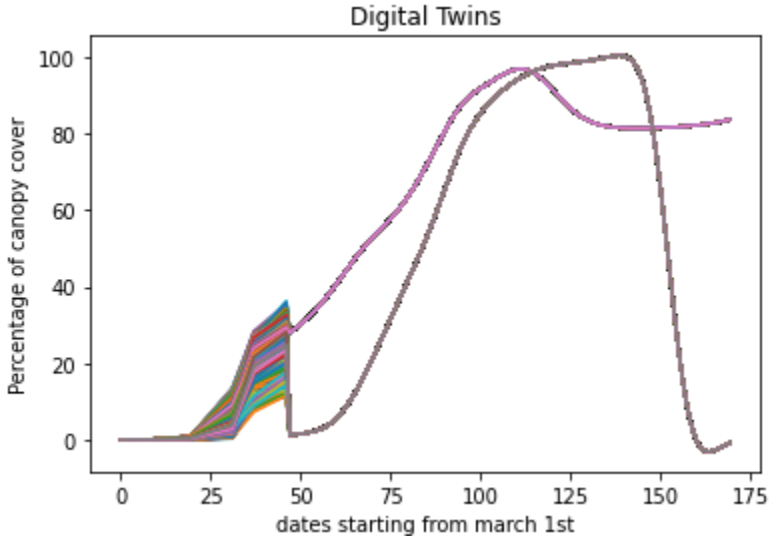


Figure 6: Digital twins matched after 5 scans.

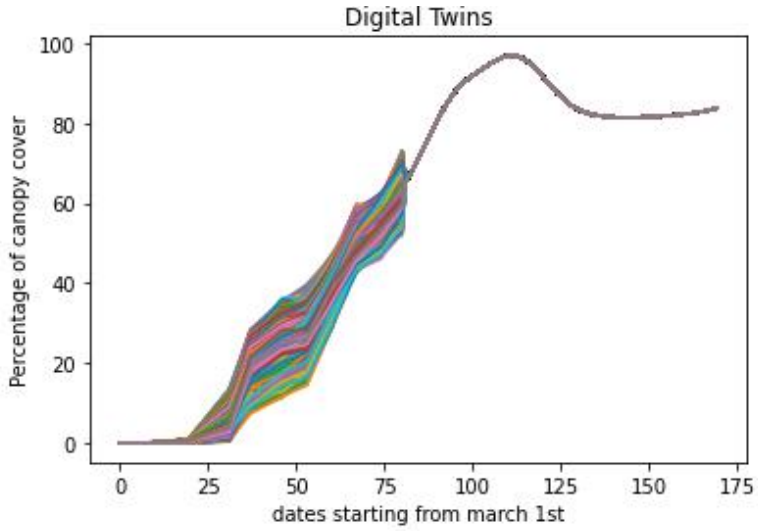


Figure 7: Digital twins matched after 10 scans.

After this the yield data had to be formatted for proper training. The maximum yield in the historical data is 115 and the minimum is zero. The data was then divided into ten buckets. Between 0 and 10, between 10 and 20, between 20 and 30 and so on until the final bucket was any value above 90. Many other formats were used, such as splitting the yield into five buckets evenly spaced, and five buckets divided by standard deviation, where the first bucket was less than two standard deviations, the next bucket was less than one standard deviation and so on. While these formats had higher percentage accuracies (around 65%), they merely predicted all the data the same way and used the fact that around 65% of the data was in one bucket. By splitting the yield

into ten buckets, the accuracy was only 35%, but the model was giving different yields and exceeded random choice. The harvest date is then predicted from the excess green index values. The historical data does not have harvest dates, so our sponsor told us that a way to predict harvest dates was to take the max value for the excess green index and then add around 1 month. This is the method I used to predict the harvest date. The last step of the model is to export the predicted data to the google drive. Currently the report is a simple csv file that gives the index of the field, yield, and predicted harvest dates. An example of this simple report is shown in figure 8 below. In 404, the user interface will read the csv file to present it in a more intuitive way.

3821	38g - 48g with a confidence of 36%	3-Aug
3822	38g - 48g with a confidence of 36%	11-Aug
3823	38g - 48g with a confidence of 35%	3-Aug
3824	38g - 48g with a confidence of 37%	28-Jul
3825	38g - 48g with a confidence of 27%	28-Jul
3826	38g - 48g with a confidence of 21%	28-Jul
3827	8g - 18g with a confidence of 46%	28-Jul
3828	8g - 18g with a confidence of 43%	28-Jul
3829	8g - 18g with a confidence of 48%	28-Jul
3830	8g - 18g with a confidence of 46%	28-Jul
3831	8g - 18g with a confidence of 50%	28-Jul
3832	8g - 18g with a confidence of 49%	28-Jul
3833	8g - 18g with a confidence of 50%	28-Jul
3834	8g - 18g with a confidence of 48%	28-Jul
3835	8g - 18g with a confidence of 49%	28-Jul
3836	8g - 18g with a confidence of 48%	28-Jul
3837	8g - 18g with a confidence of 44%	28-Jul
3838	8g - 18g with a confidence of 43%	28-Jul
3839	8g - 18g with a confidence of 46%	28-Jul

Figure 8: Example of output csv file

With further testing on different machine learning models, it was found that the random forest model gave the best results. The same digital twins that were used to train the dense neural network model were also used on a k-means clustering model, Logistical regression model, and a random forest model. For the K-means model, the elbow method was used to determine there were five clusters of data. The issues with these clusters are that they overlapped and were not distinctive, shown in figure 9. The K-means clustering model had an accuracy of 99%. However, the buckets did not have anything to do with the actual yield values as shown in figure 10. The Logistical Regression Model proved no better than the dense neural network, with an accuracy of only 35%. The random forest model proved to give the best accuracy at 43%.

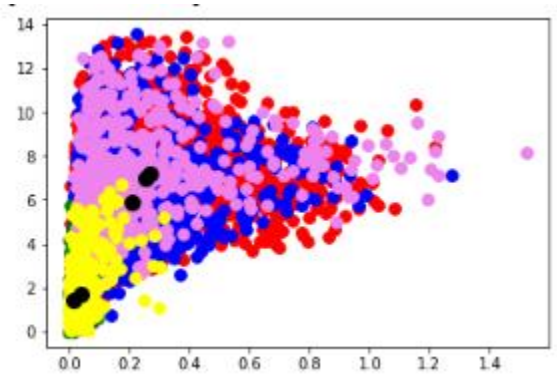


Figure 9: K-clusters

Max value	Min value	Average value
83	29	51.7
100	8	48.8
73	25	48.9
80	8	51.5
118	28	50.2

Figure 10: Table of K-means yield distribution

```
4612 examples in training, 1960 examples for testing.
/usr/local/lib/python3.7/dist-packages/tensorflow_decision_forests/keras/core.py:2036: Futurel
  features_dataframe = dataframe.drop(label, 1)
Use /tmp/tmpsmrpkhns as temporary training directory
2/2 [==============================] - 3s 412ms/step - loss: 0.0000e+00 - accuracy: 0.4357
```

Figure 11: Random Forest ML accuracy

The next steps of the Machine Learning subsystem were to create a way to automatically process incoming user data to predict yield and harvest date and send a report, without any user input besides uploading a csv containing the crop data. To do this, first the ML was mounted to a google cloud account. Then a virtual machine was programmed to start up at 1am and close at 10am every day. A code was then uploaded to the virtual machine that ran on startup and processed all available data in the ML cloud. Reports generated were then uploaded to the data management cloud.

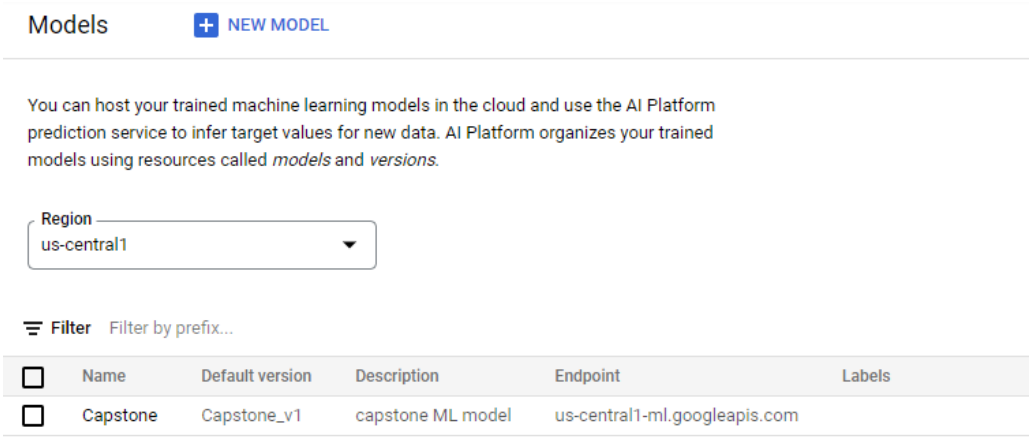


Figure 12: ML model mounted in cloud

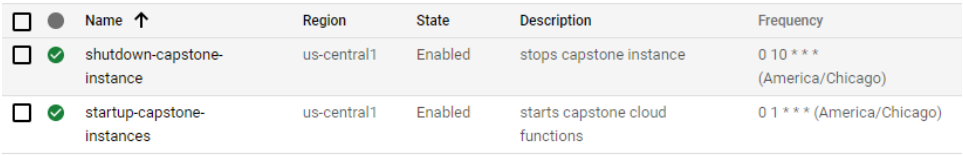
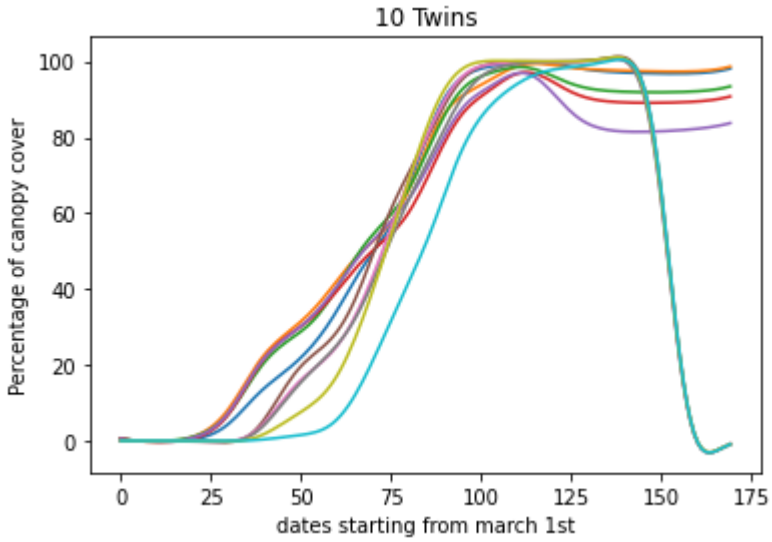
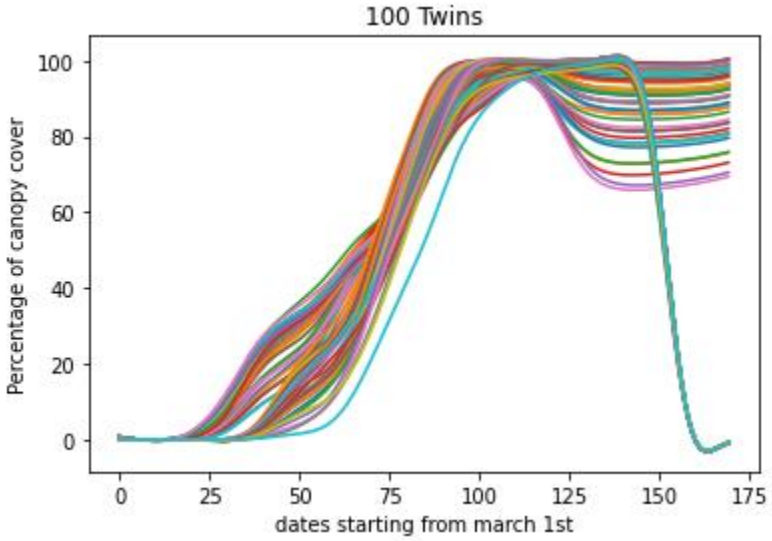


Figure 13: Scheduler that starts and stops VM

2.3. Subsystem Validation

The first validation for my subsystem is to pull the training data from the cloud. Since pulling data from the google drive is like pulling data from the cloud using a virtual machine, and we are trying to save cloud credits for 404, I instead pulled data from the google drive. This function is done with the function *get_data()* in the Twins_generation code. The output from this function can be seen with the CC data in Figure 1. The next step of my validation plan was to ensure that any number of digital twins can be generated from the historical data. This was needed since it would ease the process of assessing the model with varying digital twins. The figures below show the different number of digital twins generated for canopy coverage; the same method is reused on the other data points.


10 Twins
Percentage of canopy cover
dates starting from march 1st
0
20
40
60
80
100
0
25
50
75
100
125
150
175


100 Twins
Percentage of canopy cover
dates starting from march 1st
0
20
40
60
80
100
0
25
50
75
100
125
150
175

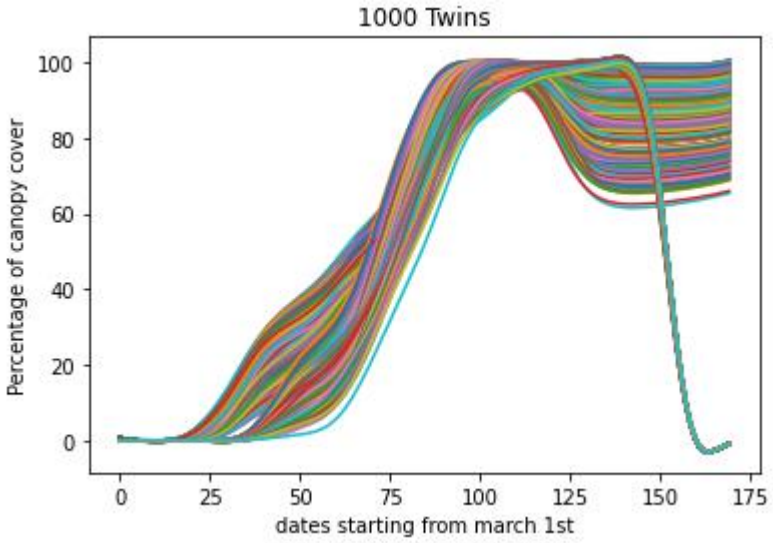


Figure 14: shows 10,100, and one thousand digital twins for Canopy Coverage

The next step of my validation plan was to match each plot to a digital twin. This is done with the dynamic time warping mentioned in the subsystem details and can be seen in figures 6 and 7. The next step of my validation plan was to create a model that can predict the yield and harvest date up to 60% accuracy. Originally this was planned to be 90% accuracy, however due to the errors in the training data, and that using the four data points, excess green index, canopy coverage, crop height, and crop volume, might be an oversimplification of a complex problem. Many other factors could lead to the yield of the crop, such as rain, sunshine, crop gene, soil quality, and more. That is why the accuracy was lowered to 60%. Unfortunately, 60% accuracy was not achieved most likely because of the above reasons.

