

AI-Integrated Autonomous Drone System for Smart Crop Health and Harvest Monitoring

sdmay26-20

Md Maruf Ahamed
Frontend: Caleb Zea, Z Harvey
Backend: Alex Pallan, Srdan Kopunovic
Hardware: Camden Senneff, Andres Lopez
AI: Jongmin Lee

Email: sdmay26-20@iastate.edu
Design website: https://sdmay26-20.sd.ece.iastate.edu/

Executive Summary

Farming is a challenging life, but in 2026, technology can assist by providing additional information to farmers so they can make the best decisions going forward. If farmers aren't harvesting at peak efficiency, that is lost food that's heading for people or animals. Our project allows farmers to monitor their crop health and predict harvest times using an AI-integrated autonomous drone system that uses a desktop app for user interfacing. There are other companies out there that utilize AI to assist farmers, mostly with



satellite imagery and some drone usage, but ours is the only one to specifically use a drone while also maintaining an affordable cost. Our project is more budget friendly which lowers the barrier to entry and helps more farmers to mitigate their crop losses.

For the design requirements, we were primarily limited by budget. We had to build an autonomous drone from scratch using off-the-shelf parts as prebuilt drones were incompatible with the functionality required to meet project goals. We were able to accomplish this using only $750. Functionality wise, our project needed to record field imaging for crop health tracking that would use our own AI model to give the user information about their crops. We needed to ensure that the system could work without the internet, so the system is not reliant on consistent online connection. Lastly, we needed to create a desktop application that is farmer-friendly that allows flight planning, telemetry monitoring, and data visualization.

Our project integrates custom hardware, middleware, and software to achieve the solution. The drone itself utilizes INAV, sensors such as a magnetometer, barometer, GPS, and a camera to take images of the fields with. When the mission has completed, the data is transferred onto an SD card to transfer images to the desktop application. On the software side, the frontend uses React, TypeScript, Electron, and uses the Google Maps API for field generation. The backend relies on Python, AWS, and Terraform for infrastructure and data management, and then uses Cognito for secure user authentication. The AI microservice uses Python to execute our custom-trained machine learning model that can analyze images and provide information about crop health and harvest readiness to the user.

Overall our design meets most of the requirements, but doesn't have complete functionality. We have all of the frontend and backend components working and communicating with each other, as well as the drone receiving information from the app, taking pictures and sending those to the backend to be analyzed. Our drone has completed an autonomous flight with a mission planner from INAV however it has not been tested to take fly autonomously from the Raspberry Pi. Additionally, our AI model is working, but we aren't certain on the accuracy of it as we haven't been able to train the model on drone pictures at a proper altitude, we have had to use existing or public datasets. Our design itself addresses the needs of the user, it has an easy to use interface, the drone is autonomous, and its cheap. Unfortunately we weren't able to talk with farmers like we wanted to, so we can't be certain of it's real-world usability.

The next steps would be to test the drone flight loop functionality, once this is accomplished, everything else falls into place. We would be able to provide our AI model with significantly more accurate and recent data to train itself, allowing us to give better insights to the farmer about their crops. The software is already in place and then we'd be able to ask farmers their opinions on the system as a whole, allowing us to better understand and make changes for our users.



Learning Summary

Development Standards & Practices Used

Software practices: Automated Testing, Code Reviews, Version Control w/ Git, Defensive Programming & Error Catching, not using global variables, CI/CD, Code Comments, Refactoring often, Iterative and Incremental development, Loose coupling,

Hardware: Flywoo Explorer LR 4 O4 Pro, Raspberry Pi Zero, Arducam 12.3 MP IMX477 Camera Module, FW10 Pro GPS Module with Compass 18650 3000mAh 4S Li-ion/850mAh Li-po 3S Batteries, RadioMaster Pocket Portable Remote Controller.

Circuit: The Pi and GPS/Compass is wired to the FC on the quadcopter for power and UART. RGB circuit on Raspberry Pi Zero.

Summary of Requirements

- Intuitive user experience for limited technical users
- Open field environment
- Portable hardware (e.g. laptop) with no guaranteed internet connection
- User repairability
- Autonomous drone system
- Accurate crop health and harvest monitoring
- Documentation

Applicable Courses from Iowa State University Curriculum

COMS 2270, CPRE 2880, COMS 3630, SE 3090, SE 3190, SE 3170, SE 3390, SE 4210, SE 1850, COMS 2280, SE 3290, SE 4170, SE 4090, SE 4910, SE 4920, CPRE 1850, EE 2010, EE 2300, ENGL 3140, COMS 2520, CPRE 4880, CPRE 3080

New Skills/Knowledge acquired that was not taught in courses

List all new skills/knowledge that your team acquired which was not part of your Iowa State curriculum in order to complete this project.

- Software such as Electron, the Google Maps API, React, Python, TypeScript, AWS, Terraform, Cognito, C++, INAV/MSP, libcamera, CMake, GoogleTest
- Hardware issues such as calibrating sensors and PID as well as working with new hardware components like a quadcopter, Raspberry Pi and Arducam
- Being prepared for worst case scenarios when testing hardware

Table of Contents

Executive Summary **1**
Learning Summary **2**
Development Standards & Practices Used 2
Summary of Requirements 3
New Skills/Knowledge acquired that was not taught in courses 3
1. Introduction **7**
1.1. Problem Statement 7
1.2. Intended Users 7
2. Requirements, Constraints, and Standards **8**
2.1. Requirements & Constraints 8
Functional Requirements 8
User Experience Requirements 8
UI Requirements 8
Database Requirements 9
AI/ML Requirements 9
Resource Requirements (Constraints) 10
2.2. Engineering Standards 10
1. IEEE 1936.1-2021 – Drone Applications Framework 13
2. IEEE 1939.1-2021 – Framework for Structuring Low-Altitude Airspace for UAV Operations .. 13
3. IEEE 14764-2021 – Software Life Cycle Processes: Maintenance 13
3. Project Plan **14**
3.1. Project Management/Tracking Procedures 14
3.2. Task Decomposition 14
Software Task Decomposition 14
Hardware Task Decomposition 16
3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria 16
I. Overview 17
II. Major milestones 17
Table 3.3.II: Major Milestones 18
III. Metrics & Progress 18
Drone 18
Database & API 18
AI Microservice 18
Desktop Application 18
Team Metrics 18
IV. Evaluation Criteria 18
V. Iteration and refinement 18
3.4. Project Timeline/Schedule 19
Figure 3.4 Project Timeline 19
3.5. Risks and Risk Management/Mitigation 19

Table 3.5 Task Risks Management/Mitigation....20
3.6. Personnel Effort Requirements....22
Table 3.6.i: Projected estimate of personnel effort requirements....22
Table 3.6.ii: Actual personnel effort requirements....23
3.7. Other Resource Requirements....23
Hardware Requirements....23
Testing Requirements....24
Cloud Requirements....24
User Hardware Requirements....24
4. Design....24
4.1. Design Context....24
4.1.1 Broader Context....24
Table 4.1.1: Broader context in which the design problem is situated....25
4.1.2 Prior Work/Solutions....25
Electronic Documents....26
Professional Internet Site....26
General Internet Site....26
4.1.3 Technical Complexity....27
Flight Control and Navigation....27
Imaging and AI Vision....27
Backend Processing and Cloud Integration....27
Frontend Dashboard and Visualization....27
Power and Hardware Optimization....28
Problem Scope and Industry Challenge....28
4.2. Design Exploration....28
4.2.1 Design Decisions....28
4.2.2 Ideation....29
4.2.3 Decision-Making and Trade-Off....30
Table 4.2.3: Drone options and pros and cons for each ideated options....30
4.3. Final Design....31
4.3.1 Overview....31
4.3.2 Detailed Design and Visual(s)....31
Figure 4.3.1: High-level system overview....31
Figure 4.3.2: Autonomous drone system diagram....32
4.3.3 Functionality....32
4.3.4 Areas of Challenge....33
4.4. Technology Considerations....34
5. Testing....34
5.1. Unit Testing....34
5.2. Interface Testing....35
5.3. Integration Testing....35
5.4. System Testing....36
5.5. Regression Testing....36

5.6. Acceptance Testing 37
5.7. Security Testing 37
5.8. User Testing 38
5.9. Testing Results 38
6. Implementation 39
Figure 6.1: User Sign in 40
Figure 6.2: User Create Account 40
Figure 6.3: Homepage 41
Figure 6.4: Crop carousel page, user can see crop pictures taken by drone here 42
Figure 6.5: (Top) and Figure 6.6: (Right) Flight Creation and outlining thanks to Google Maps API 42
Figure 6.7: Saved field viewing & flight scheduling screen, with flight data information at the bottom of the screen 42
Figure 6.8 Complete drone build 44
Figure 6.9: Pilot software running on the X-Plane simulator 44
Figure 6.10: Validation confusion matrix for prototype YOLOv8 6-class corn growth model 45
6.1. Design Analysis 45
7. Ethics and Professional Responsibility 46
7.1. Areas of Professional Responsibility/Codes of Ethics 46
Table 7.1: Table for areas of professional responsibility 47
7.2. Four Principles 48
Table 7.2: Table of the four principles 48
7.3. Virtues 49
8. Conclusions 53
8.1. Summary of Progress 53
8.2. Value Provided 53
8.3. Next Steps 53
9. References 53
10. Appendices 54
10.1. Appendix 1 – Operation Manual 54
Development Setup 54
Setup 54
Backend 54
AI Service 55
Frontend 55
Drone Pilot Software 55
Production Setup 56
Prerequisites 56
Steps 56
XPlane Simulation 57
Prerequisites 57
Setup 57
Drone Configuration 57



Drone Operation..... 58
SD Setup..... 58
Controller Mappings..... 58
Drone Use..... 58
10.2. Appendix 2 – Alternative/initial version of design..... 59
10.3. Appendix 3 – Other considerations..... 59
10.4. Appendix 4 – Code..... 59
10.5. Appendix 5 – Team Contract..... 59
Team Members..... 60
Required Skill Sets for Your Project..... 60
Skill Sets covered by the Team..... 60
Project Management Style Adopted by the team..... 61
Initial Project Management Roles..... 61
Team Contract..... 61

1. Introduction

1.1. Problem Statement

The goal of this project is to develop an unmanned drone capable of assisting farm owners in inspecting and evaluating their crops. Our drone collects data by scanning the fields and then utilizes a vision model to compare the gathered data with other information to identify any issues with the crops, such as pests or diseases.

While there may be other companies that have made similar products, we wanted to show that it is possible to create a product that can accomplish similar tasks on a cheaper budget. Other companies have created a similar platform, but require farmers to purchase both their own drone as well as a subscription service, in order to use their proprietary app. We wanted to make a product that can be created using basic off-the-shelf parts, as well as a desktop application for ease of access.

1.2. Intended Users

The three main user groups for this drone are farmers, insurance companies, and farming supply companies. Farmers are likely to use this drone to monitor their own crops to avoid loss of income and to try to catch potential issues sooner rather than later. Farmers want a user interface that is simple to manage and understand, while still allowing them complete control over the management of the drone and understanding of the information that it provides. Insurance companies can see this drone as a way to avoid/mitigate damages. Since any damage that may happen to a farmer’s crops could potentially result in a policy payout for the insurance company, they can seek to utilize our drone to help mitigate risk.