For the cloud mounted machine learning model, it had to automatically scan for data, process that data and generate a report, then upload the report to the data management cloud. It also could not process the same data twice, crash if there was an error on a user uploaded data file, or if there was no data for the user. For the automatic scanning, I created a virtual machine in the cloud that ran a script on startup. This script checked the cloud for any user that had a datafile in their folder. If they did, the cloud would scan the data. If there were any errors, such as improper name of file or different lengths for the data points, the script would delete the incorrect file, notify the user that the file was incorrect, then move on to the next user. Otherwise, the data would be processed, and a report would be generated. Graphs would also be generated of the four data points for each plot. If the script successfully uploaded the report and graphs to the data management cloud, it would delete the data file to prevent it from processing old data. This process takes approximately 8 to 9 hours for the 4000-plot field we were using.

```
jrwhitehead555@capstone-1:~$ python3 python_code/Capstone_cloud_code_one_instance.py
404 No such object: user_ml_data/justin/datapoints_2022.csv (GET https://www.googleapis.com/storage/v1/b/user_ml_data/o/justin%2Fdatapoints_2022.csv?projection=noAcl)
list index out of range
request successful for owolabidamilola
404 No such object: user_ml_data/owolabidamilola@tamu.edu/datapoints_2022.csv (GET https://www.googleapis.com/storage/v1/b/user_ml_data/o/owolabidamilola%40tamu.edu%2Fdatapoints_2022.csv?projection=noAcl)
404 No such object: user_ml_data/sam/datapoints_2022.csv (GET https://www.googleapis.com/storage/v1/b/user_ml_data/o/sam%2Fdatapoints_2022.csv?projection=noAcl)
404 No such object: user_ml_data/samf/datapoints_2022.csv (GET https://www.googleapis.com/storage/v1/b/user_ml_data/o/samf%2Fdatapoints_2022.csv?projection=noAcl)
404 No such object: user_ml_data/samlfab18/datapoints_2022.csv (GET https://www.googleapis.com/storage/v1/b/user_ml_data/o/samlfab18%2Fdatapoints_2022.csv?projection=noAcl)
404 No such object: user_ml_data/test20/datapoints_2022.csv (GET https://www.googleapis.com/storage/v1/b/user_ml_data/o/test20%2Fdatapoints_2022.csv?projection=noAcl)
```

Figure 15: shows that the code will not crash when there is incorrect or missing data

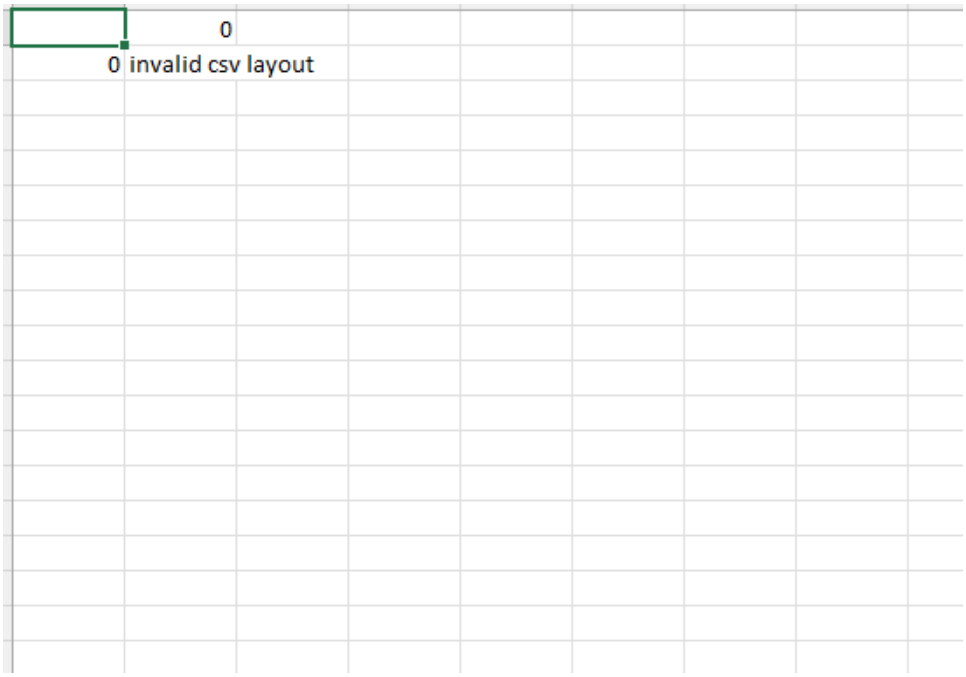


Figure 16: output of a file that had incorrect data

2.4. Subsystem Conclusion

In conclusion, the machine learning subsystem can currently pull the training data from the google drive and process it into digital twins for the model. New data can then be matched to a digital twin and use the digital twin selected to predict future values. The new combination of known data and digital twin is then sent to the model for training. The trained model is mounted in a google cloud. The cloud can automatically process data uploaded by the user and generate a report. The cloud also has error handling to prevent crashes. The only issue with this subsystem is that the ML accuracy did not meet the 60% requirement. Any code referenced in this section can be found on this public GitHub https://github.com/jrwhitehead555/Model-code

3. Flight Control and Image Capture Subsystem Report

3.1. Subsystem Introduction

This subsystem is responsible for first, reading in GPS coordinates that specify the field, running a pre-flight checklist to make sure the weather conditions permit a safe flight, then determining a flight plan to traverse the entire field, and lastly, commanding the drone to execute the flight path, while capturing images of the field before landing. Upon landing, the images captured and additional text files (weather at time of the flight, user log, and GPS coordinate location of each image) will be offloaded to the cloud storage database. This subsystem has been achieved through software and is nearly 100% automated for the user. The user will be required to enter the GPS coordinates that define their field in the user interface. From there on, the only actions they are required to take is to initiate the drone flight script by SSH (secure shell login) into the MCU, enter their account username when prompted. Lastly, if in an area without available WIFI networks, once the drone lands, they can provide WIFI over ethernet to allow a prompt offload of the images. This subsystem was achieved throughout the semester using drone flight simulations, utilizing Dronekit-SITL (software in the loop) and Mission Planner to view the simulated drone as it executes a flight mission. After successful flight simulations, we ran the fully integrated flight script on the Tarot Drone, under the accompany of FAA certified TA Swarnabha Roy. This subsystem report will first detail and validate all the individual components of the overall integrated flight script, and then validate the operation of the integrated script. (*demo.py* on the GitHub branch below). The team member, Sam Fabricant, is responsible for this subsystem. The code for this subsystem is located in the public GitHub branch; https://github.com/DamilolaOwolabi/ECEN-403-GROUP-66/tree/master/drone_flight_code_Sam .

3.2. Subsystem Details

The block diagram below portrays the overall flow of the flight control and image capture subsystem. The entirety of this process occurs in the integrated script *demo.py*, but for clarity, this section will reference the scripts that were eventually integrated into *demo.py.*

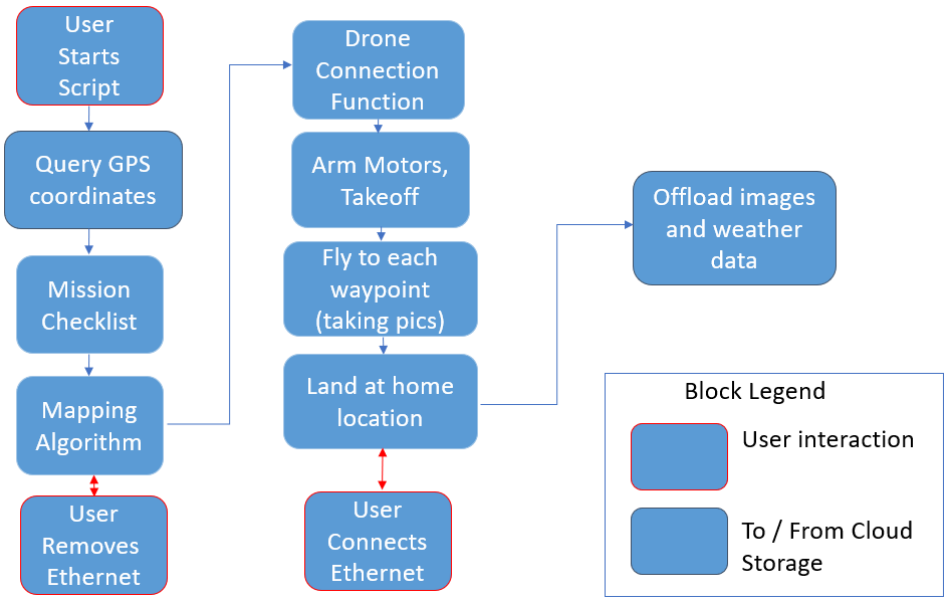


Figure 17: Flight Control / Image Capture Subsystem Flow

The programming for this subsystem was initially broken into two programs, the *startup_mission_checklist.py* and the *mappingPolygonV2.py*, python scripts. See the GitHub branch mentioned in the subsystem introduction to view these scripts. The first step in either program, is reading in the GPS coordinates for the field. The GPS coordinates come in the format shown below in Figure 12.

```
1   A, 30.622602, -96.332057
2   B, 30.623589, -96.330680
3   C, 30.625988, -96.334818
4   Home, 30.622602, -96.332057
```

Figure 18: GPS Coordinate Text File (test1_1.txt)

In the startup mission checklist code, the GPS coordinates are used to query the current weather conditions for the field location. This is achieved through weatherbit.io API, which allows for free student trials. To ensure that the weather conditions align with the drone flight requirements in the FSR document, the temperature, cloud coverage, windspeed, and precipitation levels are queried, and compared to the thresholds set in the code. If the weather conditions are satisfied, the mission startup checklist script continues to check if the area of the field and distance to the field are within the operating conditions stated in the FSR to ensure ample battery life to complete the scanning mission and return to base. If all the conditions are satisfied, the checklist script will pass, and allow the mapping script (*mappingPolygonV2.py* in the GitHub branch) to run.

Once the mission startup checklist is passed, the mapping polygon code first calculates the needed offsets between waypoints for traversing the field, and the distance offset in between consecutive images while flying to a waypoint. The code uses the flight altitude and the specifications of the Raspberry Pi V2.1 Camera to determine the needed offsets to capture the field with 75% side and frontal image overlap. See Figure 13 below for an example of image overlap and reference the image_footprint.py in the GitHub branch mentioned above to see the calculations that are executed to achieve the 75% overlap.

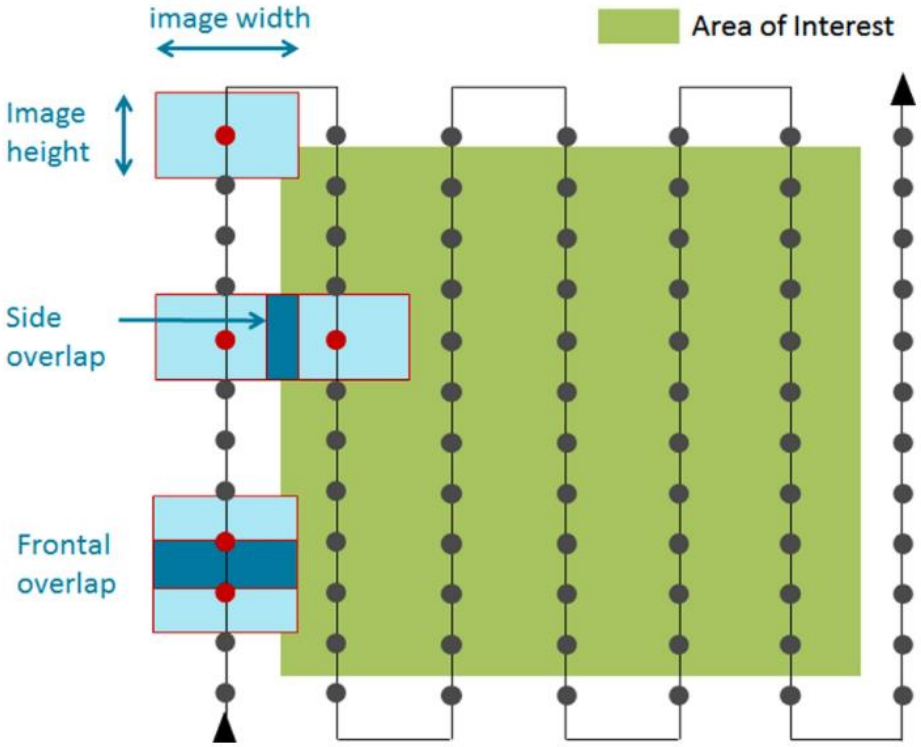


Figure 19: Image Overlap Example

Once the offsets for images and waypoints are determined, the actual waypoints are generated using my mapping algorithm. This specifically refers to the code from the script *mappingPolygonWaypointsV2.py*, which is the map_polygon function in the *mappingPolygonV2.*py script. The process of determining waypoints is an iterative code, which uses functions to do algebra with GPS coordinates. It calculates waypoints along the border of the field, spaced out by the previously calculated waypoint offset. Visit the GitHub folder to view testWaypointFiles, which include the waypoints determined for fifteen test files (GPS coordinate files for test fields), this is the output of the *mappingPolygonWaypointsV2.py* script. See Figure 14 below for an example of the waypoints determined for the test fields shown on Google MyMaps.

Figure 20: Mapping Algorithm Flight Path Outputs

One thing to point out in the rightmost picture of the Figure 14 is that although the original field was a 5-sided field, the mapping algorithm uses a larger 4-sided field to determine the waypoints. There is a condition in the mapping algorithm code that checks for obtuse interior angles in the field, and if found, removes the vertex GPS coordinate of the obtuse angle. This takes place because it simplifies logic for the mapping algorithm code (determining the waypoints for the flight). Additionally, in most cases, removing the obtuse angle coordinate speeds up the overall flight time by reducing the number of turns the drone needs to take, despite creating an overall larger field area. The obtuse interior angle consideration will be referenced again (see Figure 19), in the following section where I present data about the simulated flights for the fifteen test cases.

Once the waypoints are determined, the script continues to instruct the drone to fly to each of the waypoints. This is done using Dronekit python API, which is a library of python functions used to program drone flight. The main contribution from Dronekit that I used in my flight control code is the *simple_goto ()* function, which takes in the GPS coordinate to go to as an input. Then, once the drone is flying to the waypoint, my function *wait_to_arrive_at_wp_imaging()* is executed (see the function code in the *mappingPolygonV2.py* script on the GitHub branch). This function compares the current location of the drone to the location of the waypoint every half second, ensuring that the drone reaches the desired waypoint before the program continues and commands the drone to fly to the next waypoint. Within this function, the current location is also compared with the required image offset in meters. Every time the drone travels 9.16 meters (9.16 meters is the image offset required for 75% frontal image overlap at a flight altitude of fifty meters using the Pi V2.1 Camera) the camera is triggered to capture an image. This process is repeated for each waypoint until the entire field has been traversed and imaged Lastly, once the last waypoint is reached, the drone is instructed by the code to return to the home location specified in the input GPS file by the user, and lands.

3.3. Subsystem Validation

The first item to prove validation for this subsystem, is that the drone will not embark on missions that are unachievable due to battery considerations. This specifically means

that the drone will not take off when the field is either; 1). Too far away from the takeoff location (the threshold is set to one thousand meters), or 2). The field is too large (the threshold is set to 10 acres). Additionally, the drone will not embark on missions while the current weather is outside of the safe flight thresholds mentioned in the FSR. This specifically means that the drone will not take off when; 1). The precipitation levels are greater than one millimeter per hour, 2). The temperature is greater 39 degrees Celsius, 3). The cloud coverage is greater than 39% and 4). The windspeed is greater than fourteen meters per second.