This can also benefit farmers who use our drone, because if the insurance companies have confirmation that they are using our technology to monitor their fields, they may receive a better deal on their agriculture insurance policies. One final group that may use our drone is farming supply companies that may seek to privatize this idea. If this idea performs well, different agriculture companies may try to create their own versions of our software for their own private sale. They would look for effective, functional work from the drone, and would be looking to see how it performs and how accurately and efficiently the vision model performs when identifying these pests and diseases, as well as seeing how soon we can detect these issues, and if they can be stopped before they can do any real damage. As more companies enter this market, the number of options available to farmers will increase, creating added incentive and benefits for both companies and consumers.

2. Requirements, Constraints, and Standards

2.1. Requirements & Constraints

Functional Requirements

1. Plant Growth Tracking – Week-by-week imaging to monitor crop development.
2. Autonomous Drone Flight – Pre-programmed routes with GPS and obstacle avoidance.
3. Insect Detection – AI-based identification of pest infestations.
4. Harvest Time Monitoring – Visual and data-based estimation of crop maturity.
5. Farmer Dashboard – Centralized platform to outline and select fields, create flights, and send data to be analyzed
6. Reliable predictions of harvest readiness.
7. Server data storage, user authentication and authorization for security

User Experience Requirements

1. Ability for users to set up and configure drone flight through a cross-platform desktop application.
2. User-friendly UI that enables flight planning, field management, and data transfer after the field scan is completed.
3. Generating reports after each field scan for identified crop diseases, as well as crop growth over time.
4. Allows users to save previous field configurations and reuse them
5. Ability to use the application when not connected to the internet, as well as having data sync when a connection is established.



UI Requirements

1. Desktop application. The UI must be simple enough for any farmer to use, especially those unfamiliar with technology.
2. The UI must be robust enough to display unique field sizes and shapes, and a scheduled flight time at the farmer's selected intervals. The central dashboard should allow farmers to see all important information quickly.
3. The UI should be aesthetically pleasing with a color theme, while also being bright enough to be visible on a laptop screen in sunlight.

Database Requirements

1. PostgreSQL Database engine
2. Cloud storage for backup and offloading local storage
3. Data to store:
    a. Field:
        i. Vertex Coordinates
        ii. Intersection Coordinates
        iii. Name
    b. Flight:
        i. Date, Time, and Timezone
        ii. Flight Data
        iii. Name
    c. Flight Data:
        i. Start Battery Percentage
        ii. End Battery Percentage
        iii. Start Time
        iv. End Time
        v. Altitude
    d. Intersection Data:
        i. Unprocessed Image
        ii. Processed Image
        iii. Analysis Result
        iv. Timestamp

AI/ML Requirements

1. The training dataset includes corn images across various growth stages, with diverse lighting conditions, stress, and environmental conditions.
2. Each image or image tile in the data set must be labeled according to the standardized annotation guide. The labels include:

    Current Requirements

    a. Classifying six distinct biological growth stages of corn (Bare Soil, Early Growth, Mid Growth, Canopy Closure, Tasseling, and Senescence) and one Background class to filter out non-agricultural noise (roads, equipment, weeds).



   b. Instead of counting leaves, the model uses Excess Green (ExG) index algorithms to estimate growth by calculating the proportion of green plant pixels versus bare soil.
      Removed Requirements
   c. Insect detection
   d. Precise V-stage leaf counting
   e. Specific weed/disease classification
3. The dataset must be split into train/validation/test sets with no field or time overlap to avoid leakage.
4. Model training must be reproducible with fixed seeds, versioned datasets, and logged hyperparameters.
5. The model should provide explainable outputs to highlight regions influencing predictions.
6. All datasets must follow licensing and privacy requirements, used only for research/educational purposes and PII-free.
7. The system should include an active-learning loop to flag low-confidence samples for human review and retraining.

Resource Requirements (Constraints)

1. The drone should cost less than $750 (at the time of build).
2. The drone should be able to be built with readily available consumer parts.
3. The drone controller software should work with any standard INAV drone configuration.
4. Drone design should be able to gather a reasonable amount of data within energy constraints (the average field size of 350 acres on one charge) and support pausing and resuming data collection to recharge the drone for larger fields.

2.2. Engineering Standards

Engineering standards reinforce requirements and safety in creating designs. They are important to adhere to, as they can guide a project when followed, while also cautioning against hazards to keep not only engineers designing it safe, but also users, non-users, and the environment. They also exist to maintain complicity with regulations, morality, and the law.

Four IEEE standards that we found to apply to our project are as follows:

1. IEEE Standard for Drone Applications Framework
   IEEE 1936.1-2021
   Description: This standard defines all things related to drones and their potential functions. It defines all of the potential systems that drones with different use-cases would have on board, and provides categories for each different type of drone and how they are defined. Our project most likely fits under the “Agriculture” definition for our application scenario.

2. IEEE Standard Adoption of Moving Picture, Audio and Data Coding by Artificial Intelligence (MPAI) Technical Specification Connected Autonomous Vehicle-Architecture (MPAI-CAV) V1
   IEEE/MPAI 3307-2024

Description: This is a standard defining various aspects of an autonomous vehicle that uses AI/ML models to navigate and interpret its environment. It defines the layouts of the sensors, as well as their types. Additionally, it defines different models for how the AI components interact with the other parts of the vehicle, as well as interaction with humans

3. IEEE Standard for a Framework for Structuring Low-Altitude Airspace for Unmanned Aerial Vehicle (UAV) Operations
   IEEE 1939.1-2021
   Description: This is a standard for structuring the airspace itself for the drone. This is a more zoomed-out view from the above standard (IEEE 1936.1-2021), which is more about the drone itself. This standard defines how an airspace can be interpreted and split into regions or grids, as well as logistics regarding operating the drone. This standard, in particular, is useful because it also covers creating a mosaic of the landscape, which is a component of our project.

4. IEEE Standard for System and Software Verification and Validation
   IEEE 1012-2016
   Description: This standard defines the processes and documentation needed to verify that a system is built correctly and validate that it meets its intended purpose. It provides structured methods for testing, quality assurance, and performance assessment across all stages of software and system development. This standard is relevant to our project's vision model, as it guides how we verify that our AI pipeline (data preprocessing, training, inference) functions as intended and validate that the model's predictions—such as pest detection, crop growth, and harvest readiness—accurately reflect real-world field conditions. Following this standard ensures systematic testing, measurable performance metrics, and traceable validation evidence for our AI-based crop monitoring system.

After reviewing the technical details of these four standards, this is how they apply specifically to our project:

1) The first standard (IEEE 1936.1-2021) directly relates to our design when section five of the standard comments on different drone application types. In our case, it is information collection, transmission, and processing. There are subsections within this that also directly relate to our case, which is using the drone system to obtain target data using artificial intelligence to classify and analyze the data. The operator should also be able to monitor the drone and correct changes in flight.

2) The second standard (IEEE/MPAI 3307-2024) also somewhat correlates with our project, as it discusses the interaction of humans with connected autonomous vehicles. The standard states in section 5.1 that a CAV is defined as a couple of functions that our project will include. The CAV must converse with humans; in our case, a flight path must be taken based on user specifications. The standard also includes that the CAV must plan a route to reach the requested



destination, and it will autonomously reach it by taking a few logical steps that are not included in our design specifications, such as building a complete digital representation of the environment.

3) The third standard (IEEE 1939.1-2021) is extremely relevant to our design in that it covers standards for unmanned aerial vehicles (UAVs) in low altitude airspace, under a few hundred meters, which is the space our drone will be operating in. It covers the specific data structure and notation that should be used by our drone when subdividing the area where data collection is occurring. It also notes several requirements for how closely the drone should be able to fly within the designated flight plan (for example, altitude must be within 10% of the designed altitude, route curvature no greater than 3%, rate of ascent no more than 10m/s), in addition to safety requirements before the launch of a UAV. These requirements are designed to ensure that the data collected is as reasonably accurate as it can be, which is critical for the level of precision our drone requires for useful results.

4) The fourth standard (IEEE 1012-2016) directly relates to our project's vision model and AI data processing pipeline. It establishes a structured framework for verifying that our software components—such as data preprocessing, image classification, and AI inference—are implemented correctly, and for validating that the model's outputs align with real-world crop conditions. This connection is critical since our system's accuracy depends on how well the AI model identifies pest presence, crop health, and harvest readiness based on field imagery. Following this standard ensures that our model undergoes systematic testing, meets performance thresholds, and provides traceable validation evidence that supports its reliability in real agricultural environments.

After discussing with other members of the group, here are some other standards we found that apply to our project:

1. IEEE International Standard - Software engineering - Software life cycle processes - Maintenance
   IEEE 14764-2022
   Description: This is a standard describing how to perform the iterative process of development for software. The standard can be applied to any software product, and especially takes into account the maintenance of software. The criteria outlined in this standard includes planning and maintenance of software, as well as the execution of software maintenance activities. It aims to outline how software maintenance should be planned during software development, and then provides criteria to assess the effectiveness of software maintenance execution. The standard doesn't address the operation of software, backups, recovery, or system administration.

2. IEEE Standard for Information Technology– Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications
   IEEE 802.11-2024
   Description: This long-titled standard defines how wireless local area networks operate over short distances. This standard affects how devices like laptops, phones, and routers

communicate with each other while on Wi-Fi. The definitions in this standard include the Medium Access Control or MAC layer, which controls how multiple devices share a radio channel without interference with each other. The Physical Layer specifies how bits are transmitted in the air with the help of radio waves. Finally, the standard ensures that interoperability is maintained, in other words, the ability for devices from different manufacturers to be able to communicate with each other reliably. This standard is the foundation for Wi-Fi, addressing how data physically moves wirelessly and how devices work together while on the same Wi-Fi.

3. IEEE Standard for Software Engineering – Software Life Cycle Processes – Maintenance
   IEEE 14764-2022
   Description: This standard defines the processes and best practices for maintaining software throughout its life cycle. It focuses on planning, managing, and executing maintenance activities after the initial development phase, ensuring that software continues to function correctly and efficiently over time. The criteria outlined include documentation, modification control, error correction, and performance improvements. This standard applies to our project because the AI vision model and drone control software requires regular updates—such as retraining models with new crop data, fixing bugs, and optimizing performance—to maintain accuracy and reliability as environmental conditions and datasets evolve.

Some modifications we plan to make to incorporate these standards apply as follows:

1. **IEEE 1936.1-2021 – Drone Applications Framework**

This standard defines categories of drone systems and outlines the onboard subsystems required for different applications. To maintain this standard, we have:

- **Categorized our drone under agricultural operations**, following IEEE classification, and defined subsystems as navigation, imaging, and environmental sensing.
- **Mapped system architecture** (hardware components, AI components, and communication modules) to the standard's modular framework.
- **Documented system interactions** using standardized terminology from IEEE 1936.1, improving clarity for future integration with other packages.