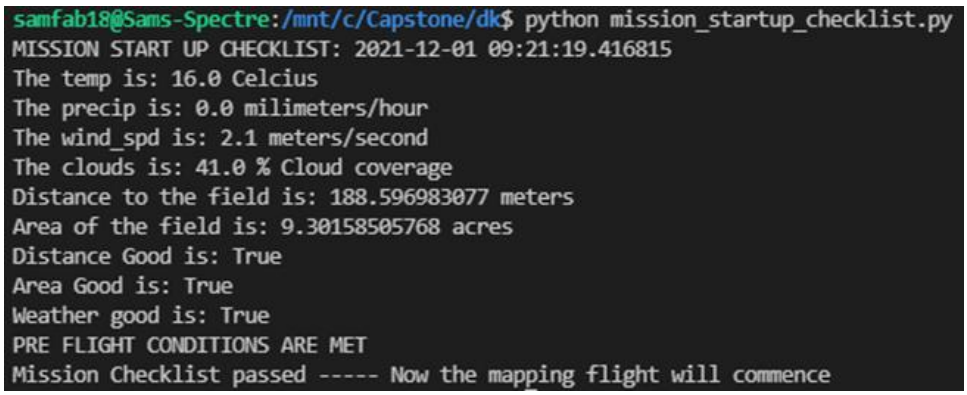


Figure 21: *mission_startup_checklist.py* Script Output: Passed Example

```
samfab18@Sams-Spectre:/mnt/c/Capstone/dk$ python mission_startup_checklist.py
MISSION START UP CHECKLIST: 2021-12-01 09:19:20.803491
The temp is: 16.0 Celcius
The precip is: 0.0 milimeters/hour
The wind_spd is: 2.1 meters/second
The clouds is: 41.0 % Cloud coverage
DISTANCE CHECK FAILED. Distance to field exceeds the allowed 1000 meters
Distance to the field is: 1679.75592999 meters
Area of the field is: 1.1028002868 acres
Distance Good is: False
Area Good is: True
Weather good is: True
PRE-FLIGHT CONDITIONS ARE NOT MET
Enter Y on the keyboard to override, or press enter to close program
N
OVERRIDE COMMAND NOT RECIEVED, MISSION WILL NOT BE EXECUTED
Mission Checklist failed ----- Now the script will close
samfab18@Sams-Spectre:/mnt/c/Capstone/dk$ python mission_startup_checklist.py
MISSION START UP CHECKLIST: 2021-12-01 09:22:37.499501
The temp is: 16.0 Celcius
The precip is: 0.0 milimeters/hour
The wind_spd is: 2.1 meters/second
The clouds is: 41.0 % Cloud coverage
AREA CHECK FAILED. Field area exceeds the allowed 10 acres
Distance to the field is: 362.944251347 meters
Area of the field is: 13.1295495667 acres
Distance Good is: True
Area Good is: False
Weather good is: True
PRE-FLIGHT CONDITIONS ARE NOT MET
Enter Y on the keyboard to override, or press enter to close program
Y
OVERRIDE COMMAND RECIEVED, EXECUTING MISSION
Mission Checklist passed ----- Now the mapping flight will commence
```

Figure 22: *mission_startup_checklist.py* Script Output: Failed Examples

Figure 15 and 16 above show outputs of the *mission_startup_checklist.py* script for different test fields. Figure 15 is an example output where both the weather and field area / distance conditions are passed, allowing the program to continue and the drone to execute its flight mission. Figure 16 shows examples of the *mission_startup_checklist.*py script failing on the distance to field condition (top), and therefore blocking the drone from executing its mission. In Figure 16 (bottom), the checklist fails on the field area too large condition. Notice that in Figure 16 (bottom) the user activates the option to override the checklist result and execute this mission despite the checklist failing. I decided to include the override option in this code in the case that the user wants to ignore the checklist and launch the flight regardless. After developing and assessing the *mission_startup_checklist.py* script, it has been implemented in ECEN 404 into the integrated overall flight mission script named *demo.py* on the GitHub.

The next item to validate is that the flight control program (referring to *mappingPolygonV2.py* on the GitHub branch listed in the introduction) can determine

suitable flight paths for 3-to-5-sided fields. To validate this, I created fifteen test field GPS coordinate files, making up fields ranging from 8-10 acres. I ran the scripts *mappingPolygonV2.py* and *mappingPolygonWaypointsV2.py* with each of the fifteen test fields to run flight simulations and record the waypoints for the simulations, respectively. An example output of the *mappingPolygonWaypointsV2.py* script is shown below in Figure 17, the test file used was *test2_1.txt*.

```
########## 29/11/2021 ##########
Test file specified: test2_1.txt
The calculated area of the field in acres is: 9.3016
The mapping step for rows is (meters): 15.4605263158
The displacement for images is (meters): 9.16318998716

 GPS COORDINATES IN ORDER
Home  :30.622602,-96.332057
1  :30.624927,-96.332343
2  :30.624927,-96.332343
3  :30.624927,-96.332343
4  :30.6248255531,-96.3322169117
5  :30.6248255531,-96.3324827493
6  :30.6247241063,-96.3326224985
7  :30.6247241063,-96.3320908234
8  :30.6246226594,-96.3319647351
9  :30.6246226594,-96.3327622478
10  :30.6245212125,-96.332901997
11  :30.6245212125,-96.3318386468
12  :30.6244197656,-96.3317125585
13  :30.6244197656,-96.3330417463
14  :30.6243183188,-96.3331814955
15  :30.6243183188,-96.3315864702
16  :30.6242168719,-96.3314603819
17  :30.6242168719,-96.3333212448
18  :30.624115425,-96.333460994
19  :30.624115425,-96.3313342936
20  :30.6240139782,-96.3312082053
21  :30.6240139782,-96.3336007433
22  :30.6239125313,-96.3337404925
23  :30.6239125313,-96.331082117
24  :30.6238110844,-96.3309560287
25  :30.6238110844,-96.3336101751
26  :30.6237096375,-96.3334798577
27  :30.6237096375,-96.3308299404
28  :30.6236081907,-96.3307038521
29  :30.6236081907,-96.3333495403
30  :30.6235067438,-96.3332192229
31  :30.6235067438,-96.3308402801
32  :30.6234052969,-96.3309767082
33  :30.6234052969,-96.3330889054
34  :30.6233038501,-96.332958588
35  :30.6233038501,-96.3311131362
36  :30.6232024032,-96.3312495642
37  :30.6232024032,-96.3328282706
38  :30.6231009563,-96.3326979532
39  :30.6231009563,-96.3313859923
40  :30.6229995094,-96.3315224203
41  :30.6229995094,-96.3325676358
42  :30.6228980626,-96.3324373184
43  :30.6228980626,-96.3316588484
44  :30.6227966157,-96.3317952764
45  :30.6227966157,-96.3323070009
46  :30.6226951688,-96.3321766835
47  :30.6226951688,-96.3319317045
Last  :30.622602,-96.332057
```

Figure 23: *mappingPolygonWaypointsV2.py* output for test file *test2_1.txt*

Figure 17 above shows the waypoints generated to scan the test field two_one. Please visit the folders GPSFieldTestFiles and testWaypointsFiles, in the GitHub branch, to view the test field GPS coordinate files and the generated waypoint files for each test field. There are five test cases each for 3, 4, and 5 sided fields. These files validate that my subsystem code can determine a waypoint-based flight path for fields of 3, 4, and 5 sided fields.

The next item to validate in my subsystem validation plan is that the drone flight programming has an image coverage greater than 85% of the field, with 60-80% side and frontal image overlap. The 60-80% overlap coverage is validated by the *image_footprint.py* script mentioned in the above subsystem details section. To validate the image coverage of the field, I recorded data from each of the fifteen test fields in mission planner simulation.

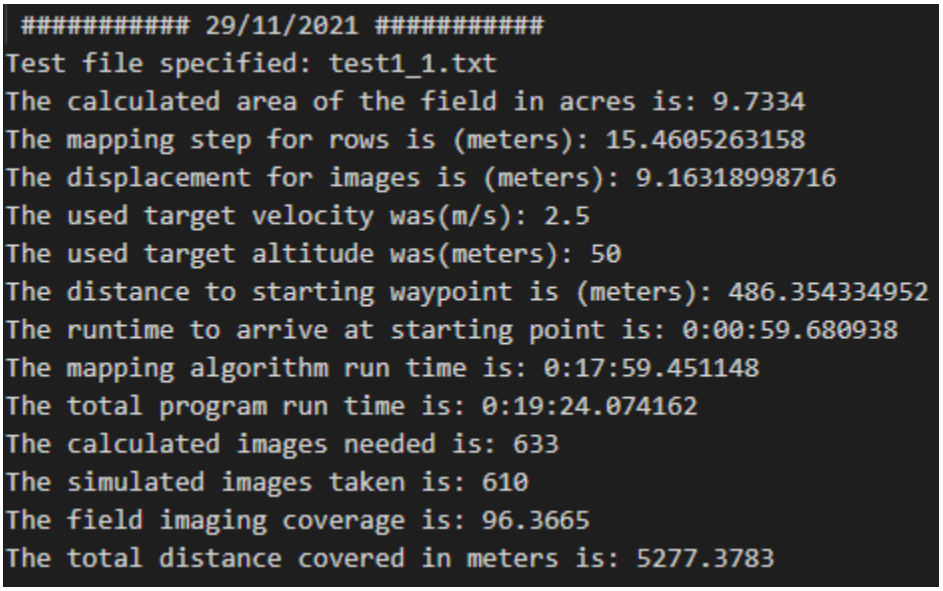


Figure 24: mappingPolygonV2.py Test Data Output File Example

Figure 18 above is an example output from the flight simulation script *mappingPolygonV2.py*. The testing data includes the test file used, calculated field area in acres, the image and waypoint offsets calculated/used in the flight, the simulated images captured / needed, and lastly information regarding the runtime of the program and the mapping flight. See the testResultFiles folder in the subsystem GitHub branch to view each of the output test files.

3 Coordinate Plots:						
Test File #:	1	2	3	4	5	Average:
Plot Area (Acres):	9.7334	9.7876	9.0091	9.0549	9.9911	9.51522
Home to Start Distance (m):	486.35	507.02	481.76	0	267.18	
Home to Start Flight Time (mins):	0.98	0.7583	0.7688	0.6	0.55	
Mapping Flight Time (mins):	17.99	16.33	17	13.57	12.45	15.468
Total Flight Time (mins):	19.19	17.74	18.74	15.6	14.07	17.068
Total Distance Traversed (m):	5277.39	5155.94	4755.87	4783.2	3734.7	4741.42
Expected Images Needed:	633	619	586	541	466	569
Modeled Images Taken:	610	603	568	529	453	552.6
Image Field Coverage (%):	96.37%	97.42%	96.93%	97.78%	97.21%	97.14%
Mapping Speed (Acres/Min):	0.541045	0.599363	0.529947	0.667273	0.802498	0.628025

4 Coordinate Plots:						
Test File #:	1	2	3	4	5	Average:
Plot Area (Acres):	9.3016	9.3682	9.4924	9.518	7.5515	9.04634
Has Obtuse Angle (Y/N):	N	N	N	Y	Y	N/A
Obtuse Adjusted Plot Area (Acres):	N/A	N/A	N/A	13.13	8.6435	10.88675
Home to Start Distance (m):	260.77	0	0	0	169.22	
Home to Start Flight Time (mins):	0.56	0.008	0.5	0.0417	0.63	
Mapping Flight Time (mins):	13.82	10.6	12.93	14.56	11.18	12.618
Total Flight Time (mins):	15.23	12.63	14.87	16.58	13	14.462
Total Distance Traversed (m):	4525.78	4516.54	4529.16	6352.8	4550.2	4894.896
Expected Images Needed:	556	524	529	760	514	576.6
Modeled Images Taken:	552	524	524	749	505	570.8
Image Field Coverage (%):	99.28%	100.00%	99.06%	98.55%	98.25%	99.03%
Mapping Speed (Acres/Min):	0.673054	0.883792	0.734138	0.901786	0.773122	0.793178

5 Coordinate Plots:						
Test File #:	1	2	3	4	5	Average:
Plot Area (Acres):	8.9601	9.5852	9.1285	8.6207	8.115	8.8819
Has Obtuse Angle (Y/N):	N	N	N	Y	Y	N/A
Obtuse Adjusted Plot Area (Acres):	N/A	N/A	N/A	10.18	8.8806	9.5303
Home to Start Distance (m):	113.36	0	0	215.99	0	
Home to Start Flight Time (mins):	0.59	0.468	0.008	0.467	0.008	
Mapping Flight Time (mins):	10.87	11.8	11.87	9.97	11.532	11.2084
Total Flight Time (mins):	12.64	13.29	13.8	11.93	13.44	13.02
Total Distance Traversed (m):	4129.8	4435.8	4330.44	4519.91	4181.013	4319.393
Expected Images Needed:	462	510	527	494	510	500.6
Modeled Images Taken:	455	454	506	494	509	483.6
Image Field Coverage (%):	98.48%	89.02%	96.02%	100.00%	99.80%	96.66%
Mapping Speed (Acres/Min):	0.824296	0.812305	0.76904	0.864664	0.703694	0.7948

Figure 25: Test Flight Data Tables

Figure 19 above shows the data collected for all simulation test flights. The cells highlighted green are those that are meeting the requirements stated in the validation and execution plan, and the red cells are those that are not meeting the requirements.

Cumulative Averages	
Plot Area (Acres):	9.14782
Mapping Flight Time (mins):	13.09813
Total Flight Time (mins):	14.85
Image Field Coverage (%):	97.61%
Mapping Speed (Acres/Min):	0.738668

Figure 26: Test Data Cumulative Average Table

Figure 20 shows the cumulative averages of the important data in the simulation tests shown in Figure 19. By observing Figures 19 and 20, most of the flight requirements are being satisfied by my flight control code. I would like to point out that the reason for test files 1, 2, and 3 in the three coordinate plots section of Figure 19 are failing the flight time parameter due to the unusual shape of the test fields. Those three test fields are long right triangles, which resulted in a flight path with a lot of turns, causing a larger mapping flight runtime. These are unlikely orientations for real cotton plots, but nevertheless, the algorithm could be improved to identify and better handle fields like these. With the data shown in Figure 19 and 20, I can confidently validate that my subsystem will be able to determine and execute flight paths for 3 to 5 sided fields. Additionally, while on the flight, my flight program will capture enough images to cover 85% of the field within the 60-80% frontal and side overlap specified in the validation plan.

Lastly, I will now validate the successful integration of all the code and its ability to successfully fly the drone. To validate the overall integration and drone flight success, I have provided the terminal output of the final demo flight script recorded on 4/20/2022 below. The demo script can be located at the GitHub link https://github.com/DamilolaOwolabi/ECEN-403-GROUP-66/tree/master/drone_flight_code_Sam/demoFiles/demoScripts.

```
pi@raspberrypi:~/demo $ python demo.py --connect /dev/ttyAMA0
Connected to the Internet
Type your account username, then press enter. (case senstitive)
samf
File selected from Cloud Storage is: samf.txt
Latitude step for imaging rows is: 3.04340615132e-05
Meter offset of images is: 2.74895699615
The GPS coordinates of the field recieved are:
A,30.64039, -96.47098
B,30.64062, -96.47107
C,30.64062, -96.47079
Home,30.64039, -96.47098
-----------------------------------------
MISSION START UP CHECKLIST: 2022-04-20 17:41:26.316619
The temp is: 24.0 Celcius
The precip is: 0.437424 milimeters/hour
The wind_spd is: 7.5 meters/second
The clouds is: 85.0 % Cloud coverage
WEATHER CHECK FAILED ON: Cloud Coverage: 85.0 % Cloud coverage
Distance to the field is: 33.2097992489 meters
Area of the field is: 0.08483435979666346 acres
-----------------------------------------
Distance Good is: True
Area Good is: True
Weather Good is: False
PRE-FLIGHT CONDITIONS ARE NOT MET
Enter Y on the keyboard to override, or press enter to close program
y
OVERRIDE COMMAND RECIEVED, EXECUTING MISSION
Mission Checklist passed ----- Now the mapping flight will commence
-----------------------------------------
Determining which mapping algorithm to run:
Obtuse was not found, running normal algorithm
Latitudes are close: Running Mapping Algorithm Reverse
Mapping Polygon function has concluded
The waypoints determined by the mapping algorithm are:
Home,30.64039,-96.47098
1,30.64039,-96.47098
2,30.6404204341,-96.4709548588
3,30.6404204341,-96.470991909
4,30.6404508681,-96.471003818
5,30.6404508681,-96.4709297176
6,30.6404813022,-96.4709045765
7,30.6404813022,-96.4710157269
8,30.6405117362,-96.4710276359
9,30.6405117362,-96.4708794353
10,30.6405421703,-96.4708542941
11,30.6405421703,-96.4710395449
12,30.6405726044,-96.4710514539
13,30.6405726044,-96.4708291529
14,30.6406030384,-96.4708040117
15,30.6406030384,-96.4710633629
Last,30.64062,-96.47079
---------- End of Waypoint list ------------
-----------------------------------------
Enter 'pic' if you would like to image on this flight.
pic
```

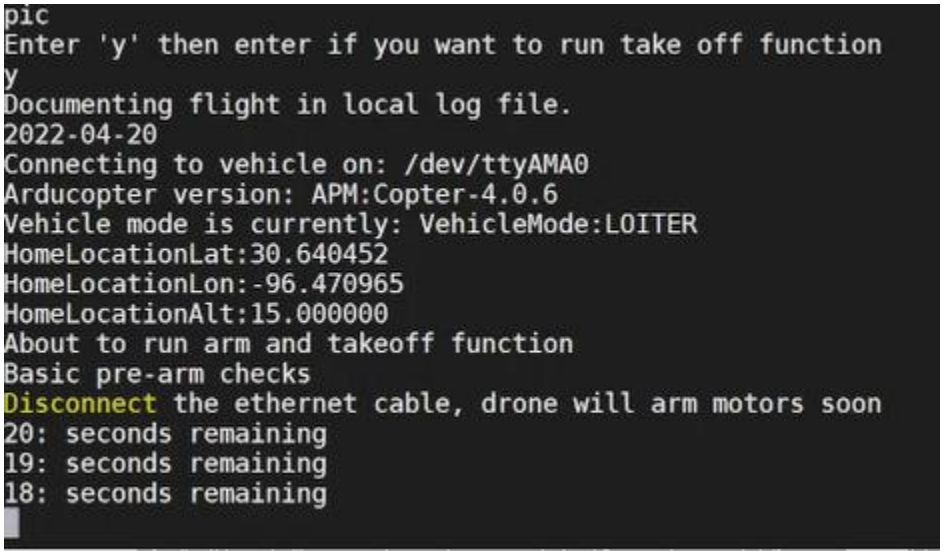


Figure 27.1: *demo.py* Terminal Output Before Flight

The above images show the script first prompting the user for their account name registered on the user interface application. This is necessary to specify which account folder on the cloud database from which to query the GPS coordinate text file. Next the GPS coordinate text file is queried, and the previously mention mission startup checklist code is executed to ensure safe flight conditions. Note that in this case, we did decide to override the cloud coverage failure on the mission checklist for the sake of recording data for our project demo. Another thing to note is that we did include an additional field for the user to specify if they want to run the takeoff function to ensure that the surroundings were safe before allowing the drone to start flying. Lastly, once the ethernet cable was disconnected, the SSH session ended, hence the break before the next set of terminal outputs.

```
pi@raspberrypi:~ $ screen -r
('Altitude: ', 0.087)
('Altitude: ', 0.284)
('Altitude: ', 1.987)
('Altitude: ', 4.122)
('Altitude: ', 6.383)
('Altitude: ', 8.632)
('Altitude: ', 11.02)
('Altitude: ', 13.132)
('Altitude: ', 14.35)
Reached target altitude
Takeoff complete
Distance to wp: 7.358093
Arrived at WP
Distance to wp: 3.710252
Arrived at WP
Distance to wp: 4.435945
Arrived at WP
Distance to wp: 3.903450
Arrived at WP
Distance to wp: 7.846824
Arrived at WP
Distance to wp: 4.577852
Arrived at WP
Distance to wp: 11.534989
Arrived at WP
Distance to wp: 4.379180
Arrived at WP
Distance to wp: 16.309296
Arrived at WP
Distance to wp: 4.688087
Arrived at WP
Distance to wp: 19.444012
Arrived at WP
Distance to wp: 5.713349
Arrived at WP
Distance to wp: 24.904164
Arrived at WP
Distance to wp: 4.621443
Arrived at WP
Distance to wp: 28.409295
Arrived at WP
Distance to wp: 30.305879
Arrived at WP
Initiating Landing Sequence
Vehicle closed, Drone is all done!
Writing Image Location
Drone has landed, connect ethernet after rotors are done spinning
Cloud offload will start once internet connection is established
Offloading in 30 seconds
No internet connection.
attempt # 0
Not Connected, reset connection
Connected to the Internet
```

```
Initiating Landing Sequence
Vehicle closed, Drone is all done!
Writing Image Location
Drone has landed, connect ethernet after rotors are done spinning
Cloud offload will start once internet connection is established
Offloading in 30 seconds
No internet connection.
attempt # 0
Not Connected, reset connection
Connected to the Internet
Connected to the Internet
Image offload function starting at time: 2022-04-20 17:58:04.624031
^[[B^[[B^[[B^[[B^[[B^[[BNumber of images/files successfully uploaded to cloud: 75
Image offload function ending at time: 2022-04-20 18:00:14.821019
pi@raspberrypi:~/demo $
```

Figure 27.2: End of *demo.py* Terminal Output of *demo.py*

The above images show the completion of flight script once the drone has landed and the ethernet cable has been reinserted to the MCU to allow WIFI sharing and recovery of the SSH session. At the end of the terminal output, it shows that seventy-two images of the field and three additional text files have been offloaded back to the cloud database to the account ‘samf.’ The seventy-five total files were offloaded in one minute and 56 seconds meeting the three second per file requirement mentioned in the attached Interface Control Document. Finally, to validate the image coverage of the field, reference the two images below. The image on the left is the planned flight path, the image on the right is the flight path with image locations marked.

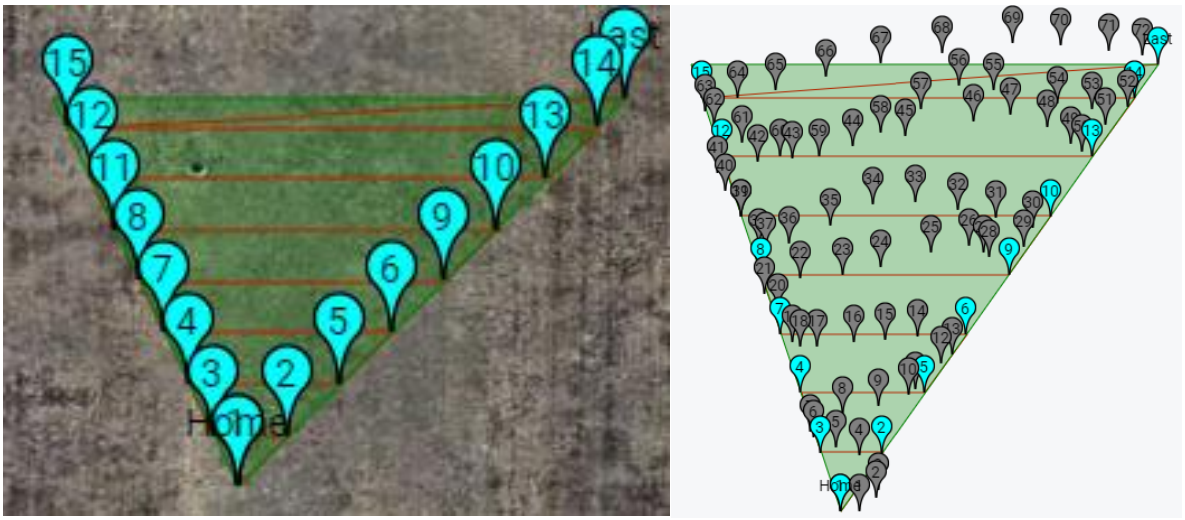


Figure 28: Flight Path and Image Location

In the above images, the green shaded region shows the field specified by the GPS coordinate text file. The cyan labels mark the waypoints determined by the flight path algorithm and the red lines show the determined flight path. The gray markers indicate the location of the drone at the time each image was captured. I note that the image locations do not exactly line up with the planned flight path. We believe this to be the result of windy conditions during the day of the flight, but due to complications and a limited time frame to actually fly the drone, we were not able to determine the issue and

correct it. To further validate the effectiveness of flight ability, please reference the videos *demoMovie and Flight Video* at the following links where you can view the drone flying the flight path in the above figure.

https://photos.app.goo.gl/syskFH5DS3jfRvUY9

https://photos.app.goo.gl/rdEKUZFQDXw4TAya7

3.4. Subsystem Conclusion

With the information outlined in the previous three subsections, I conclude that my subsystem is fully integrated with the other subsystems and operates as intended. The above details section outlines some of the work I completed for my subsystem and walks through the general flow of my subsystem. In the subsystem validation section, I break down how I have validated all items in my subsystem validation and execution plan, and present testing data that I collected, to justify that the validation items are in fact met. Please visit the GitHub repository branch https://github.com/DamilolaOwolabi/ECEN-403-GROUP-66/tree/master/drone_flight_code_Sam . to view the code referenced in the Flight Control / Image Capture Subsystem report. All the scripts in the GitHub branch were developed by me to complete this subsystem.

4. Drone Enclosure and Hardware

4.1. Subsystem Introduction

In this subsystem, the 3D design and printing of the drone enclosure takes place as well as the selection of components needed for the MCU to function as desired. The team member, Luis Sanchez, is responsible for this subsystem.

4.2. Subsystem Details

An image showing the high-level connection of the power circuit for the MCU is shown below.

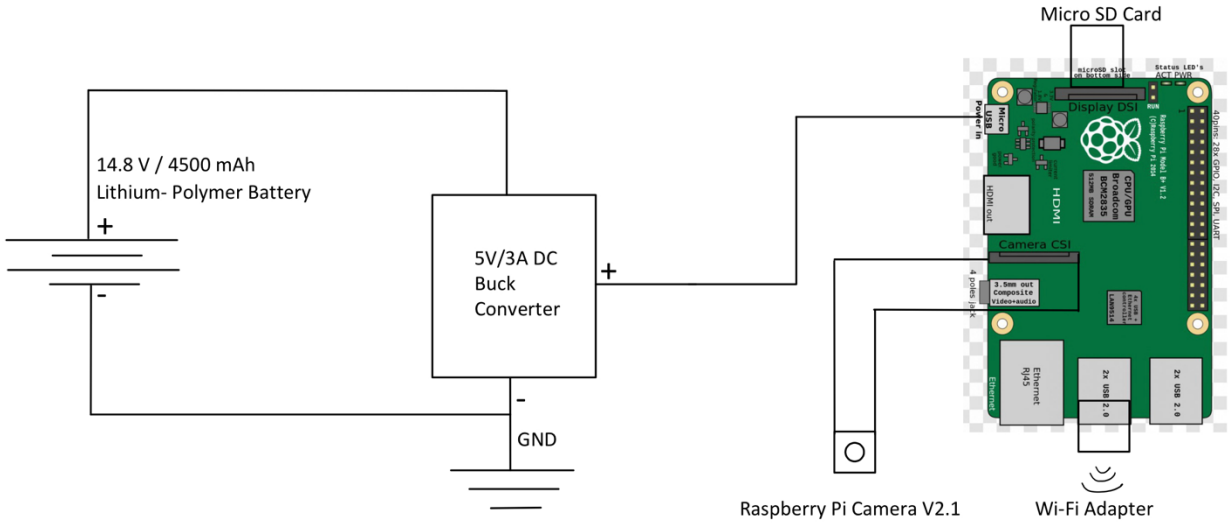


Figure 29: Circuit for Powering MCU

The MCU is powered using a 4500 mAh li-po battery which delivers 14.8 volts to a 5V/3A buck converter that is connected to the pins in the Raspberry Pi (MCU).

The battery has enough mAh (milliamp-Hours) to power the MCU for 1.5 hours since the buck converter delivers 3A. However, the MCU will only need to operate for 30 minutes at a time, which allows for image capture during flight and image upload to google cloud post-flight.

The image below shows the 3D model of the enclosure done in solid works and the one below is the enclosure with all the components connected.

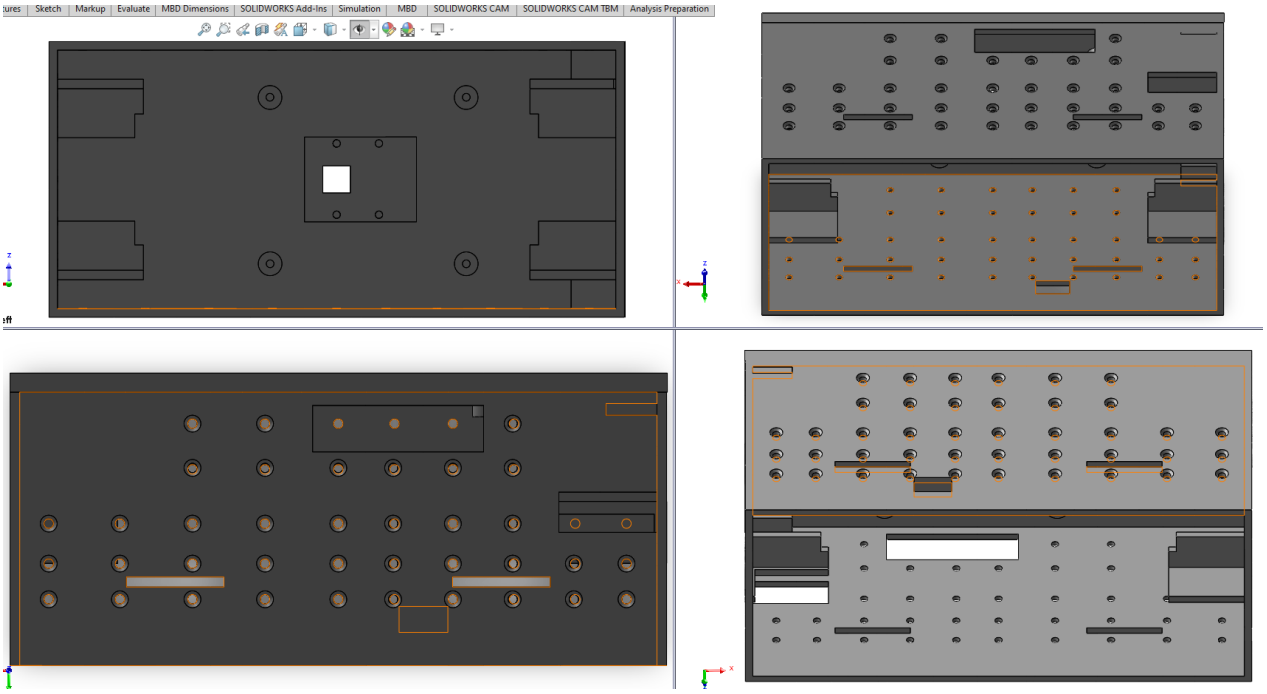


Figure 30: Drone Enclosure 3D design

Figure 31: Drone Enclosure

The enclosure includes an opening on the underside for the camera to take images while in flight. It also includes holes on all sides for cooling and ventilation of the components as well as openings on the sides for connections between the MCU and the drone/laptop. In addition, each component has a designated space, the camera on the very bottom, the MCU is supported by four elevated points extending from the bottom to provide space between the MCU and the camera for cooling, and finally, the battery is supported above the MCU by ledges on the either side allowing separation for cooling as well.

4.3. Subsystem Validation

The buck converter was assessed to check if it delivered 5V by connecting the battery to the receiving end and a multimeter in the converting end. This confirmed that the buck converter outputs the 5V necessary to power the Raspberry Pi for the functioned properly and would be able to power the MCU.

Figure 32: Buck Converter Output Test Successful

The image above shows that the output of the buck converter is 5V (+- 2%) when assessed using the multimeter.

Additionally, by knowing that the battery delivers 4.5 amps for an hour before needing to recharge and the fact that the buck converter delivers three amps to the MCU, we can calculate that the Raspberry Pi can run for 1.5 hours before turning off. The MCU was run for 45 minutes while running other tests to confirm that it would run for more than the needed 30 minutes, which proved successful. We then repeated the testing before recharging the battery and the Raspberry Pi stayed on for another 47 minutes before powering off. This confirmed that the battery can power the MCU for a bit over 1.5 hours when used at full charge. Finally, the entire enclosed system must weigh less than the maximum payload of 3.5 lbs for the drone to fly. Therefore, we ensured that the enclosed system weighed less than 1.75 lbs. We did this by weighing the enclosed system on a scale in the FEDC.

4.4. Subsystem Conclusion

Each part of this subsystem, the drone enclosure and power circuit, passed all validation tests and can power and house the MCU needed for image capture during flight and data upload onto the google cloud after each flight.

5. Data Management and Architecture

5.1. Subsystem Introduction

This subsystem manages the data that is collected from all subsystems. Everything from the storage of code used to run different tasks to the upload of images from the MCU to a cloud storage for use by other subsystems. The team member, Luis Sanchez, is responsible for this subsystem.

5.2. Subsystem Details

We are using one bucket on the google cloud storage to store everything used in the user part of the project. Each time a user makes an account and uploads their GPS coordinates for their field, a unique folder is created including subfolders where their data will be safely stored.

Figure 23 shows all the current accounts that reside inside the User Account bucket.