2. IEEE 1939.1-2021 – Framework for Structuring Low-Altitude Airspace for UAV Operations

This standard provides a structured model for UAV airspace management and coordination. To maintain this standard, we have:

- **Designed the drone's flight control logic** to comply with airspace segmentation and altitude bands, ensuring safe autonomous operation in low-altitude regions.



- **Integrated a mapping mode** consistent with this framework to standardize how imagery and field data are collected and aligned to geospatial grids.

3. IEEE 14764-2021 – Software Life Cycle Processes: Maintenance

This standard defines the framework for planning and executing software maintenance. To maintain this standard, we have:

- **Established a formal maintenance plan** early in development, detailing packaging, architecture, and testing.
- **Used version control and CI/CD pipelines** to track maintenance activities and regression tests.
- **Adopted iterative review cycles** for updating both flight software and the AI processing microservice, ensuring reliability.

4. IEEE 1012-2016 – System and Software Verification and Validation

This standard establishes guidelines for verifying and validating software systems to ensure they function as intended and meet user requirements. To incorporate this standard, we have:

- Developed a structured **V&V plan** detailing verification steps (unit and integration tests) and validation procedures (accuracy benchmarking and field evaluation).
- Maintained **traceability** between requirements, dataset specifications, model configurations, and validation results.
- Used standardized **performance metrics** (e.g., precision, recall, and inference latency) as acceptance criteria for validation.
- Documented all verification and validation evidence—including test results, error analyses, and retraining outcomes—to ensure transparency and compliance with IEEE 1012 guidelines.

3. Project Plan

3.1. Project Management/Tracking Procedures

The management style we used is agile, as there were many moving parts to consider, including the frontend and backend of our project, as well as the AI component. Agile allowed us to work on multiple parts of the project at the same time, while having a group goal to ensure that we stay on track to integrate those parts together.

Progress was tracked via pushes and merge requests in the group Git repository.

Communication was made in the group Discord server, with separate channels for different sections of the project, so as not to clutter team channels with unnecessary information.

This method of communication helped the group to understand what each individual was working on, as well as understand how each part contributes to the whole.



3.2. Task Decomposition

Software Task Decomposition

1. Set up frontend and backend environments and get all versions set up, making sure that everything was compatible with each other
    a. Synced up all versions within Git
    b. Ensured that our environment worked with CI/CD
    c. Checked for incompatibilities between Linux/Mac/Windows/Android libraries
2. Split up work and started development on frontend and backend sections.
    a. Frontend began by working on desktop application and UI
    b. Backend developed database and started working on AI implementation, even if minimal
    c. Made sure there was connectivity between frontend, backend with REST, and AI program by syncing and testing on git.
        i. Information came from the backend, where AI information is stored and later displayed on the UI.
    d. Set up CI/CD tests, making future testing and deployment easier.
3. Continued development on all ends of the software
    a. Frontend fully developed the UI for the desktop application, ensuring critical requirements and functions are implemented. The farmer dashboard should be accessible at this point.
        i. Developed UI that allows farmers to see flight information, drone information, and crop information, and integrated the Google Maps API
        ii. Ensured that information from the AI microservice is displayed and updated on the UI
        iii. Programmed the application to produce a report after every flight to give a summary to the farmer
    b. Cloud and local storage were implemented at this point.
        i. Functionally done using flash drive storage for flight data
        ii. Stores frontend UI information locally so that when the application isn't connected to Wi-Fi it will display properly and sync when re-connected to wi-fi
        iii. Connects to the database to make sure it works with local and cloud storage as well.
    c. The AI was given its dataset to learn from, and the microservice should be functional. After AI has processed the dataset and starts producing predictions, the frontend is ready to receive that data and display it to the UI for plant growth tracking, insect detection, and harvest readiness.
        i. AI is able to predict: Pest presence, leaf damage, canopy cover, weed presence, and lodging indications.
        ii. Revised AI Functionality Scope: After evaluting public agricultural drone datasets (G2F UAV dataset) and initial model testing, we identified physical resolution limitations that prevented precise V-R stage leaf-collar counting at standard flight altitudes. As a result, the AI scope was pivoted to act as a robust Phase Estimator. The final AI microservice successfully classifies 6 distinct biological growth stages of corn (Bare Soil, Early Growth, Mid Growth, Canopy Closure, Tasseling, and Senescence) and utilizes a 7th class (6_Background) to actively filter out non-agricultural noise like dirt roads, tractors, and weeds. Pest and



weed detection were scoped out of the MVP due to the lack of high-quality, high-resolution annotated drone data available in public datasets.

   d. Database development finalized at this point, supporting all of the information that needs to be stored.
      i. Flight Data: flight coordinates, flight time, hazards, and pathing
      ii. Crop Data: crop images, GPS coordinates, model predictions
      iii. Telemetry data: battery usage, field coverage, time date information
      iv. Field Data: Field GPS location, image capture points, field bounds
4. Integrated all pieces of software together and ensured that they all work together
   a. At this point, we should have full flights and data exchange that syncs with the cloud while on Wi-Fi.
   b. The desktop application should be fully functional at this point, allowing the user to view any information about flights and crops.
   c. Documented all of our work

Since we're an agile team, each of the numbers relates to where the sprints should be spent

Initial estimation:

1. Sprints 1-2
2. Sprints 2-5
3. Sprints 5-13, biggest chunk of our project is here
4. Sprints 13-16

Actual result:

1. Sprints 1-2
2. Sprints 2-8
3. Sprints 8-15
4. Sprints 15-16

Originally we assumed that 2 would take much less time than it did, but API connectivity with frontend, backend, and the AI microservice took longer than we anticipated as well as getting the Desktop application developed enough to utilize the connectivity. This resulted in a backloaded schedule that could've costed us but we were still able to get full software development implemented at the end of the day. Additionally, since we did good work in 2 & 3, 4 was a cake walk and full software integration didn't take much time at all.

AI Sprints:

- **Sprints 1-3:** Evaluated public agricultural datasets, identified resolution limits, and pivoted to Fractional Canopy Cover and 6-stage classification. Trained initial YOLOv8 prototype on a 24k-image dataset.
- **Sprints 4-6:** Engineered the *auto_labeler.py* script utilizing ExG and HSV algorithms to bypass manual labeling. Overcame Nova cluster storage limits.
- **Sprints 7-10:** Processed over 9GB of training data from 2017/2018 flights. Mitigated class imbalances and solved the "Label 5" logic error.

- **Sprints 11-14:** Finalized model training on the cluster, achieving 99.3% validation accuracy. Integrated the Python AI microservice with the C# backend API for full system data flow.

Hardware Task Decomposition

1. Ordered parts for the drone after doing extensive research on different possibilities to accomplish our goals
2. Assembled the drone physically, ensuring that it could handle being flown for hours a day. Durability was an important aspect.
3. Connected the drone with open-source flight APIs that allow the drone to be flown, keeping GPS coordinates and flight information on board until reconnected with the charging station.
4. Implemented the pilot software that controlled the drone.
    a. Implemented camera functionality for image capture.
    b. Created MSP functions that would be used to communicate between the Pi and the FC.
    c. Used the MSP functions to create a flight loop that will fly the drone autonomously.
5. Implemented XPlane 11 flight simulation.
6. Tested in XPlane, then in the field.

3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria

I. Overview

The goal of our project was to develop an Autonomous Drone with the capability to monitor and analyze crop fields for agricultural producers. This includes several components:

- Custom drone with the ability to fulfill all functional requirements specified in the initial project documentation
- Cross-platform desktop application
- REST API for data management
- Microservice for AI processing

II. Major milestones

Milestone	Description	Deliverable	Accomplished
1. Init and Setup	Establish Git repo, define architecture, set up development environments	Version control repo	Week 4
2. Hardware Integration	Assemble drone, integrate flight controller, GPS, camera, and sensors	Complete drone with flight, camera, and sensor functionality	Week 30
3. Backend & API	Implement REST API for data storage, flight logs, and telemetry data	Functional API with at least 60% functionality	Week 10

4. AI Integration	Develop an image-processing and crop-analysis pipeline; train an initial model using sample data	AI microservice with at least 70% accuracy	Week 30
5. Desktop App	Create cross-platform UI for control, flight monitoring, and data visualization	Desktop app for flight visualization and API integration	Week 12
6. Testing	Integrate drone, backend, and AI microservice with the desktop app	Full flight and data exchange	Week 31
7. Optimization	improve ML model performance, enhance UI/UX Improve growth-stage accuracy	Improvements on identified issues	Week 14
8. Documentation	Create documentation for each component	Component and instruction documentation	Week 30

Table 3.3.II: Major Milestones

III. Metrics & Progress

Drone

- Autonomous flight is enough to cover the field
- Accurate flight based on path determined (<= 3 m error)
- Sufficient data collected (Full GPS data, sensor readings, and imaging)

Database & API

- 100% of defined REST endpoints are functional
- Unit test coverage >=70%

AI Microservice

- Initial accuracy target 70%
- Dataset size >= 2500

Desktop Application

- All primary UI elements implemented
- Data visualization
- 0 critical UI/UX bugs

Team Metrics

- At least one deliverable per sprint



- >= 80% Git issues closed per sprint
- Code reviews completed within 72 hours

IV. Evaluation Criteria

- Drone achieves autonomous navigation across a predefined area with > 90% completion success.
- AI model meets target accuracy (≥ 70%) requirements.
- Backend/API passes testing and satisfies all data requests
- Data visualization and upload success rate ≥ 90%.
- System architecture maintains modularity and scalability for future integration
- Code quality and documentation meet basic review standards

V. Iteration and refinement

As the project progressed, goals and standards were updated to meet realistic expectations and help improve the team's development cycle. Future iterations refined AI accuracy requirements, drone field coverage success rate, API and database performance, and UI/UX design goals.

3.4. Project Timeline/Schedule

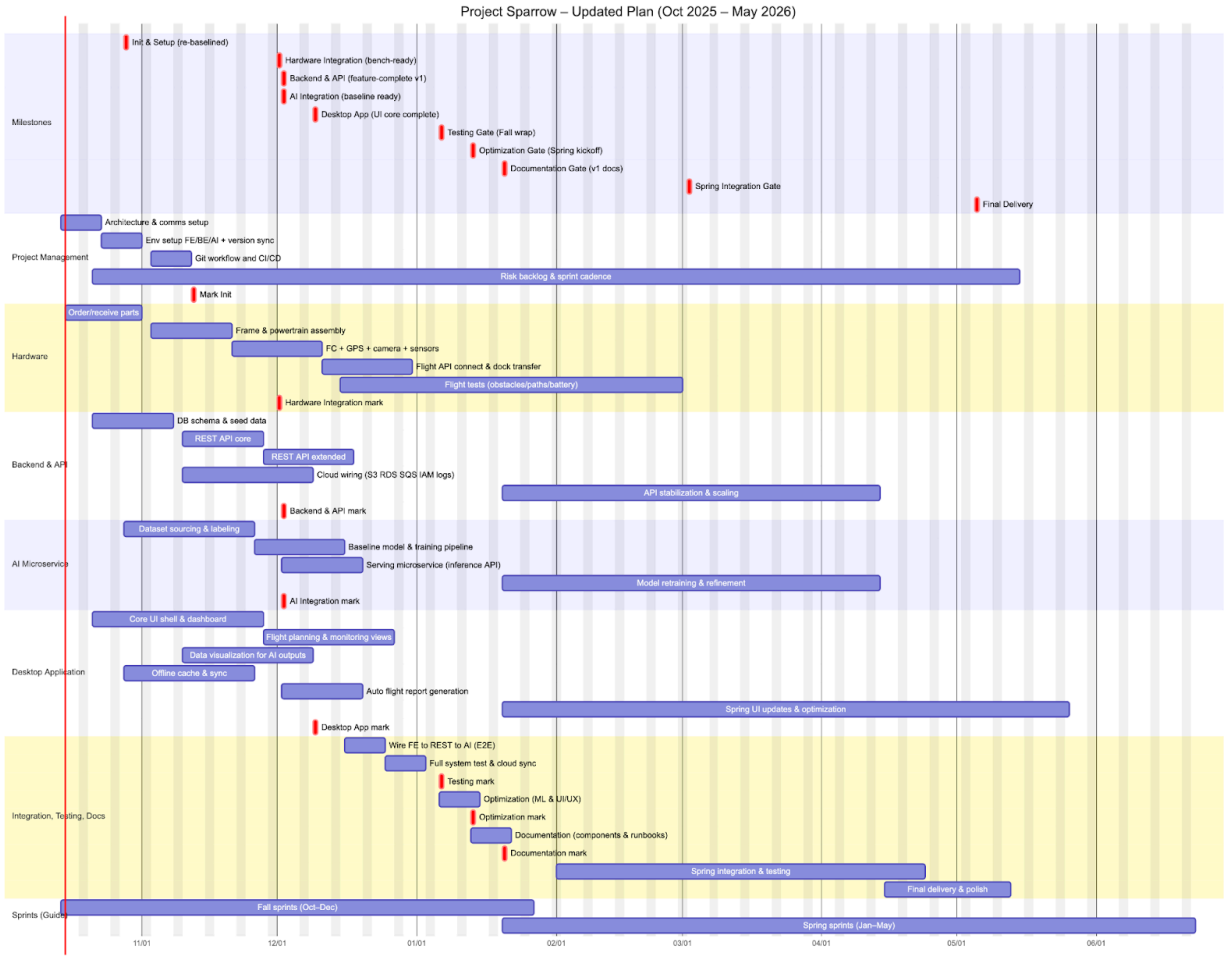


Figure 3.4 Project Timeline

3.5. Risks and Risk Management/Mitigation

Our project involved integrating multiple complex systems, including drone hardware, onboard AI vision models, cloud connectivity, and a farmer dashboard. As a result, several technical and operational risks were identified across our major tasks. Each risk was evaluated based on its probability and severity, with mitigation strategies proposed for those exceeding a probability of 0.5 or those with a high potential impact.

Agile Risk Tracking
Risks were re-evaluated during each sprint.

Task	Risk Description	Probability	Severity	Mitigation / Management Strategy



Drone Flight and Navigation System	GPS-based navigation may experience signal loss or instability.	0.6	High	Integrate fallback IMU-based navigation and manual override mode.
	Limited battery capacity may prevent reaching the 30-minute flight target.	0.7	High	Investigate higher-capacity Li-ion packs or lighter frame materials. Schedule test flights to optimize power draw by adjusting motor speed and sensor sampling frequency.
Vision AI Model (Plant Growth, Pest Detection, Harvest Estimation)	Model accuracy may be insufficient due to lack of locally collected crop data.	0.8	High	Request dataset access from agricultural research institutions in Iowa (e.g., ISU Ag Science Dept., Bayer, Corteva). In the short term, train a general-purpose vision model on public datasets (PlantVillage, CropDeep).
	Training infrastructure may be too slow for iterative retraining.	0.5	Medium	Use AWS EC2 GPU instances for training; enable model checkpointing for partial training progress recovery.
System Integration (.NET ↔ Python Microservice ↔ AWS)	Communication between the .NET backend and the Python microservice could fail due to API incompatibility or latency.	0.6	High	Define clear API contracts (JSON schema). Add logging and retry mechanisms. If persistent issues arise, containerize both services in a shared environment (e.g., Docker Compose).
	AWS service limits or misconfiguration could lead to data loss or downtime.	0.4	High	Implement S3 versioning and CloudWatch monitoring. Use IAM least-privilege roles and periodic backup snapshots.
Farmer Dashboard and Data Visualization	UI performance may degrade with large datasets or real-time feeds.	0.5	Medium	Use incremental rendering techniques and pagination. Offload analytics to AWS Lambda or preprocessed summaries.
Hardware / Sensor Integration	Camera or environmental sensor modules may not be compatible with chosen drone control board.	0.6	High	Verify interface compatibility (UART/I2C) early. If incompatibility persists, consider purchasing off-the-shelf compatible sensor suite.

Table 3.5 Task Risks Management/Mitigation

Sprint 1–2: Emphasis on hardware compatibility and base flight control.



Sprint 3–4: Vision model dataset acquisition and preliminary model training.
Sprint 5–6: Integration testing and cloud synchronization.
Each sprint will maintain a “risk backlog,” where mitigations (such as fallback algorithms, component replacements, or cloud service adjustments) are logged and prioritized for review during sprint retrospectives.

One unforeseen risk that came to pass was the discovery of a XSS vulnerability in the react-router version we had been using for this project. Fortunately we heard about it almost as soon as it was discovered, and were able to update the version of React that we were using to patch this vulnerability.

Several unforeseen risks emerged regarding dataset integrity and storage. First, our pipeline encountered corrupted zip files within the 2019 G2F dataset, limiting our historical data access. Second, we hit a strict 10GB disk quota bottleneck on the university Nova cluster, which required rewriting our Slurm scripts to handle data in smaller, sequential batches. Finally, we encountered a significant "Domain Shift" issue where our auto-labeler misclassified bright tire tracks and dead weeds as 5_Senescence (dead crops). We mitigated this by writing a biological "dirt filter" using OpenCV to eliminate non-plant structures, and introduced a 6_Background class into the YOLO model to teach the AI to safely ignore roads and equipment.

Well there were several, severe risks with testing the quadcopter let alone letting it fly autonomously. There were a couple that were foreseen thankfully with our meeting in the previous semester with Dr. Jones. One action he suggested we take to avoid harm to us and the drone was to have a tether to the drone when it flies so as to not have it fly far away. Another action was that we had a kill switch. Both ideas were phenomenal, and while the tether should have been a great idea as well, we did not implement it well in practice.

So while we had previously identified the risks that would happen, they still proceeded to happen. We first tested flight within a tightly constricted space that did not allow for much movement, well what would that lead to when the flight seemed to veer off just slightly let alone how much it actually did. It resulted in hard crashes that broke two Pis. And while we built a little brace for the Pi, we still could have just not equipped them since we weren't using it at the moment, or protect it more at the tradeoff of weight on the front of the drone. Then we proceeded to test in a larger space… with water right by.

While ideally, if we had a tether with a correct amount of slack, we would not have had it go to the water, it did almost immediately. It was almost as if the gods were telling us the worst that could happen will happen. Thankfully the drone was ok besides the Raspberry Pi. We then made sure the tether would not let it go all the way to the water.

I think these risks come from the combination of a lot of factors mainly the pushing it off to the end and having the rush feeling to get it done as quickly as possible, as well as just like pure ignorance of not thinking clearly of the consequences that could happen with a system like this, even though we were warned and were told of the severe consequences that occurred to somebody. So even though it happens to someone else and we were told, we still looked past. Although I am not saying it went horribly, we are all ok, nobody got hurt, besides maybe the drone a few hundred times. Either way I think we learned a lot to take this seriously, and from now on, we learned a valuable lesson in testing a very fast flying object.



3.6. Personnel Effort Requirements

Task	Time (hrs)
Hardware Integration Assemble drone, integrate flight controller, GPS, camera, and sensors	10
Backend & API Implement REST API for data storage, flight logs, and telemetry data	60
AI Integration Develop image-processing and crop-analysis pipeline; train initial model using sample data	60
Desktop App Create cross-platform UI for control, flight monitoring, and data visualization	50
Testing Integrate drone, backend, and AI microservice with the desktop app	100
Optimization improve ML model performance, enhance UI/UX	40
Documentation Create documentation for each component	40

Table 3.6.i: Projected estimate of personnel effort requirements

Task	Time (hrs)
Hardware Integration Assemble drone, integrate flight controller, GPS, camera, and sensors	30
Drone Software Write pilot software for the companion computer on the drone.	70
Backend & API Implement REST API for data storage, flight logs, and telemetry data	75
AI Integration Develop image-processing and crop-analysis pipeline; train initial model using sample data	80
Desktop App Create cross-platform UI for control, flight monitoring, and data visualization	90
Testing Integrate drone, backend, and AI microservice with the desktop app. Test drone in the simulator and field.	80

Optimization improve ML model performance, enhance UI/UX	20
Documentation Create documentation for each component	35

Table 3.6.ii: Actual personnel effort requirements

The desktop application took longer than anticipated to create and make look good, especially with user authentication. This means that more of our frontend developers' time was spent on this and less on enhancing the UI/UX of the app, as functionality is most important.

For hardware integration, the initial setup to get a flying drone did not take that long, probably around the 10 hours that we had initially predicted. The test flights would result in lots of adjustments made to the drone physically like where we would place modules or replace them entirely or calibrate sensors and PID took a long time. Which I am not quite sure if that is testing but it is both.

Either way we did not test the entire system working together starting from autonomous flight to an AI prediction. We have however tested the autonomous flight as well as transferring an image off the drone to the app that AI analyzed the image.

Setting up the backend functionality proved more time consuming than initially planned, with design and implementation of the core API and service layers taking roughly 15 hours, alongside the local and server database configuration taking another 5 hours. A major time constraint was caused by implementing user authorization, and implementing security, which took about 20 hours by itself when accounting for testing integration with the frontend portion. Setting up pipelines for testing and deployment, as well as support for cloud deployment cost approximately 20 hours, with the majority of work devoted to deployment configurations. Lastly, a major challenge was unit and integration testing, firstly for the backend in an isolated environment, and secondly for the fronted, backend, and AI service integrated in one singular system, 15.

3.7. Other Resource Requirements

Hardware Requirements

The drone used in this project is custom-built using components readily available online, with an approximate total cost of $600. Our suggested drone build is as follows:

- Flywoo Explorer LR 4 Drone ($192)
- 3000mAh 4S Li-Ion Batteries ($90)
- Raspberry Pi Zero 2 W ($31)
- Raspberry Pi HQ Camera ($71)
- Remote ID module ($35)

Additionally, the following parts are needed to interface with and maintain the drone:

- 4S LiPo battery charger ($35)
- ELRS RC controller ($73)



- 18650 battery ($15)

During the build process, some additional parts were ordered to improve the drone:

- RGB LED ($9)
- FPV GPS with compass ($20)

Testing Requirements

Testing the drone software virtually requires the XPlane 11 flight simulator, which can be found on Steam for $10 during sales or $30 normally.