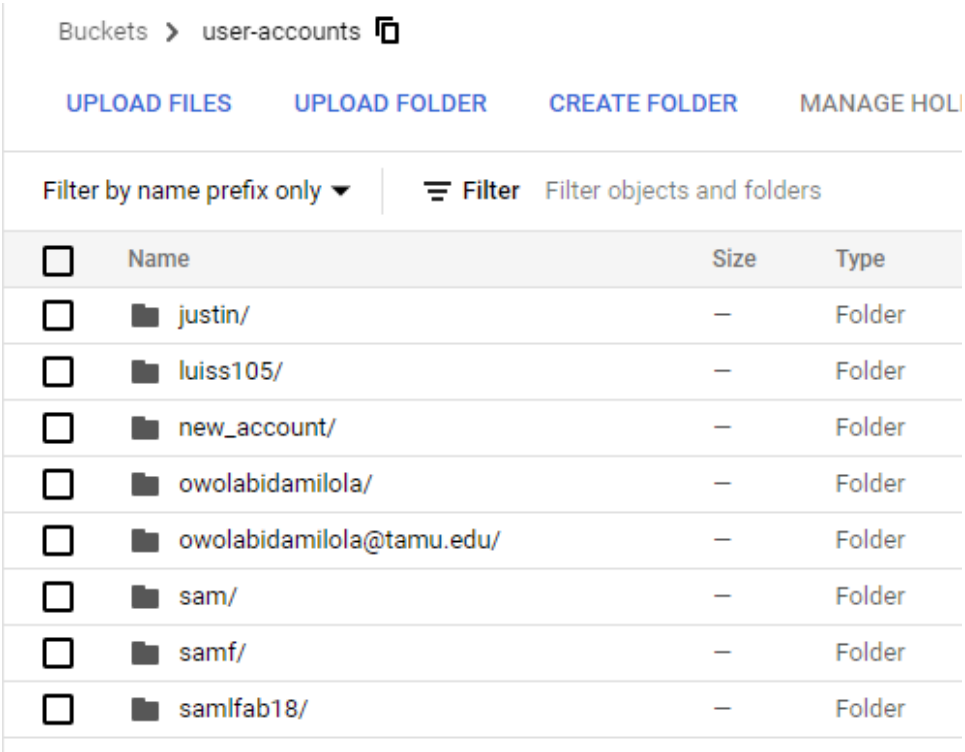


Figure 33: User Accounts Bucket

Within each user folder, they will have a text file with the GPS coordinates they entered on the website, a folder for pictures taken while the drone is in flight, a folder for the prediction reports the ML generates for the user, a folder for graphs that aid the said reports, and finally a folder with additional data to help the ML model with the generation of reports and graphs.

Figure 24 shows the folders for one of the accounts created by us.

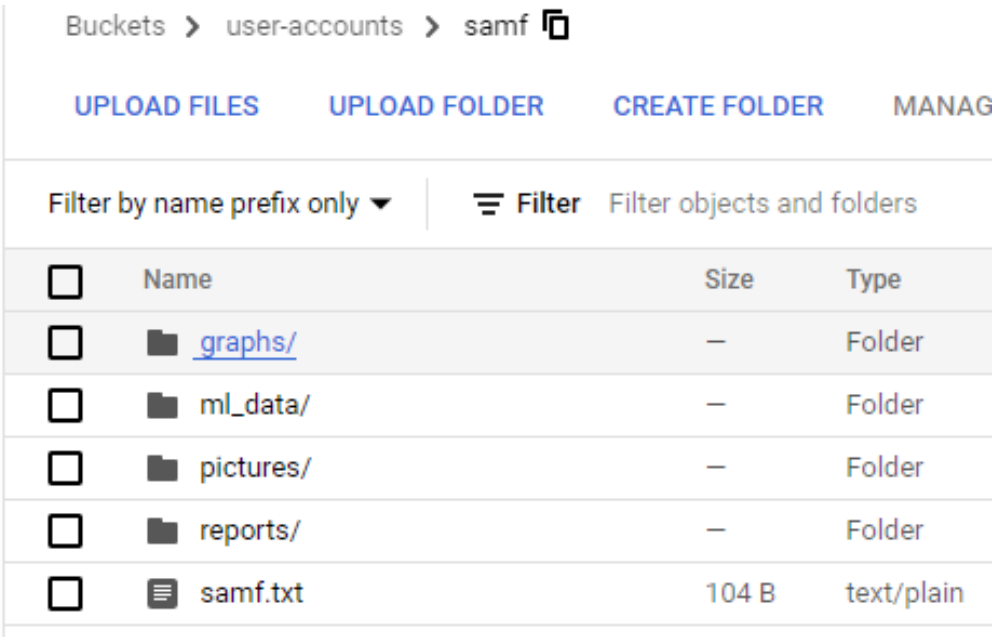


Figure 34: Folders/Files in Each User Account

Like mentioned before, all the subsystems have access to this cloud storage. The pictures folder is accessed by the MCU. It uploads the images taken during flight after the drone lands. The MCU iterates through the picture folder in its local storage and uploads one image at a time. Once an image is uploaded to the cloud successfully, it deletes that image to save memory for the next flight. On the other hand, the ML and UI both access the graphs and reports folders. The ML accesses data from the ml_data folder to generate the predictions for the harvest date and amount. After these predictions are generated, the ML uploads a report into the reports folder and corresponding graphs onto the graphs folder. Afterwards, the UI accesses these files and uploads them to the website where the user can view this information and amend their farming process accordingly. One important thing to note is that because each user has their own account, they are only able to access their own data. This is an integrated safety feature.

Finally, we are safely storing all the code that makes this topology possible as well as the code for the rest of the subsystems in a GitHub repository. This aids us to always have our code backed up and track the changes we make to these codes. Figure 35 shows the teams GitHub repository.

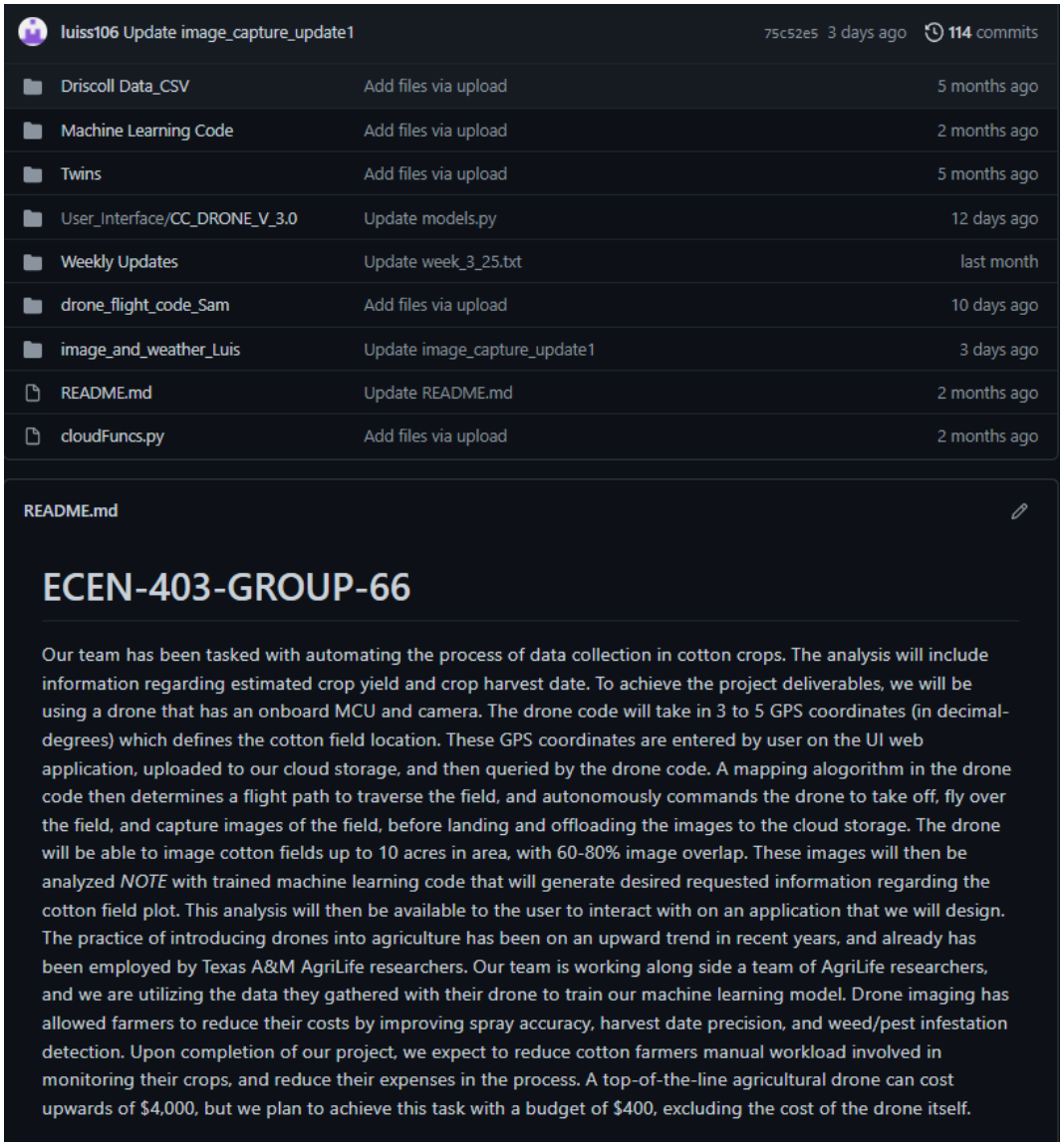


Figure 35: GitHub Repository

5.3. Subsystem Validation

To validate that the data management subsystem functions as needed, the topology was checked to ensure that the correct data was delivered to its corresponding location.

Figure 36 illustrates the picture folder with a subfolder for the day a set of pictures was taken.

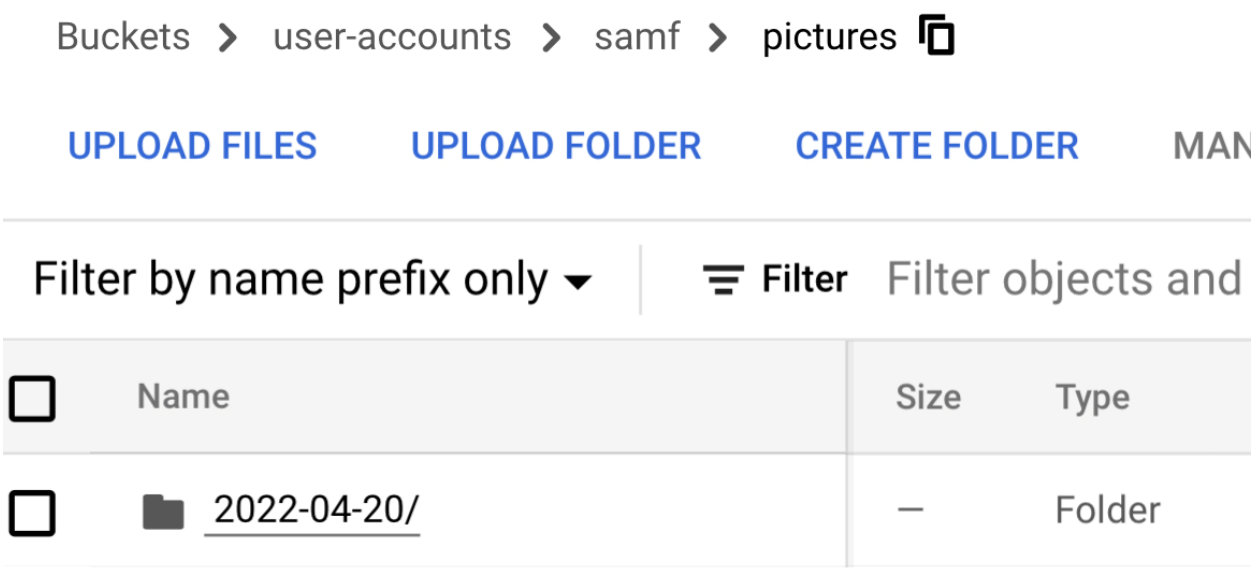


Figure 36: MCU pictures folder

Figure 37 shows that a set of pictures were successfully uploaded from the MCU to the cloud after a flight.

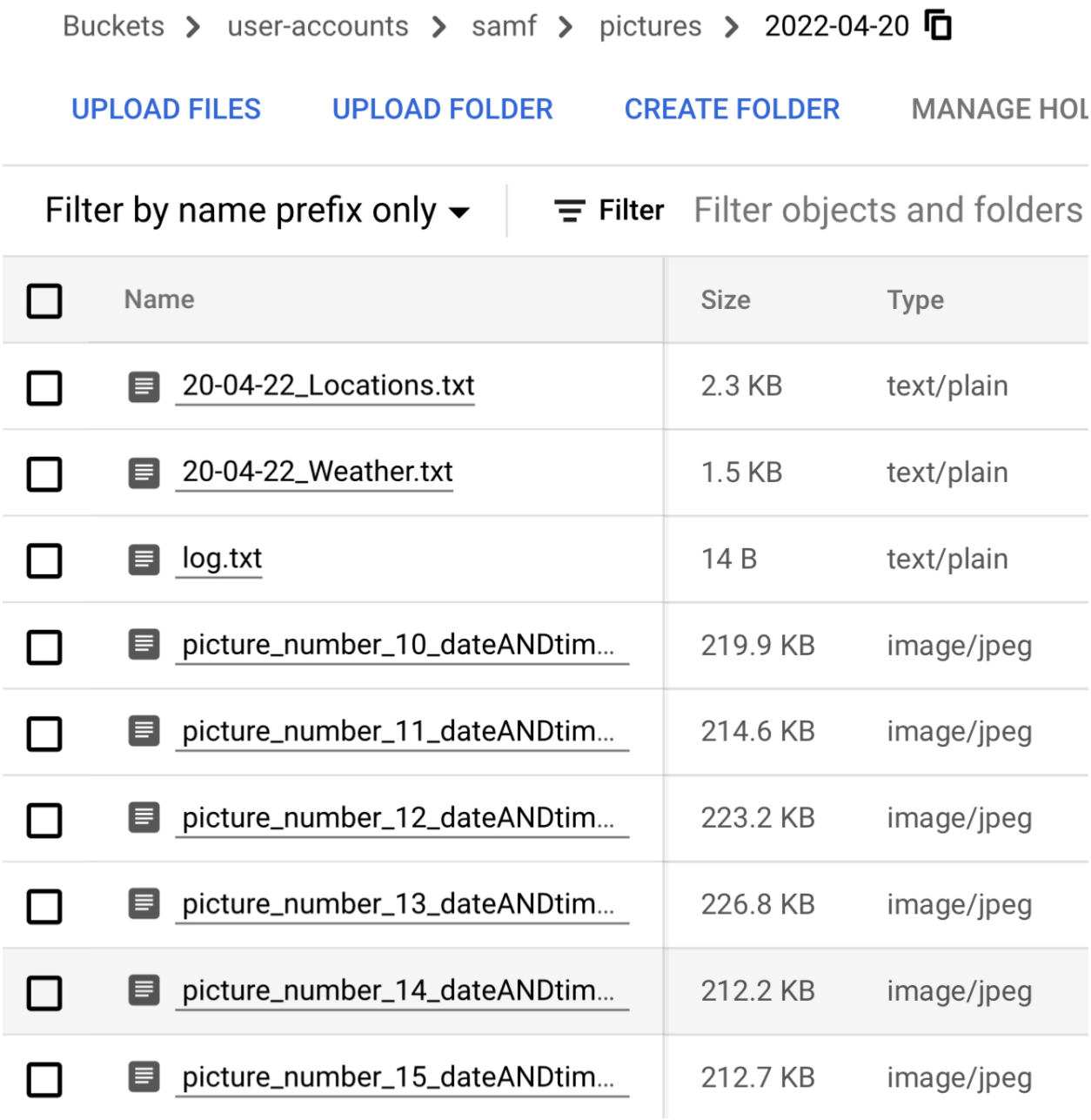


Figure 37: MCU pictures Upload from MCU to Cloud

Figure 38 shows the graphs folder, which the ML uploads to for the UI to access and display on the website.