Cloud Requirements

Our project will be performing data analysis in a backend server hosted within a cloud provider. We have chosen AWS to provide this functionality. Our proposed cloud architecture includes:

- **AWS Lambda** or **EC2:** to host the C# backend that handles the data the drone produces
- **AWS SQS:** a message queue to handle the images uploaded by the drone asynchronously
- **AWS EC2:** to perform AI processing on the images in the SQS queue and publish the results to our database
- **AWS RDS** (Postgres)**:** to store the results of data processing, drone telemetry information, and authentication data
- **AWS S3:** to store the raw images uploaded to the backend from the drone

User Hardware Requirements

To operate the drone in the field, one of the following desktop platforms is required (to operate the client application, which will create a flight plan and upload it to the drone prior to takeoff):

- macOS >= 14
- Windows >= 10
- Debian >= 9 / Ubuntu >= 16.04 / Fedora >= 30

4. Design

4.1. Design Context

4.1.1 Broader Context

Area	Description	Examples



Public health, safety, and welfare	Farmers will have the ability to purchase our drone platform and fly it over their fields for a report on the status of their fields. Within communities, equipment is often shared between farms, so one drone could be used for multiple farms.	Decreased cost of field monitoring tools, earlier identification of disease or pests, finding areas of improvement for future years (dead spots, pests), increased overall yield over time. Make sure the drone system is safe enough to not hurt anyone.
Global, cultural, and social	Farmers are often opportunistic, and so providing a cost effective tool to be an early warning or crop monitoring system is a value that our cultural group holds. Farmers are not afraid to adopt cutting edge technology, and this is just one more example of a new technology they are adapting to improve their operation.	Some communities view drones negatively citing privacy concerns, e-waste from another device, laws could be a limiting factor in some areas.
Environmental	Having a better understanding of what is affecting crops and how to treat it is the core environmental effect we are having with this project. More informed decisions can be made regarding what chemicals to use or solutions to implement. This reduces unnecessary negative environmental actions from agriculture.	Decreasing usage of chemicals, more efficient usage of resources, remediation of issues earlier on, better awareness of problems, more precise field management.
Economic	Our project aims to provide a cost effective solution to field monitoring systems compared to competitors. This reduces the barrier to entry to more farmers for similar tools. Through the usage of our tool, more proactive measures can be taken to fix issues, and resources can be used in a more efficient manner, reducing costs.	Competitive pricing, identifies problems earlier, identifies crop yield problems, farmers don't have to waste money and time on implementing the wrong solution, and increases efficiency for overall operations

Table 4.1.1: Broader context in which the design problem is situated

4.1.2 Prior Work/Solutions

There exist a ton of similar options in the market. There are almost a dozen established businesses that use AI to analyze a multitude of things about crops. Similar to our project, these companies have developed an AI, trained it on a dataset, and now sell their product for farmers to buy and use to get the most information about their crops.

From what we can tell, using AI in this capacity is new, with all available studies and articles coming out in 2025. This is a growing industry as AI capability continues to advance. One of the core reasons why these businesses are popping up is because the AI works extremely well. "Achieved exceptional predictive performance, with R^2 scores reaching 0.92, mean squared errors as low as 0.02, and mean absolute errors of 0.015 [1]." This quote shows that the AI is highly accurate and has low error rates.



While most businesses don't use drones (we will touch on this later), one of the studies mentions how "drones can be used in locations for real-time monitoring by capturing aerial data to access crop health and detect diseases [2]." This excerpt sounds eerily similar to our project and goals. This paper was published in April this year and outlines some of the strengths and existing challenges to making this idea a reality. It was a great tool for us to use as we continue to develop our software.

We know AI is good at its job and that there is at least one paper that outlines our entire project, so let's look at the challenges of using AI. "High upfront costs make AI technology inaccessible for many small-scale farmers, limiting adoption. Technical expertise is another barrier, as many farmers struggle to integrate AI. Reluctance to share farm data hinders AI model improvements [3]." While we couldn't directly relieve some of these challenges, namely price inaccessibility, we could definitely produce UI that is friendly to farmers and allows people of all technological backgrounds to use and benefit from AI analysis. After researching the products that businesses have to offer, they are much more robust than ours. They do many different types of crops, tell you when to plant, perform canopy-moisture analysis, tell you when to harvest, and provide many more data points of analysis.

The way that our project differs from all of the businesses that we could find is our drone. The businesses all use satellite imagery to take pictures of fields and then have their AI analyze those pictures, they don't use drones. This is where our project has a unique attribute, which is a blessing and a curse. None of the AI datasets are drone pictures, they're all satellite images, so we couldn't train our AI on satellite images and then have our drone take the pictures. Given the similarity of our project with existing businesses, the differences are pretty clear. The pro of our project is that we're programming and using a drone to take the pictures, which could allow for better data, as it's much closer and can get a better view of the crops, but at the very least, it's a unique approach to a current innovation with AI. The biggest con of our project is its scope: we're only focusing on corn, and we simply don't have as many features as these fully fleshed-out businesses. We are not following any previous work.

Electronic Documents

Professional Internet Site

[1] R. N. V. J. Mohan, P. S. Rayanoothala, and R. P. Sree, "Next-gen agriculture: integrating AI and XAI for precision crop yield predictions," *Frontiers in Plant Science*, vol. 15, Jan. 2025. [Online]. Available: https://doi.org/10.3389/fpls.2024.1451607. [Accessed Oct. 28, 2025]

[2] N. Aijaz, H. Lan, T. Raza, M. Yaqub, R. Iqbal, and M. S. Pathan, "Artificial Intelligence in Agriculture: Advancing Crop Productivity and Sustainability," *Journal of Agriculture and Food Research*, vol. 20, p. 101762, April. 2025. [Online]. Available: https://doi.org/10.1016/j.jafr.2025.101762. [Accessed Oct. 28, 2025]

General Internet Site

[3] Dr. Chantal Spleiss, "AI in Agriculture: Smarter Crops, Better Yields | CSA," *Cloudsecurityalliance.org*, Feb. 2025. [Online]. Available: https://cloudsecurityalliance.org/blog/2025/02/10/ai-in-agriculture-smarter-crops-healthier-livestock-better-yields#. [Accessed Oct. 28, 2025]



4.1.3 Technical Complexity

Our project demonstrates high technical complexity through its integration of multiple hardware and software subsystems, each employing distinct scientific, mathematical, and engineering principles. The overall goal was to develop an AI-integrated autonomous drone system for crop health monitoring, which required interdisciplinary expertise across control systems, embedded hardware, artificial intelligence, and cloud-based software design.

Flight Control and Navigation

This subsystem applies embedded control systems to achieve stable and autonomous flight. It relies on several sensors for navigation and their calibration along with accurate PID values keep itself alive. Firmware configuration through INAV, combined with communication via MSP protocol enables somewhat precise navigation.

Imaging and AI Vision

The imaging module combines deep learning and structural feature extraction. The first iteration focuses exclusively on growth-stage detection using the Ultralytics YOLOv8 architecture. Because the model acts as an Image Classifier rather than an Object Detector, it relies on complex Convolutional Neural Network (CNN) layers and Global Average Pooling (GAP) to extract geometric features (e.g., leaf ribbed textures, parallel row shadows) rather than relying purely on color. This allows the model to remain highly accurate across diverse geographic domains and lighting conditions, such as recognizing brown, dried-out corn in late-August flights as Senescence rather than bare soil.

Backend Processing and Cloud Integration

The backend handles data management. Collected images and telemetry data are transmitted to the server or cloud using RESTful APIs built with .NET and Python microservices. The system performs aggregation, and geospatial mapping before storing results in server or cloud databases (e.g., AWS). This requires knowledge of database architecture, network communication protocols.

Frontend Dashboard and Visualization

Built using Electron + Vite + React, the frontend integrates software design, data visualization, and human-computer interaction principles. The dashboard displays flight information and field + flight data in an intuitive layout. Communication is synced with the backend when the drone’s data is plugged in and selected to sync flights, which allows users to view drone telemetry and processed imagery. Interface design follows IEEE 1936.1-2021 (Drone Applications Framework) for standardized subsystem communication and IEEE 14764-2022 for software life-cycle processes.



Power and Hardware Optimization

Achieving the targeted 30-minute flight time requires the application of electrical and mechanical engineering principles. The team uses energy modeling equations to balance thrust, weight, and power consumption, optimizing motor-propeller efficiency and ESC performance. These calculations determine the appropriate battery capacity and hardware layout to ensure stable flight under variable conditions. Which is what we thought until it was kinda thrown a little out the window when we bought a smaller capacity battery and managed to fly for actually thirty minutes like we had anticipated before for the big battery. Maybe there was a slight science to balancing weight to battery size.

Problem Scope and Industry Challenge

The scope of this project matches current industry standards by integrating AI analysis, autonomous navigation, and cloud synchronization into a unified platform. Unlike most commercial AI agriculture systems, which rely on satellite imagery, our system utilizes drone-based data collection, offering higher-resolution imagery and more adaptable field coverage.

Key technical challenges include:

- Maintaining autonomous stability and accuracy under a new quadcopter that has not calibrated its sensors and PID along with outside factors such as environmental variability (wind, lighting, GPS noise).
- Designing robust cloud pipelines capable of synchronizing multiple data sources efficiently.
- Extending battery life and reducing payload weight while supporting high-performance sensors.
- Training our own AI model on unique, locally sourced images taken from a drone of corn in order to provide analysis on crops.

Together, these elements demonstrate that our project is not only technically complex but also ambitious in scope, integrating multiple scientific and engineering disciplines to deliver a system that approaches the sophistication of current commercial agricultural drone solutions.

4.2. Design Exploration

4.2.1 Design Decisions

A major design decision we had to make for this project was selecting the appropriate drone to use. We had a number of options available ranging from pre-built hobbyist drones, to partially built drone kits, to assembling a drone from a completely custom parts list that suits our needs. There were numerous tradeoffs with each approach. A pre-built drone would have given us the highest probability of having a working drone, but would have restricted us severely in terms of the hardware customizability (to add an on-board computer, cameras, and other sensors), and in terms of the software SDKs available, in that we



would be completely limited to whatever APIs the drone manufacturers exposed to us. On the other hand, a custom drone provides us with complete control over the software and hardware, but has a lower chance of success in that we may encounter hardware incompatibilities during assembly, and would have had to do more work software-wise to get the drone flying. This decision was further complicated by the unavailability of most drones and drone parts due to tariffs. Ultimately, we landed somewhere in between the two, creating a semi-custom drone build using specialized drone parts following the specifications from one of their recommended drone builds. This way, we know the hardware will be compatible, but we also have the ability to implement any additional hardware or software we need.

Another key decision we had to make was what framework to use for our frontend design. Initially, we had to decide between .NET MAUI and Electron due to the need for cross-platform compatibility (we wanted our app to be accessible regardless of computer OS), and decided on .NET via a team vote, because it offered more support for cross-platform compatibility with added support for Android/IOS if we decided to compile for a phone application. We later revisited this decision when we discovered that .NET MAUI does not compile for Linux and would thus be unable to communicate or compile with our CI/CD testing pipeline in the Git repository. We decided instead to compile with Electron-Vite.

The final design decision focused on planning the backend architecture to integrate with the AI microservice. To ensure stability and maintainability, the team chose to implement the backend using ASP.NET Core, with Entity Framework Core (EF Core) managing the data access layer.