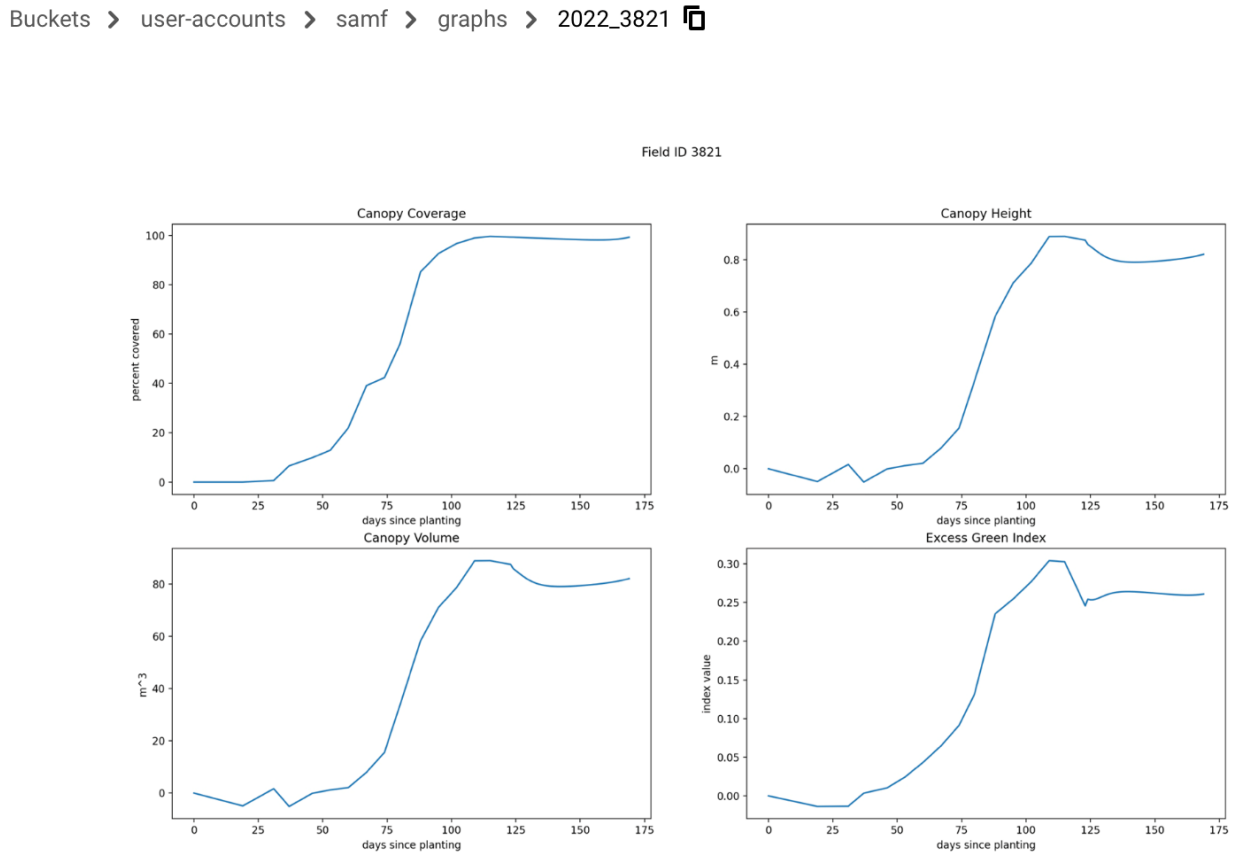


Figure 38: Successful Upload of Graphs from ML to Cloud

Figure 39 illustrates the reports folder, which the ML uploads to for the UI to access and display on the website.

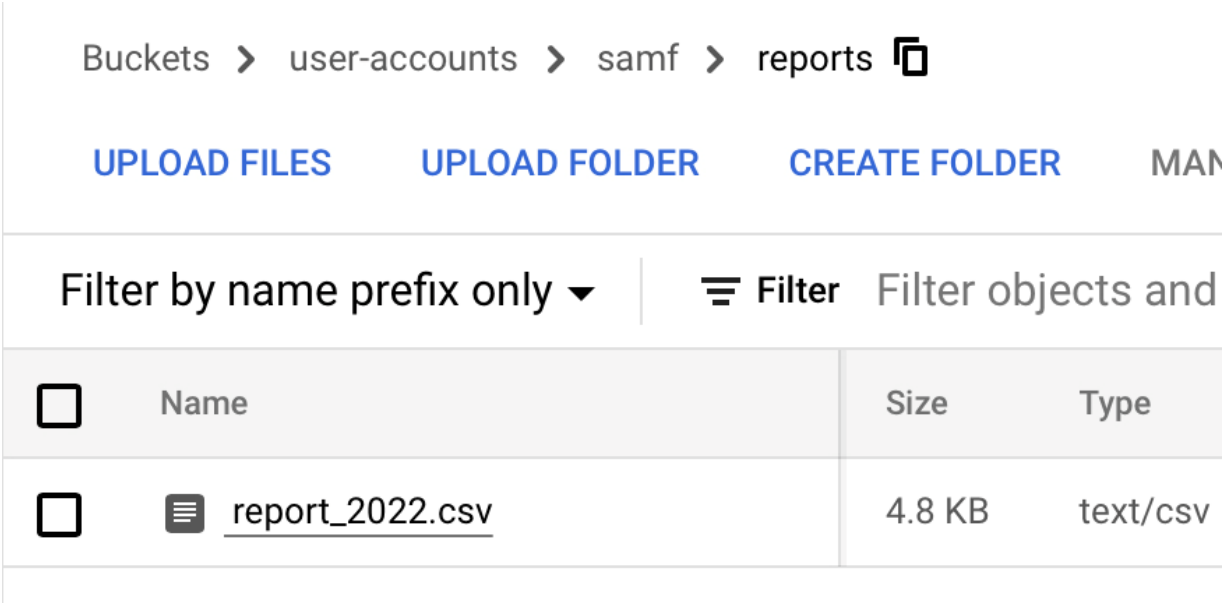
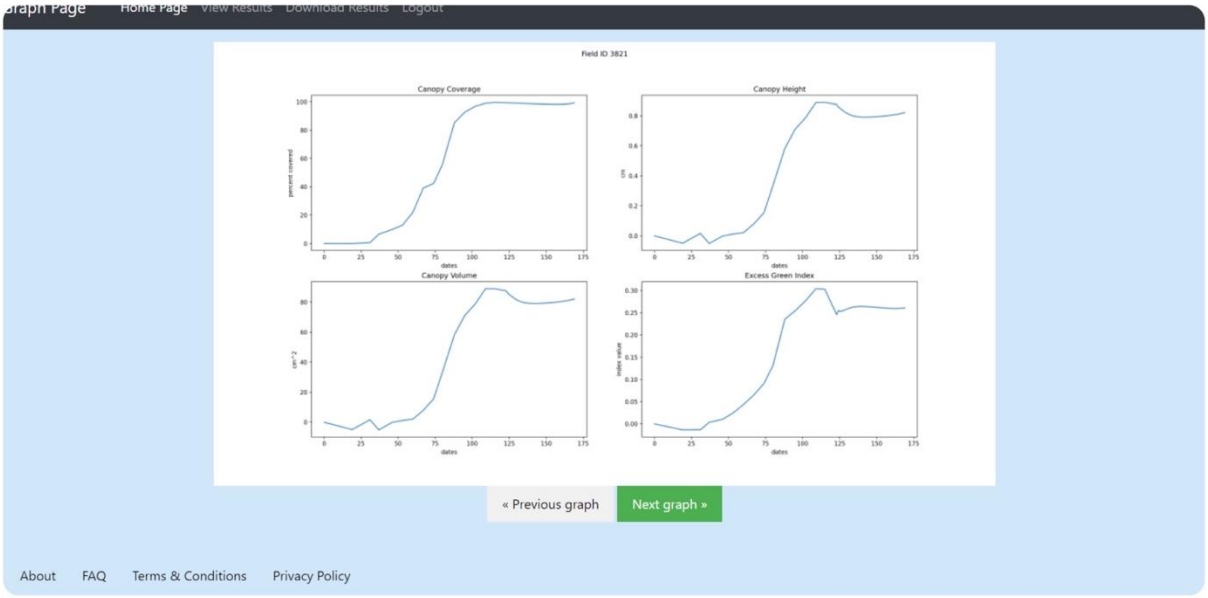


Figure 39: Successful Upload of Reports file from ML to Cloud

Figure 40 shows the graphs and report being displayed to the user on the website, validating that the UI can access the data on google cloud storage.



fieldID	crop yield (gram) with confidence ratings (in %)	Predicted Harvest dates (day, month)
3821	38g - 48g with a confidence of 36%	03, Aug
3822	38g - 48g with a confidence of 36%	11, Aug
3823	38g - 48g with a confidence of 35%	03, Aug
3824	38g - 48g with a confidence of 37%	28, Jul
3825	38g - 48g with a confidence of 27%	28, Jul
3826	38g - 48g with a confidence of 21%	28, Jul
3827	8g - 18g with a confidence of 46%	28, Jul
3828	8g - 18g with a confidence of 43%	28, Jul
3829	8g - 18g with a confidence of 48%	28, Jul
3830	8g - 18g with a confidence of 46%	28, Jul
3831	8g - 18g with a confidence of 50%	28, Jul
3832	8g - 18g with a confidence of 49%	28, Jul
3833	8g - 18g with a confidence of 50%	28, Jul

Figure 40: Successful Extraction and Display of Data to User via UI

5.4. Subsystem Conclusion

In conclusion, the data management subsystem successfully builds a bridge between the other subsystems that allows them to share information and display it to the user

6. User Interface Subsystem Report

6.1. Subsystem Introduction

The user interface is the first contact the user has with the system. It is designed to receive GPS coordinate input from the user and make the information available for the drone to use to program a flight. The MCU displays the output of the flight commands to the user to make them aware of any errors during the flight program. The subsystem is also in charge of assuring total security and privacy to the user. This is done by utilizing login and registration pages, which tailors the interface specifically for the user. The subsystem was assessed to confirm its simplicity, responsiveness, and maintainability. The tests helped prove that the subsystem integrates well with other subsystems in the CC_DRONE project. The team member, Damilola Owolabi, is responsible for this subsystem. The subsystem code is located in the GitHub link: https://github.com/DamilolaOwolabi/ECEN-403-GROUP-66/tree/master/User_Interface .

6.2. Subsystem Details

A functional flowchart for the subsystem is attached below.

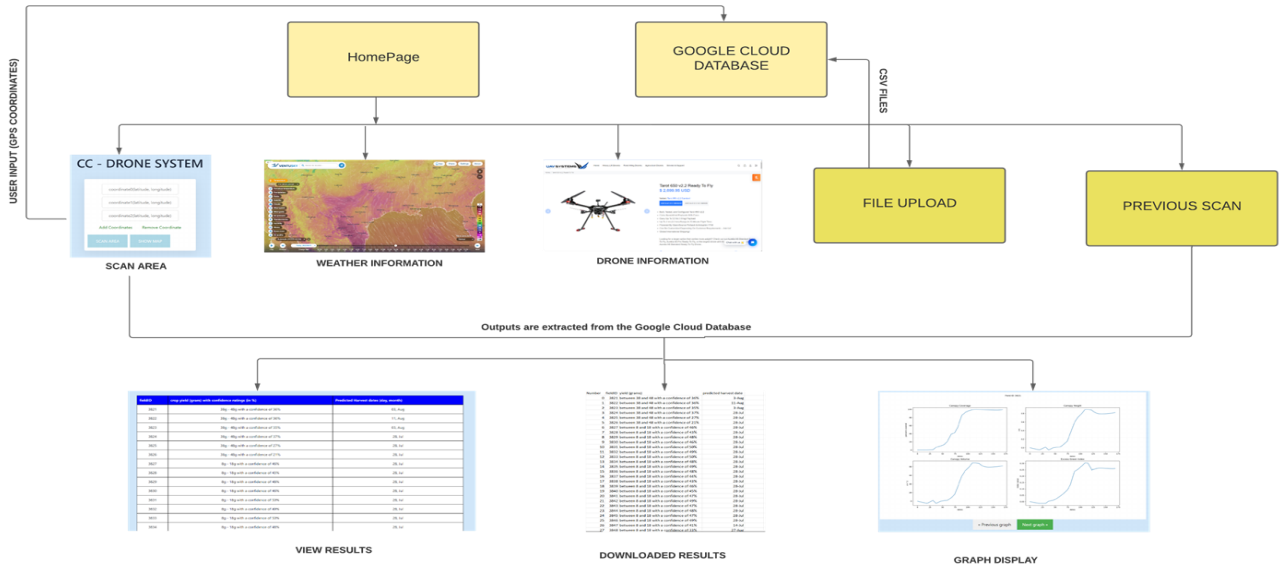


Figure 41: Functional Flowchart of the User Interface subsystem.

The frontend files (html, CSS, images) and backend files (python functions, JavaScript) are all located in the file directory where the Django app is kept. The link to the user interface is accessed through the Linux command “python3 manage.py runserver.” The

user gets access to the interface via the login and registration page. This step can be overcome if the user has previously logged in to the link. After logging in, the user is redirected to the main page, which contains the greetings, link to the user interface, and sign out link. The user can access the link to the user interface from the main page. In the User Interface, the user inputs GPS coordinate in decimal degrees format for their desired field. Once the user clicks the submit button, the UI's backend code receives the user input and redirects the user to the 'drone output' page where the ML derived results are displayed.

6.3. Subsystem Validation

6.3.1 Security and Privacy Validation

The first step in the validation plan is to ensure the user's security and privacy. The webapp utilizes sign-in and registration pages, and session variables to ensure that no one has access to the UI apart from the user. It also tailors the UI directly to the user's experience.

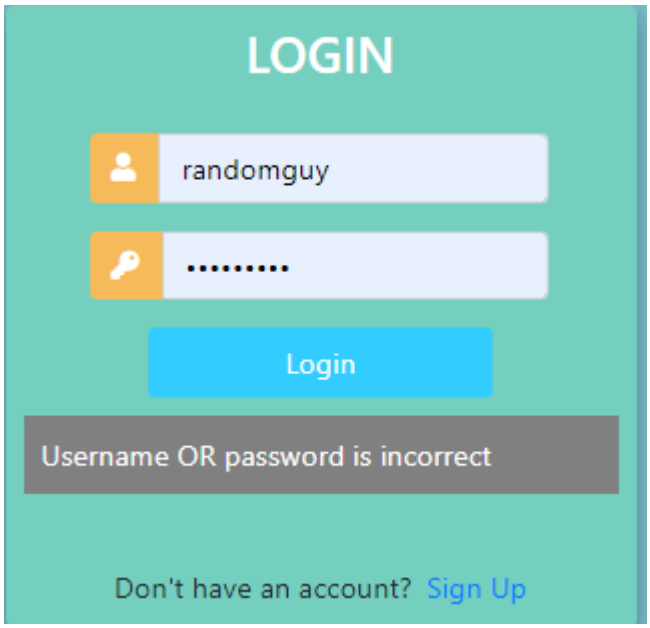


Figure 42: The Login Page

The Login page is validated by iterating through different account fields in the Django admin page and verifying if the character field the user typed in is correct. If the user input is correct, it redirects the user to the main page, if it is not, it throws an error

message. An example of such verification is shown in Figure 42. If the user does not have an account on the webapp, there is a link on the signup page, where the user can be redirected to.

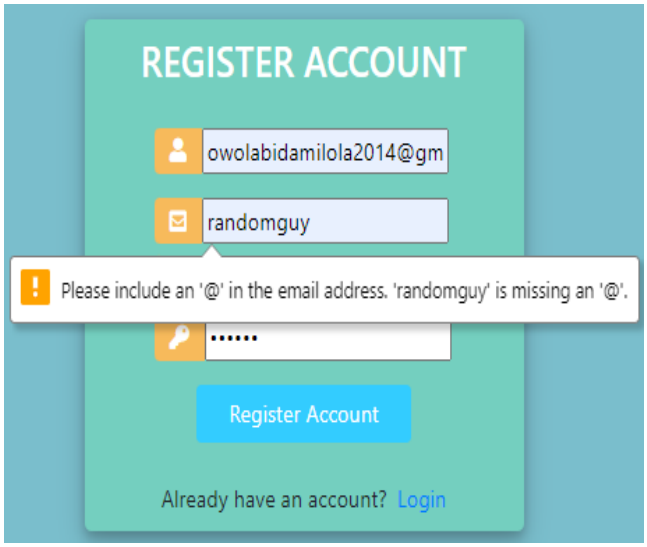


Figure 43: The Registration Page

The Registration Page performs some error checks to receive valid input from the user. A similar example is shown in Figure 43 above, where an error message is thrown when the user types in an incorrect email address. It also performs various other validations, such as: checking for empty input fields, checking the password and its verification input is the same, and lastly checking if the password or username is similar to any previous existing input.

Privacy is also a key factor to consider when creating a website. Whenever a user registers a new account and logs in, the UI app automatically creates a folder named after the user's username in the database directory. Three empty subdirectories named; 'graphs/,' 'pictures,' and 'reports' are generated in the newly created user-specific directory, an example is shown in Figure 44 below.

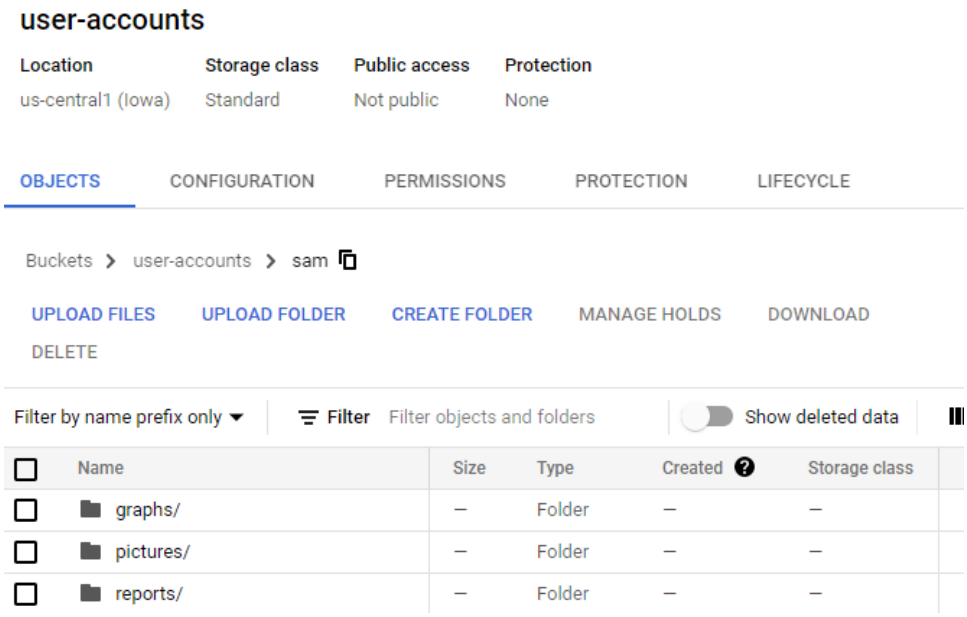


Figure 44: Proof of generated subdirectories

This is done is to ensure that the website does not access information from a different user other than the one currently logged in. It also acts as a designated location for the files, pictures and data that will be generated by the by the ML and the pi-camera.

6.3.2 GPS input Validation

After signing in, the UI can be accessed from the main page, where the user is initially redirected to. Various JavaScript functions were written and implemented to ensure valid user inputs.

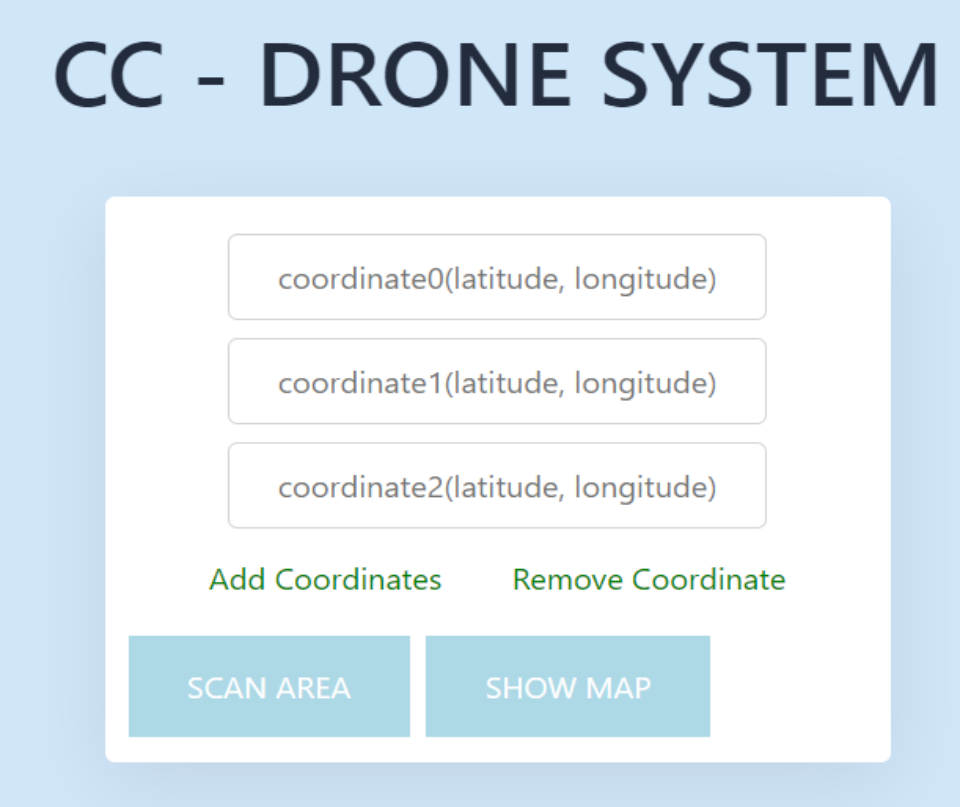


Figure 45: The User Interface

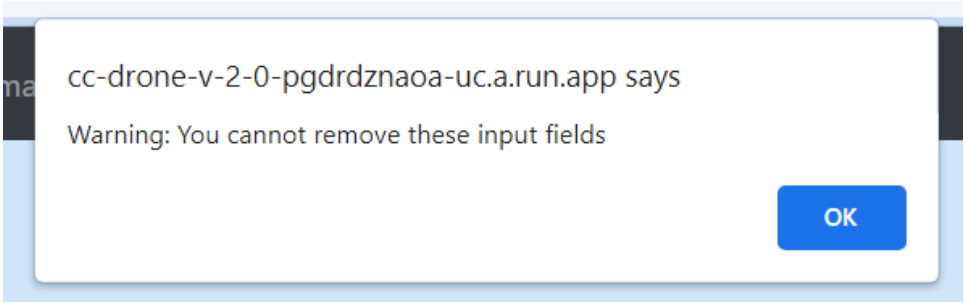


Figure 46: JavaScript Popup Message

The drone requires 3 - 6 inputs from the user to start scanning the input field. There are two buttons called “Add Coordinates” and “Remove Coordinate” in the UI, which allows the user to add or remove coordinate fields as needed. An example is shown in figure 45 above. The user must enter a minimum of three and a maximum of six coordinate points. Anything less than 3 or more than 6 will result in a popup error message like the one shown in Fig. 46 above.

```python
#function to verify coordinates
def verify_inputs(input):
    #print("testing ..... gps coordinate should be in the format lat, long")

    #checking if input is an empty string
    if (input == 'None'): #checks if input is empty
        #print("input == none")
        return

    #testing to see if format is correct
    try: #checks if input is in the correct format
        inputArray = input.split(", ")
        lat = float(inputArray[0])
        long = float(inputArray[1])

        #test cases for range
        if lat >= -90.00 and lat <= 90.00 and long >= -180.00 and long <= 180.00: #checks if input is in the correct range
            return True

        else:
            return False

    except:
        return False
```

Figure 47: Validity Check Test Cases

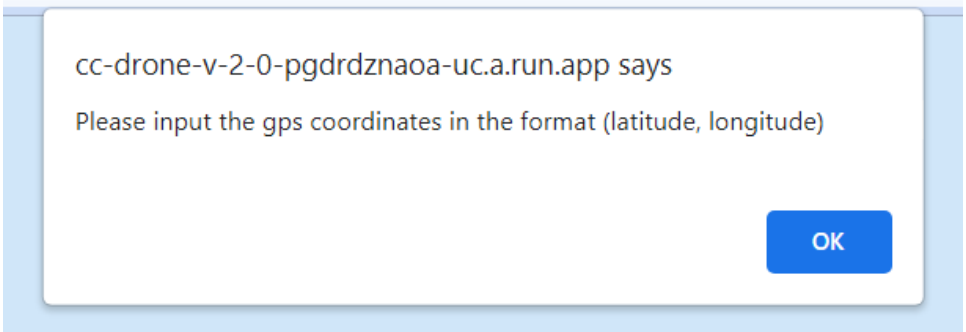


Figure 48: Invalid Input format pop up message

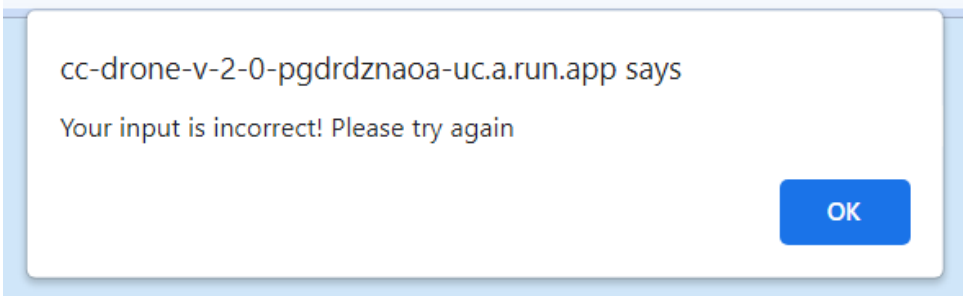


Figure 49: Incorrect input pop up message

There are also input validation testcases that are written, to check if the GPS inputs are correct. Once the user clicks the “SCAN AREA” button, JavaScript function checks the file to see if the GPS input is valid like the one shown in the function “verify_inputs” in figure 49. Validity checks looks through the user input for valid inputs like; if input is in the format <Longitude, Latitude> checking if the input is not a string, checking for invalid input string like non-integers, symbols, etc., and checking if the input is in between the boundary values ‘-180’ and ‘180’ for longitude and ‘-90’ and ‘90’ for latitude. If any of the test cases fails, the UI throws popup error messages like the ones shown in Figure 48 and 49.

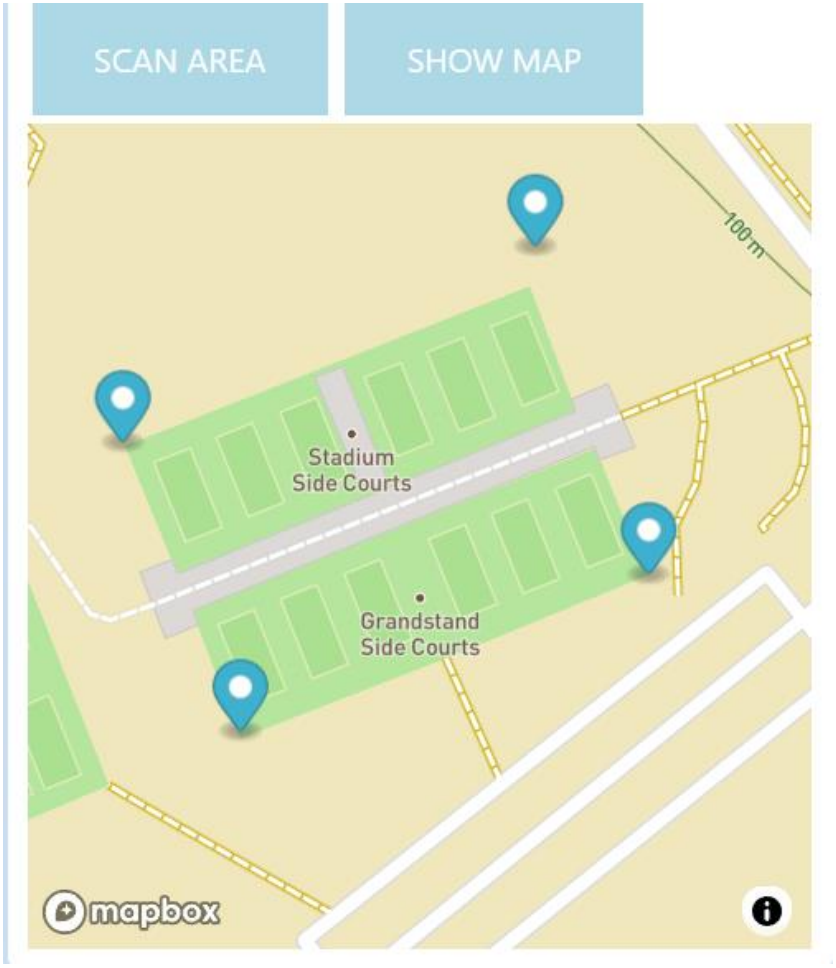


Figure 50: Map showing Texas A&M’s George P Mitchell Tennis Center with Pop up Markers

There is also a ‘show map’ button in the user input webpage, which primary function is to help the user visualize the GPS coordinates of the field area they are attempting to scan. An example can is shown in Figure 50 above, where there are popup markers displaying

the boundaries of a designated location. This is done using ‘MapBox API,’ an online JavaScript tool that help in the application of map related software uses, at no charge to the user.

6.3.3 Database Interaction Validation

Collected GPS coordinates from the user are converted to text files and sent to the google cloud database where it can be accessed by the MCU. The storage API is accessed by an API key and other parameters required by the google cloud services. All this information is stored in the Json file, which is accessed by the python function shown in figure 51 below.

```
##### NEW TEST ####
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = 'UI/shared_gcs.json'
storage_client = storage.Client()
dir(storage_client)
myBucket = storage_client.bucket('user-accounts')
my_bucket = storage_client.get_bucket('user-accounts')

#this function will take the name of a file, the location of the file, and the desired destination
storage_client = storage.Client()

# it will then upload the file onto the desired bucket in the cloud
bucket = myBucket
blob = bucket.blob(userName + '/' + filename)
blob.upload_from_filename(filename)
return render(request, 'UI/droneOutput.html',{})
```

Figure 51: File Upload Codes

If the file is uploaded successfully, it redirects the user to the output page where they can download the MCU generated files. If the process fails, the user is redirected back to the UI.

Proof of a successful upload is shown in Figures 52 & 53 below.

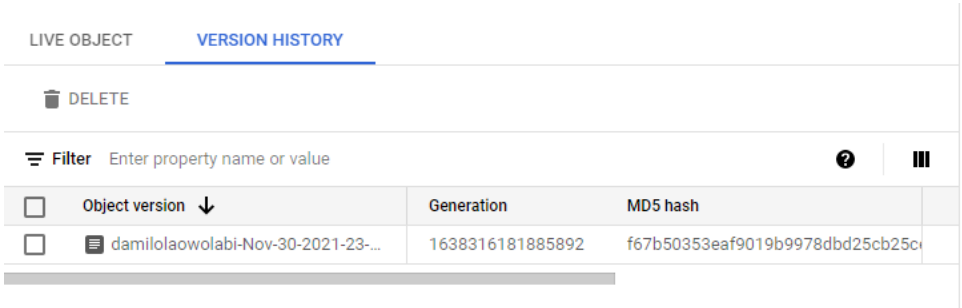


Figure 52: The uploaded file in google cloud services

```
1  StartingPoint(1.00, 1.00)
2  A(ad,cd. df,02)
3  B(0.01,0.01)
4  C(0.00, 0.01)
5  D(0.00,0.00)
6  E(1.02. 1.01)
```

Figure 53: Example of generated text

6.3.4 ML Data Extraction Validation

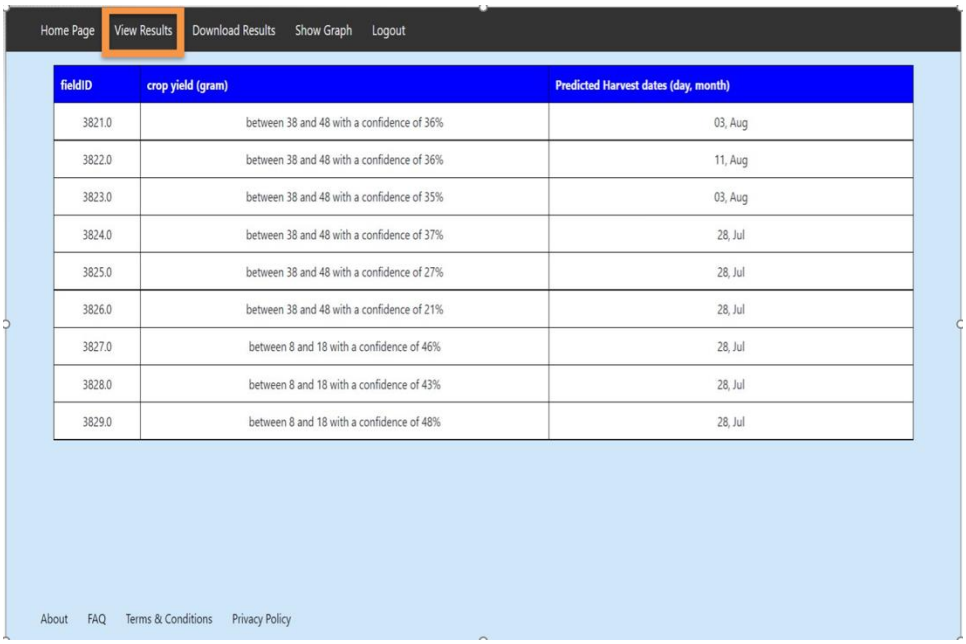
One of the requirements for the User Interface subsystem, is the ability to extract generated results from the system and display it on the webpage so the user can view it. As we can see from the subsystem's functional flowchart in figure 53 above, once the user input the GPS coordinates, they are redirected to the output page where they can view the ML results, the ML graphs, and download the results as a csv file.