The backend's API layer serves as the communication bridge between the user interface, database, AI microservice, and cloud infrastructure. This modular structure supports parallel task execution and enables the implementation of fallback systems to maintain reliability in case of component failures. All core operational logic is encapsulated within the API layer, which leverages both server resources and AWS cloud services to ensure system performance.

4.2.2 Ideation

We decided on the drone hardware to use for this project by finding any viable options for the project that could meet the minimum requirements (in terms of fly time, software control, and camera and GPS availability), then created complete parts lists for each drone in a spreadsheet. We then evaluated the pros and cons of each drone option in these categories:

- Cost. Is the drone under (or near) the $750 budget we'd like to target?
- Work to fly. How much will we have to do before the drone will be ready to fly?
- Software customizability. Would we be able to easily implement the autopilot software that we need in order to collect data?
- Hardware customizability. If the required sensors are not on the drone itself (or inaccessible), will we be able to add them ourselves?
- Risk. Is it likely that we will encounter some issue that prevents us from getting the drone in the air? Additionally, will we be able to implement all of the software that we need?
- Flight time. Will we be able to collect a reasonable amount of data before the drone needs recharged? For custom builds, we estimated this using eCalc.

Ultimately, we had 5 real contenders for the drone after a few revisions:



- DJI Mini 3. This drone is completely pre-built and we would be able to get one quickly. It has a very strong fly time and would be easy to get flying, but limits us on custom hardware and software with the SDKs made available. We originally chose a DJI Mini Pro 4 for this, but those are no longer available.
- Custom 10” Build. This class of drone can carry a lot of weight, meaning we would get near the best fly time we could get for a custom drone within our budget. We also get complete hardware and software control, but it is the most risky choice.
- Custom 7” Build. The smaller size means we could use smaller motors, props, and batteries, which improves the cost. However, we lose some flight time. Same risk as the 10” build.
- Partial Custom 4” Build. This drone is built mimicking a pre-built drone provided by the same supplier, nearly guaranteeing hardware compatibility and giving us complete hardware and software control. It gives us a good flight time, but less than all the other options. Part availability is questionable.
- Pre-Built 4”. This is the drone the custom 4” build mimics. We get less hardware customizability and less control over how the parts are assembled. The flight controller runs open-source software that provides a UART control interface for our on-board computer.

4.2.3 Decision-Making and Trade-Off

To decide on the drone, we ranked each option in the categories described above and picked the drone that best fit the categories overall. The three factors that are most important to the project, which were the risk factor and hardware/software compatibility, are weighted double as if the drone is too risky to build or unable to satisfy the project requirements it is not a good option.

Drone	DJI Mini 3	Custom 10”	Custom 7”	Partial Custom 4”	Pre-Built 4”
Cost	4 ($450)	1 ($750)	2 ($650)	3 ($600)	3 ($500)
Work to Fly	5 (Easy)	1 (Hard)	1 (Hard)	3 (Okay)	4 (Easy)
Software Customizability (x2)	2 (Bad)	5 (Great)	5 (Great)	5 (Great)	5 (Great)
Hardware Customizability (x2)	1 (Bad)	5 (Great)	5 (Great)	5 (Great)	3 (Okay)
Risk (x2)	3 (Okay)	1 (Risky)	1 (Risky)	3 (Okay)	5 (Little risk)
Fly Time	5 (51 mins)	4 (38 mins)	3 (30 mins)	3 (30 mins)	3 (28 mins)
Total	26	28	28	35	36

Table 4.2.3: Drone options and pros and cons for each ideated options

Ultimately, the two 4” drone builds were the strongest contenders: they give us great software control, since they run on open-source software, and both are generally not risky options relative to the fully custom drones or the DJI drone with limited software support. The only areas where they are not as good is in fly time, though most of the drone options we have were around the 30 minute mark, making them viable. The pre-built small drone (Flywoo Explorer LR, with parts listed in Section 3.7), was ultimately the least risky, safest option, while still including hardware and software that allowed the customization needed to operate the drone autonomously and for long distances.

4.3. Final Design

4.3.1 Overview

The drone system takes commands from a user, such as creating a flight path or flying a flight path, and captures data with its sensors to provide information to the user once the flight is completed.

The frontend design consists of a main “home” page, as well as several sub-pages. The home page will display overviews of the most important collected data, which can be found on each of the sub-pages. By clicking on this data, the app directs you to the respective sub-page for more in-depth information. The app also features user accounts, allowing each user to view their recorded data and identify any issues with their crops.

4.3.2 Detailed Design and Visual(s)

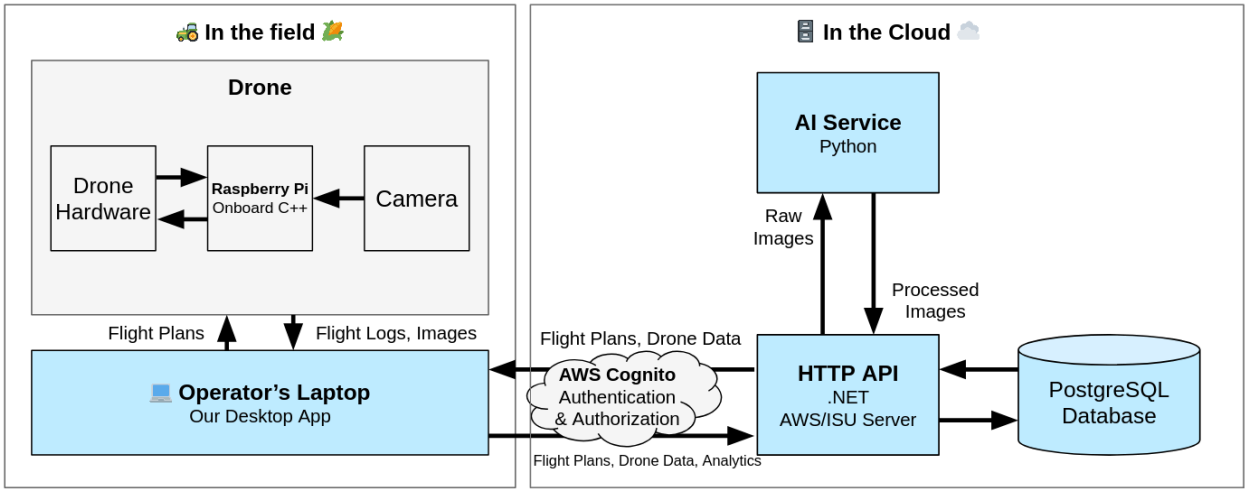


Figure 4.3.1: High-level system overview

The design begins with the drone system, which is configured with a magnetometer, barometer, GPS, camera, and on-board Raspberry Pi Zero. The Raspberry Pi uses software written in C++ to communicate with the open-source flight controller software, INAV, which flies the drone, to get telemetry data and set

waypoints for the drone to fly to, using a serial connection and the MSP protocol. While the drone is flying, telemetry data from INAV back to the Pi will note where the drone is, and at certain waypoints the Pi takes an image with its attached camera and stores it to an SD card. The drone will continue to capture data until either the battery runs low and it needs to return prematurely, or all waypoints are reached. Once the user wishes to process the data, the user can pull the data (images and telemetry) from the SD card on the drone, which will be read by the frontend.

For the frontend, we are using an Electron-Vite configuration using React. This allows us to work in TypeScript as well as JavaScript and use CSS files for styling. Electron and React provide a preview of

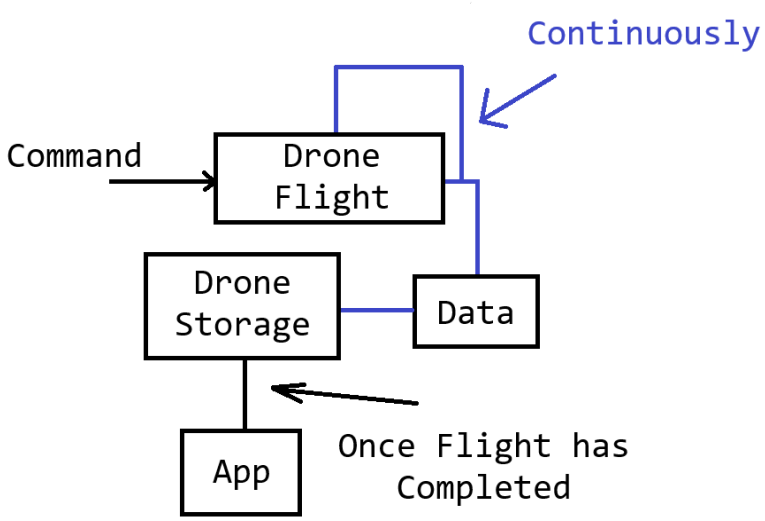


Figure 4.3.2: Autonomous drone system diagram

the page that updates in real-time and can be resized. On the home page, we feature dynamically updated data images from the most important sections of each separate page on the website. Each image directs you to its respective page when interacted with. Additionally, we include a navigation bar at the top of the screen. The website features user accounts so that each user can only view their respective data, and we will utilize an empty page for logging in when the app is launched, which redirects to the home page when the process is complete to make sure no user data can be leaked. User passwords are censored as they are entered, and the website's entry fields adhere to cybersecurity guidelines for websites.

4.3.3 Functionality

In a real-world context, our design is intended to minimize the amount of work required by the user while providing an intuitive experience across operations. Upon opening the application, the user logs into their



account and is presented with a dashboard displaying past flights, including telemetry data, flight paths, surveyed locations, and notifications about any detected field issues.

From the dashboard, the user can navigate to the flight planning screen to configure a new field survey. Here, the user defines the desired field they want to survey and sends a signal to the drone to initiate the survey.

After the flight is completed, the user uploads the collected data—such as telemetry logs and captured images—via a microSD card connection to the device running the flight planning and overview software.

Once uploaded, the frontend application sends the data to the backend API, which:

1. Stores the data on the server.
2. Interacts with an AI-based image processing microservice.
3. Returns the processed results to the frontend.
4. Synchronizes the data with the cloud database for backup.

When processing is complete, the user can view the results within the application. The UI presents the analyzed field data, highlighting crop health, growth status, and any detected anomalies (e.g., signs of disease). This process aims to ensure users can easily plan, execute, and review drone surveys with minimal manual effort.

4.3.4 Areas of Challenge

Our primary development priority was to acquire the required hardware components so we can begin integration testing and refine data collection and transmission processes. Due to additional safety considerations that needed to be made and approved, as well as limited parts availability, this had taken much longer than anticipated, but our design was eventually approved, and we acquired the necessary parts. During this process, we also strived to make sure we were not blocked while waiting for this, working to get as much done as possible in the meantime, so we could move more quickly once the drone was assembled.

Well there were a lot of areas of challenge in the second semester. First we'll start off with technical progress. When the functionality of capturing an image on the Pi was being implemented, the challenge that was encountered was the limited knowledge of C++ that slowed down the process. This then sequenced into the next part of the drone software which was to create a flight loop with MSP commands. The main challenge that came in this part was the lack of motivation to keep going. Thankfully we just got it.

Now in regard to satisfying requirements, how are we going to detect diseases on these crops when it's hard enough to get data for regular corn, how about data for corn with certain diseases? Or detecting if there are bugs on the crops, now maybe the camera can be really zoomed in as well, but unless these bugs grew a bit then how are we going to see the little monsters rummaging the corn? Regardless, the objective that we were requested to have corn and soybean health, we do not have anything regarding soybeans and nothing to health for either crop. Also we were supposed to have night time detection but the budget set would not allow us to capture thermal images on a drone as they are lots of money.



Then for meeting user needs, an important point that we did not cover was to ask for feedback for farmers as those were our users. However we did not get in contact with any which regards us to making heavy assumptions as to what the users might be wanting.

AI & Data Processing Challenges:
Developing the AI vision model presented significant challenges regarding data handling and hardware limitations. While processing the G2F UAV dataset, we encountered a strict 10GB disk quota on the university's Nova HPC cluster, which crashed our data extraction scripts. Additionally, the 2019 dataset suffered from severe zip file corruption, making the data inaccessible. To overcome the HPC storage limits, we rewrote the automated Slurm pipeline to download, process, tile, and clear the data in small, sequential batches rather than attempting to hold the entire dataset in memory.

4.4. Technology Considerations

Depending on the drone system that we were going to receive we would be implementing a few different technologies. A couple of them are distinct by the size of the frame—spanning from 4-7” which allows for greater carry while weighing more, but will allow for higher battery or being able to sustain wind conditions more. The higher the frame the more expensive the parts become. They are all deployed on relatively similar drone parts from their frames since the power needed varies. However they all contain motors, props, flight stacks, a computer, and additional modules to fulfill the technical requirements of the design.

Given that we can either receive a custom/partially custom drone kit or a pre-built, the process in how they will fly varies. The ability to be able to program the drone to our specifications is more feasible under the custom drones as they do not restrict us to proprietary SDK’s while allowing us to use open-source software that will expedite the process for our application. However building a custom drone will involve slightly more risk as although we have extensively researched parts and drone systems, pre-built drones have higher reliability in working immediately and throughout the project.

Being able to process the data or images that we receive from the drone will also be unique to custom drones or pre-builts. For pre-builts, there seems to not be a distinct way to be able to freely process the images that the camera captures. Meanwhile, the custom drones will be able to utilize a computer and an SD card to extract images and process on our app.

5. Testing

5.1. Unit Testing

- For the frontend, we are using the React Testing Library for Unit testing. We are testing individual windows, interfacing between windows (such as changing views), and also testing for both regular and erroneous inputs. It sounded simple, and usually the tests were, but as the frontend continued to grow in scale, the number of different paths a user could take, or buttons they could click, rose significantly, requiring more unit tests.



- In order to test frontend API functionality, we created a separate API testing page, with buttons to run each API call, and input fields to change which field, flight, or ID we wanted these calls to use. In this way, we were able to manually test each individual API call, as well as see the what order we should put these API calls in for end-user usage

- The drone software, written in C++, utilizes the Google Test library for automated unit testing. The major sections of the software being tested are the serial communication code (between the flight controller and the on-board computer), the camera controller code, the flight plan generation code, and the telemetry parsing and collection. Within each of these sections, individual functions are tested for common expected cases and possible edge cases, as well as for larger classes as a whole, to ensure that class state and methods interact correctly.

- The backend uses xUnit as its primary testing framework within the .NET environment. All tests are organized under the AutomationDrone.Tests project, which includes:
    - **Base/** – shared setup utilities for test classes
    - **Services/** – unit tests for core service-layer logic
    - **TestUtils/** – test data factories and helper objects

  Service-layer tests focus on validating logic, data transformations, and interactions with the data layer. SQLite (in-memory) is used during testing to simulate database behavior and constraints, ensuring that service operations behave correctly without requiring a live database.

5.2. Interface Testing

The frontend UI is rendered using React for seamless testing and development. This allows frontend designers to make changes in the code and see their effect immediately, to determine if the output is what they envisioned or not.

The backend API is tested using Postman to simulate network POST, GET, PUT, and DELETE calls to the specified endpoints. This allows both the frontend and backend to determine which input fields are required for each database call, as well as verifying that their API calls will function correctly before they are implemented into the codebase.

In order to test communication between the frontend UI and the backend API, we created a separate hidden page for testing purposes. This page contains buttons for every API call, as well as input fields for any input information (like field number, flight ID, etc.) that may be needed to make those API calls, in addition to having test intersection and flight objects with data hardcoded for testing purposes.

The PostgreSQL database where we store the flight data is also viewable via the VM that we were provided by the university, and this allows us to view the data that we POST and PUT to make sure we are both sending the right data and also that our GET requests are receiving the correct data from the right endpoint.

5.3. Integration Testing

- Integration between the frontend and backend occurs through API requests, where the frontend sends flight and location data to be stored in the database. Testing for this involved simulating API requests using Postman and then replicating the same format as those successful requests to integrate into the frontend functionality. This was then tested at a later time individually to ensure functionality still behaved within the expected bounds, and if errors occurred, that they are handled properly.
- Within the drone software, the interaction between all major components is tested end-to-end using our Google Test test suite with mocked interfaces. The software is used to communicate over serial with the drone flight controller and with camera hardware attached to the drone's on-board computer, so using these mocked interfaces allows us to test (to the extent possible) how the software as a whole works together with general inputs that we define, without needing to risk damaging the actual hardware with real-world tests.
- The full drone simulation loop (including the interaction between the drone software, INAV, and a drone) is tested fully within the XPlane 11 flight simulator and INAV's Software in the Loop simulation capability. This allows us to ensure that the on-board pilot Pi is communicating properly over the MSP protocol, that the INAV drone accepts the messages we give it, that INAV is able to pilot the drone as our software tells it to, and that the drone software is able to read data and take pictures at the proper locations.

The backend integration tests target the API layer, validating end-to-end request handling across controllers, services, and the database. Tests are implemented using xUnit for consistency with the existing .NET test environment.

These tests verify that the backend provides correct and stable responses to the frontend and the AI microservice. Critical integration paths include:

- **Flight Operations:** create, update, retrieve (single and list), and delete flights
- **Flight Data Operations:** create, update, retrieve (single and list), and delete flight data
- **Location Management:** create and delete locations

Each integration test issues HTTP requests against the API, ensuring the full request pipeline operates correctly, including routing, validation logic, service-layer interactions, and database access.

5.4. System Testing

For the AI vision subsystem, system testing focused on validating the end-to-end pipeline from drone image input to the growth/weed/pest outputs consumed by the backend and dashboard. The goal was to ensure that, given a UAV image, the system produces correct detections and growth estimates, in the expected format, within the required latency.

At the unit level, we tested individual Python components, including image preprocessing (resizing, normalization, and tiling), model loading, inference wrappers, and post-processing logic (thresholding, class mapping, and growth percentage calculation). These tests were implemented using pytest and utilize small, fixed test images and mock model outputs to verify correctness and handle edge cases.



Interface tests verify that the AI microservice API matches the contracts defined with the backend, including the correct HTTP method, request schema (image or image URL, crop type), and response schema (classes, bounding boxes/masks, confidence scores, and growth percentage). These tests use automated API calls against a staging instance to ensure backward compatibility as the model evolves.

Integration/system tests run the full pipeline using a held-out set of UAV images and ground-truth labels. For each test batch, we compute accuracy/IoU and compare against target thresholds defined in the requirements. We also measure end-to-end latency (from image received to JSON response) to verify real-time constraints. Model-level evaluation uses the built-in YOLO/Ultralytics validation tools, while higher-level system tests are scripted in Python and integrated into our CI pipeline.

5.5. Regression Testing

All changes made to the project were posted to the Git repository as merge requests, which were then reviewed by other team members before being merged. The repository has CI pipelines set up, which detect the specific section of the project where changes were made (frontend, backend, drone software, or Terraform) and run all available automated tests on it, reporting the results back to the GitLab interface and blocking the request from being merged. Running the entire test suite for any change ensures that earlier features, which were previously unit/integration tested, will run again with the changes, allowing us to detect unintentional changes to earlier behavior.

Critical system features, as dictated by our project requirements, will be covered by these automated unit and integration tests wherever possible, thereby reducing the risk of testing new changes against the actual drone due to undetected regressions.

- Drone serial communication is extensively tested with unit tests and a mock serial port implementation in software to ensure the drone flight controller receives the commands it expects. Additionally, it will test entire mocked flights with simulated data transmission between the mock flight controller and our drone software to ensure that changes do not cause issues with the autonomous flight capabilities.
- With each drone software change, a test flight is performed in XPlane to ensure that the change does not affect any critical system of the drone.
- The frontend's critical features, such as field waypoints and data transmission to the drone, also include automated testing in a similar form, with unit tests on a low level and integration tests against the entire flow as much as possible.
- The backend is set up to automatically test that every API endpoint functions according to the project requirements and that data persistence and storage are correct.
- The cloud infrastructure configuration utilizes Terraform, and merge requests that modify infrastructure will generate a plan (detailing the changes that will be made if applied) in the CI logs. This plan is reviewed before being applied, which helps uncover unintentional changes to previously deployed infrastructure.

5.6. Acceptance Testing

To demonstrate that our design requirements are being met, our drone will accurately fly along a pre-designated flight path and take pictures of the crops in that field. These pictures will then be uploaded to our app's frontend, which will take the flight data and pass it to our backend database. From

there, our database will send the crop images to the AI model, where the trained model will analyze these pictures to determine and return the correct growth stage and growth percentage. To involve our clients in the acceptance testing, we wanted to reach out to farmers and ISU farms to gauge their willingness to test our product and provide feedback, whether based on their UI/UX experience or the accuracy of the AI model responses. Unfortunately, given the timing of the senior design class (starting in fall/winter and ending in spring) we were simply in the incorrect window of time to try and get feedback on our project's functional capabilities in this regard. We would've either had to have a working drone and software at the start of the project (potentially during fall 2025 harvesting season), or the semester would've needed to last longer in order for us to gather samples over the 2026 growing season.

5.7. Security Testing

When considering security testing for our project, we had to ensure that we cover every aspect of our project. The components we identified were the frontend dashboard, backend (API), drone software, AI microservice, and cloud infrastructure. Because the system can give the drone commands to operate autonomously, it is crucial to prevent unauthorized access to ensure the integrity of the flight instructions and the data/information itself. We used STRIDE as a guide for a threat model for this project to identify the risks. From there, we ensured that each threat is assigned a specific test case, ensuring our testing covered all relevant security requirements.

Backend security testing was performed using Postman & xUnit, where we validate authentication, authorization, and input validation by sending malformed payloads, invalid tokens, and large numbers of requests to ensure the backend handles potentially malicious behavior correctly. The drone software was tested through services like Google Test and mock serial interfaces, ensuring the onboard computer only accepts commands that are currently authenticated and formatted. Full simulated tests were also run using the XPlane flight simulator. For the frontend, React Testing Library and manual testing were used to test secure token handling, form validation, and unauthorized page access. The AI microservice has similar testing for payload validation and endpoint isolation checks.