ML generated data are extracted from its specified database location using Django's python backend codes like the ones shown in Figure 54 below. ML data are extracted as strings and managed differently depending on how the user wants to view the data. Either by viewing them or downloading them.

```
user =  str(request.user) #getting username
PATH = os.path.join(os.getcwd() , 'UI/shared_gcs.json')
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = PATH
storage_client = storage.Client(PATH)
storage_client
bucket = storage_client.get_bucket('user-accounts')
filearray = [filearray.name for filearray in list(bucket.list_blobs(prefix='')) ]
response = HttpResponse()
response['Content-Disposition'] = 'attachment; filename=' + user + '_reports_report_2022.csv'
```

Figure 54: Python Function to extract the ML data

The results from this process are displayed in figures 55 and 56 below.



fieldID	crop yield (gram)	Predicted Harvest dates (day, month)
3821.0	between 38 and 48 with a confidence of 36%	03, Aug
3822.0	between 38 and 48 with a confidence of 36%	11, Aug
3823.0	between 38 and 48 with a confidence of 35%	03, Aug
3824.0	between 38 and 48 with a confidence of 37%	28, Jul
3825.0	between 38 and 48 with a confidence of 27%	28, Jul
3826.0	between 38 and 48 with a confidence of 21%	28, Jul
3827.0	between 8 and 18 with a confidence of 46%	28, Jul
3828.0	between 8 and 18 with a confidence of 43%	28, Jul
3829.0	between 8 and 18 with a confidence of 48%	28, Jul

Figure 55: ML results view function in the UI

Number	fieldID	yield (grams)	predicted harvest date
0	3821	between 38 and 48 with a confidence of 36%	3-Aug
1	3822	between 38 and 48 with a confidence of 36%	11-Aug
2	3823	between 38 and 48 with a confidence of 35%	3-Aug
3	3824	between 38 and 48 with a confidence of 37%	28-Jul
4	3825	between 38 and 48 with a confidence of 27%	28-Jul
5	3826	between 38 and 48 with a confidence of 21%	28-Jul
6	3827	between 8 and 18 with a confidence of 46%	28-Jul
7	3828	between 8 and 18 with a confidence of 43%	28-Jul
8	3829	between 8 and 18 with a confidence of 48%	28-Jul
9	3830	between 8 and 18 with a confidence of 46%	28-Jul
10	3831	between 8 and 18 with a confidence of 50%	28-Jul
11	3832	between 8 and 18 with a confidence of 49%	28-Jul
12	3833	between 8 and 18 with a confidence of 50%	28-Jul
13	3834	between 8 and 18 with a confidence of 48%	28-Jul
14	3835	between 8 and 18 with a confidence of 49%	28-Jul
15	3836	between 8 and 18 with a confidence of 48%	28-Jul
16	3837	between 8 and 18 with a confidence of 44%	28-Jul
17	3838	between 8 and 18 with a confidence of 43%	28-Jul
18	3839	between 8 and 18 with a confidence of 46%	28-Jul
19	3840	between 8 and 18 with a confidence of 45%	28-Jul
20	3841	between 8 and 18 with a confidence of 47%	28-Jul
21	3842	between 8 and 18 with a confidence of 49%	28-Jul
22	3843	between 8 and 18 with a confidence of 47%	28-Jul
23	3844	between 8 and 18 with a confidence of 48%	28-Jul
24	3845	between 8 and 18 with a confidence of 47%	28-Jul
25	3846	between 8 and 18 with a confidence of 49%	28-Jul

Figure 56: Downloaded CSV file from the UI

6.3.5 ML Graph Display Validation

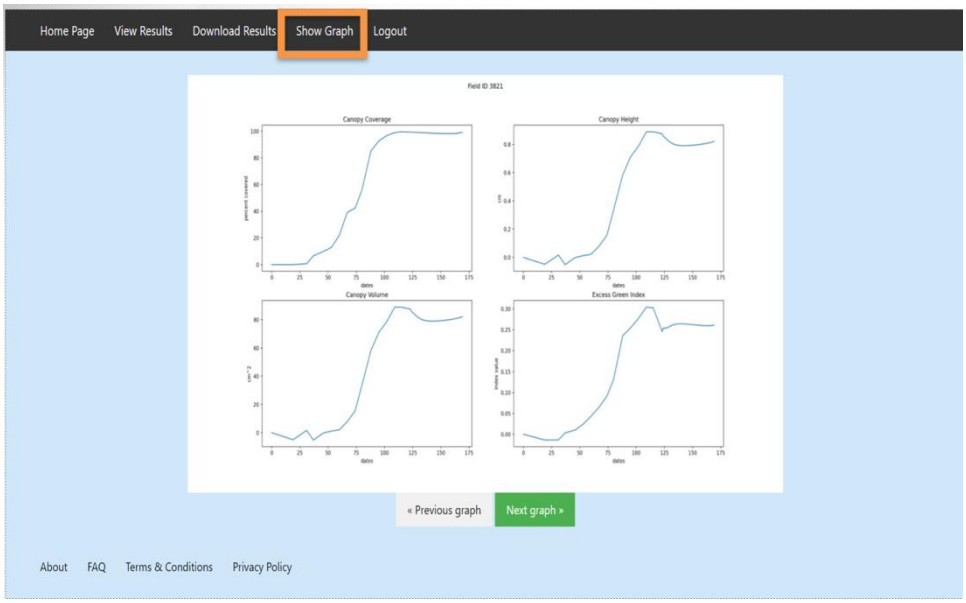


Figure 57: Graph Display in the UI

ML generated graphs are extracted from the database are displayed individually in the UI. An average of one hundred graphs are generated by the ML for each scan done by the user, hence the need to display them individually. The user is given the option to iterate through the graphs using the ‘previous graph’ and the ‘next graph’ button as shown in figure 57 above.

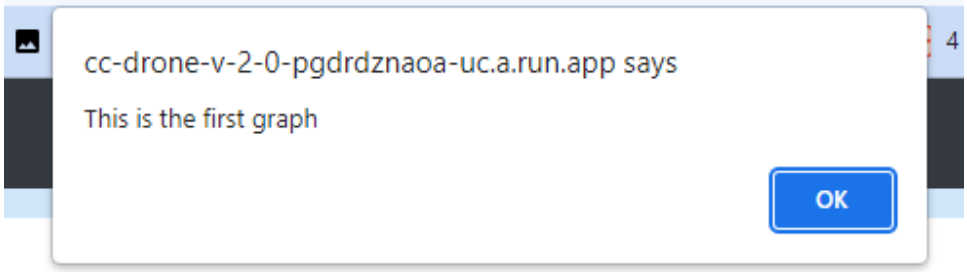


Figure 58: Popup message for graph display

There are validation checks in place, to ensure that the user is not accessing the wrong file. A popup message like the one shown in Figure 58 is generated whenever the user tries to access a graph prior to the first graph, or a graph after the last graph.

6.3.6 Multiple Connection Validation

The UI app was successfully deployed to the Google Cloud Run via the Django framework, where it will be run on the Google server instead of the prior local host server. For every change made to the app, the app must be deployed for those changes

to be reflected in the cloud run. An example of a successful deployment is shown in Figure 59 below.

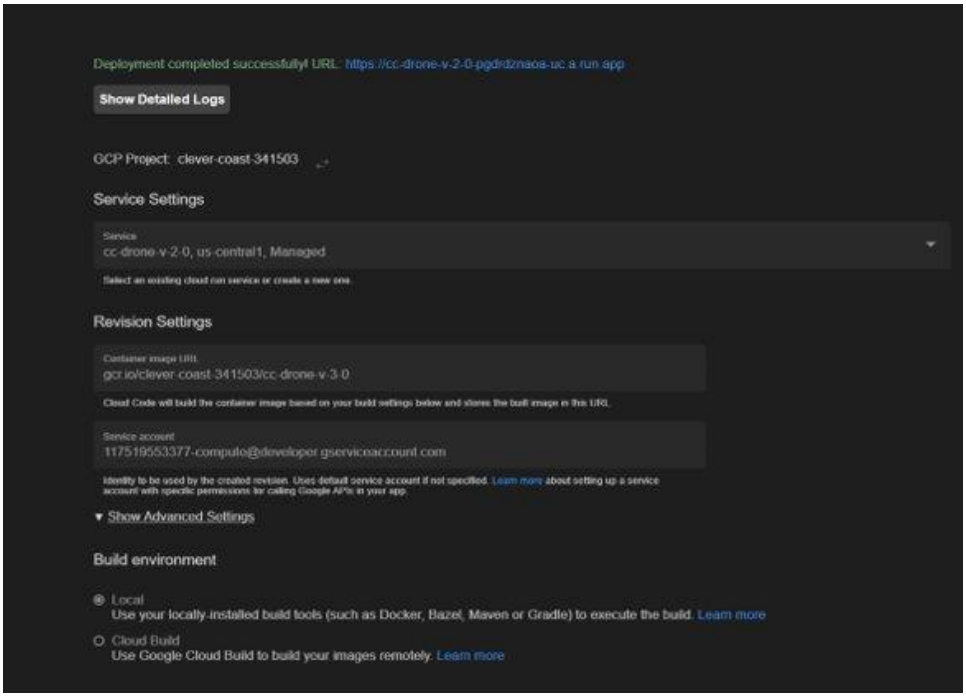


Figure 59: Successful Deployment Message

Benefits of deploying to the cloud run includes multiuser access, multi-screen display, etc. Figure 60 below shows an example of two different users accessing the app at the same time.

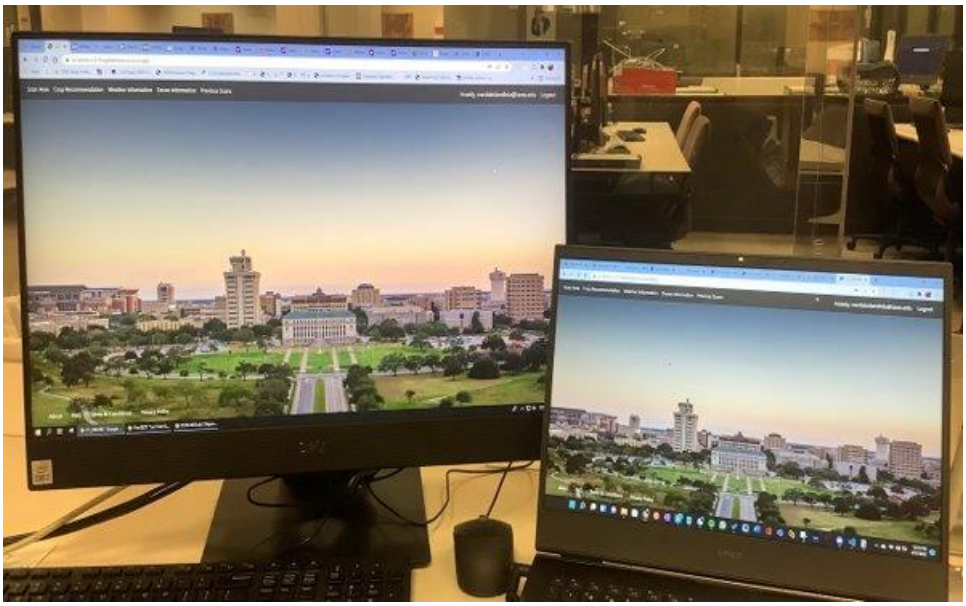


Figure 60: Proof of Multiple User access

6.4 Subsystem Conclusion

The User Interface subsystem was successfully able to prove user's security and privacy, validate that the user inputs are valid GPS coordinates, ensure successful file upload and download, and provide an unforgettable user experience in the process. It was able to successfully integrate with the other subsystems to generate a full working system.

7. Final System Report

7.1. System Introduction

This is the final system validation for the cc-drone. It encompasses the integration of the Machine Learning, Flight Control and Image Capture, Drone Enclosure, Data Management and Architecture and the User Interface subsystems. The tests helped proved that the system is fully integrated for the CC_DRONE project. This was done by the team effort of group members: Damilola Owolabi, Samuel Fabricant, Luis Sanchez, and Justin Whitehead. The full system code is located in the GitHub link: https://github.com/DamilolaOwolabi/ECEN-403-GROUP-66 .

7.2. System Details

The full system overview diagram is attached below:

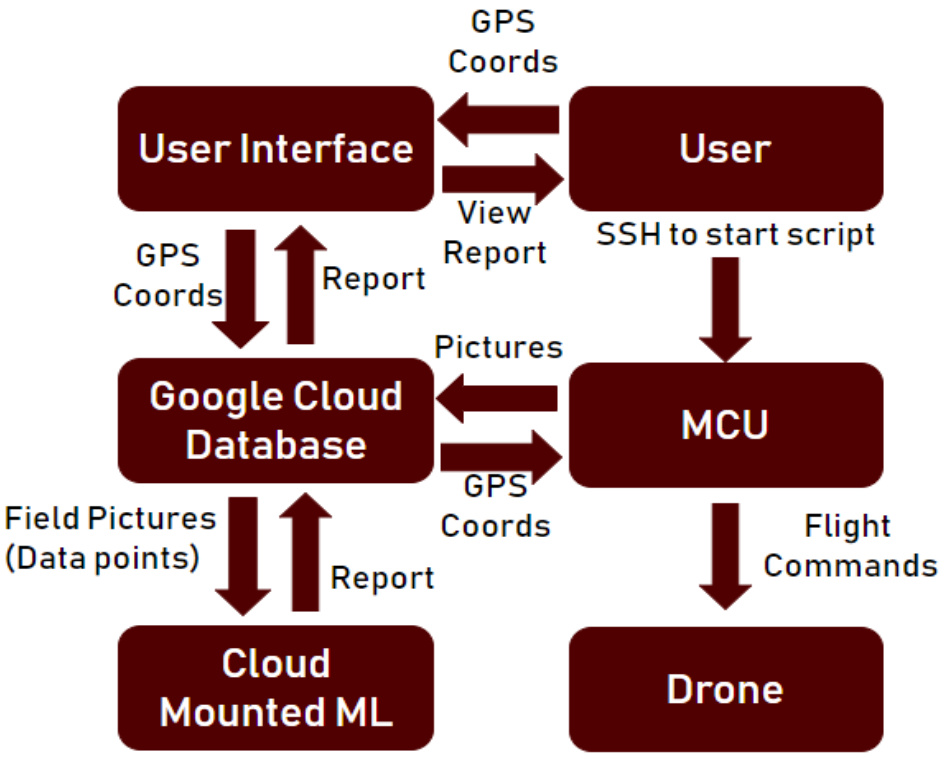


Figure 61: System overview of the Cotton Drone Analysis project

The system begins with the user making an account on the user interface app and entering the GPS coordinates of the boundaries of their specific field, which are then sent to the cloud database. When the user is ready to image their field, they SSH into the Raspberry Pi MCU, initiate the flight script, and enter their account's username. The flight script then runs a flight path algorithm, autonomously flies the determined path while imaging the field. Once finished, the drone lands and offloads images back to the database. Next, the machine learning model analyzes data points from the pictures and creates a report predicting yield and harvest date for the cotton field. Lastly, the report is viewable to the user on under their account in the user interface.

7.3. System Validation

The overall validation for the system report is shown in the video link below: https://photos.app.goo.gl/syskFH5DS3jfRvUY9 . As the video shows, the subsystems were successfully integrated and able to interact with each other via the Google Cloud database. From the video above, the user is able to successfully login into the UI, type in the gps coordinates, launch the drone in the field, and view the results in the UI.

7.4. System Conclusion

The Machine Learning, Flight Control and Image Capture, Drone Enclosure, Data Management and Architecture and the User Interface subsystems were proven to be successfully integrated. The system has proven it main purpose of automating the data extraction and analysis process. The user just has to perform 2 actions, typing the gps coordinates and launching the drone. Therefore, saving the user both time and energy.