For infrastructure, Terraform plans and CI-based security checks verify that our cloud resources are configured correctly, adhering to the principle of least privilege. Static analysis tools could have been used for this (such as SonarCloud, ESLint, or cppcheck), but were not throughout this project as development is still ongoing.

Altogether, these efforts demonstrate that security is a priority in our project and was implemented throughout the entire development process.

5.8. User Testing

We wanted to ensure that our product flow, UI, drone, and everything else are usable, safe, and accessible to users without requiring advanced technical skills to understand how to use them. We would have liked to focus on asking farmers how they feel about the product, the UI, and the drone. Unfortunately, due to previously mentioned timing restrictions (starting the project in the late fall and being unable to acquire parts until late winter, and the project ending before crops have really grown), we were unable to reach out to gather any useful feedback on drone operations and functionality as a result.



5.9. Testing Results

Backend Testing

Thus far, we have successfully implemented a backend API and integrated it with the frontend user interface. We can successfully send and retrieve flight and location data, provided the proper flight ID is provided, allowing users to access only their own flight data. We also expanded this to include authentication and user accounts, allowing us to track flight data per user rather than users per flight data.

Drone Testing

Given the importance of reliable operation, the drone was extensively tested with three increasing layers of tests to give us the highest possible chance of success in field tests.

First, the drone pilot software includes mock interfaces covering any dependency only operable when running with the drone and the Pi. The software implements a virtual controller for arming and changing drone modes, a UART-over-TCP implementation for testing communication with the flight controller, a mock camera interface for testing where images are taken, and a mock LED to allow us to see how the software would operate if run on the drone. These mocks allow for extensive unit testing without requiring the drone to be attached.

Second, the mock interfaces and use of INAV allow for full simulated flights with the real INAV flight controller code and a drone in a flight simulator against the full drone pilot software. This gave us a complete software stack for testing how the drone will respond to pilot software commands, and allowed us to detect numerous severe bugs without having to send the drone into the air, saving lots of time in debugging and preventing crashes in the field.

Finally, controlled field tests were performed in a grassy area with the drone tethered to the ground. This gave the drone a soft area to land, limiting any damage that occurred during crashes. It also ensured that the drone could not fly away or get to dangerous areas in case of incorrect commands or misconfigurations.

Ultimately, through these testing strategies, we were able to complete a full autonomous test flight with the drone without damaging any critical drone components.

AI Vision Subsystem Testing

Our initial prototype AI model demonstrated exceptional performance. Upon training the YOLOv8 model on a balanced 24,000-image dataset spanning 6 crop growth stages, the system achieved a **99.3% validation accuracy**. Evaluation of the model's Confusion Matrix revealed only 9 misclassifications out of approximately 1,400 validation images. Critically, the few errors the model did make occurred on the boundary between biologically similar stages (e.g., Mid Growth vs. Canopy Closure), proving that the Convolutional Neural Network successfully learned the mathematical progression of the plant structures rather than just memorizing environmental noise.

6. Implementation

We have successfully implemented our frontend UI, complete with user authentication, site navigation, and field functionality. These separate pages allow us to show a simplified breakdown of important information on the home page, and then link to the more in-depth pages when each breakdown is interacted with. We have implemented the Google Maps API for our users to mark out the size of their field that they would like the drone to patrol. This allows us to then plan the drone's flight path to maximize coverage of the subdivisions of the field, while also minimizing flight time (to conserve battery power on larger fields). This is also where we get the intersection points where the drone takes the pictures. We have a page to view existing fields, and the ability to schedule flights for the drone there. Additionally, that page shows all the flight data for the selected field for the user to see. There is also lightmode & darkmode capabilities as shown in the screenshots. All field & flight data is tied to each individual user.

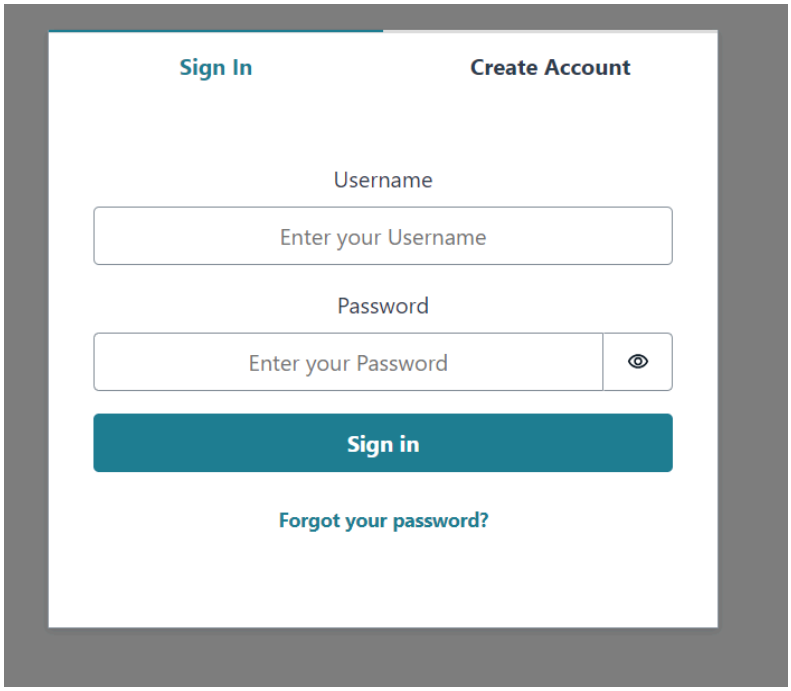




Figure 6.1: User Sign in

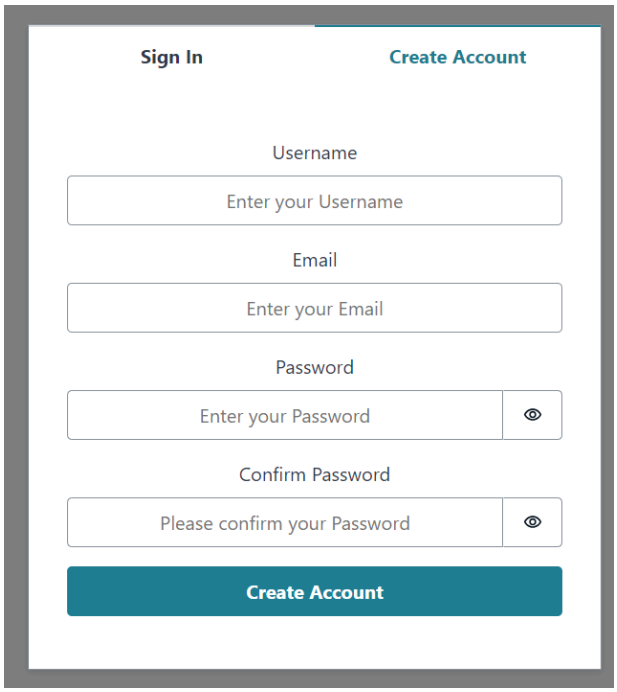


Figure 6.2: User Create Account

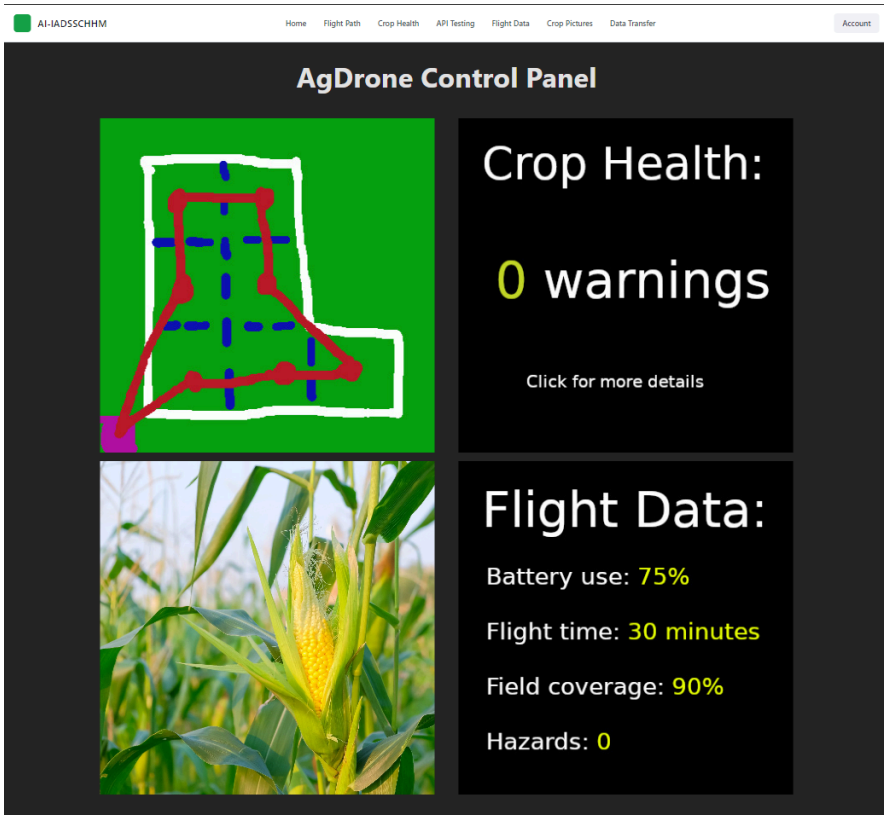


Figure 6.3: Homepage

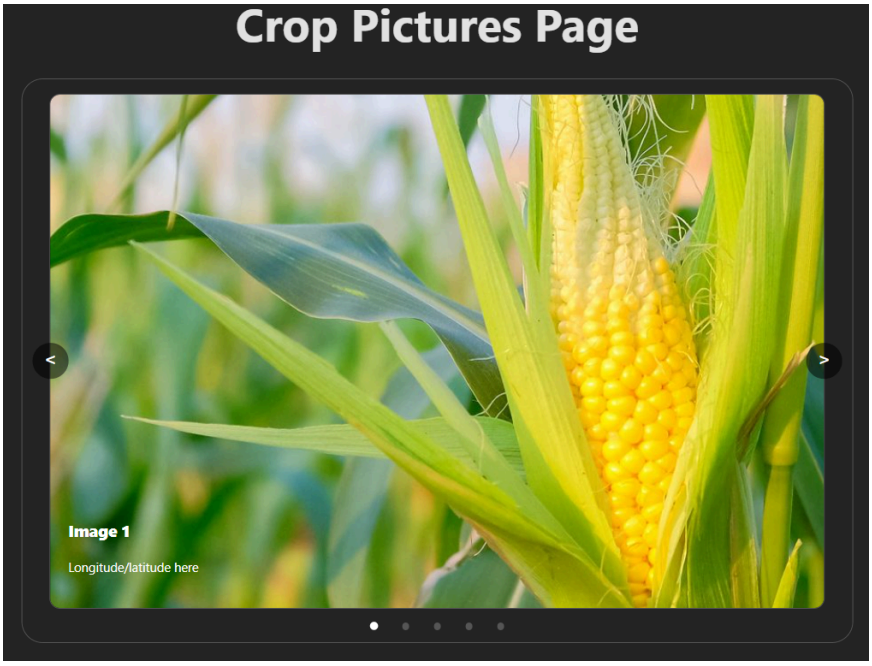
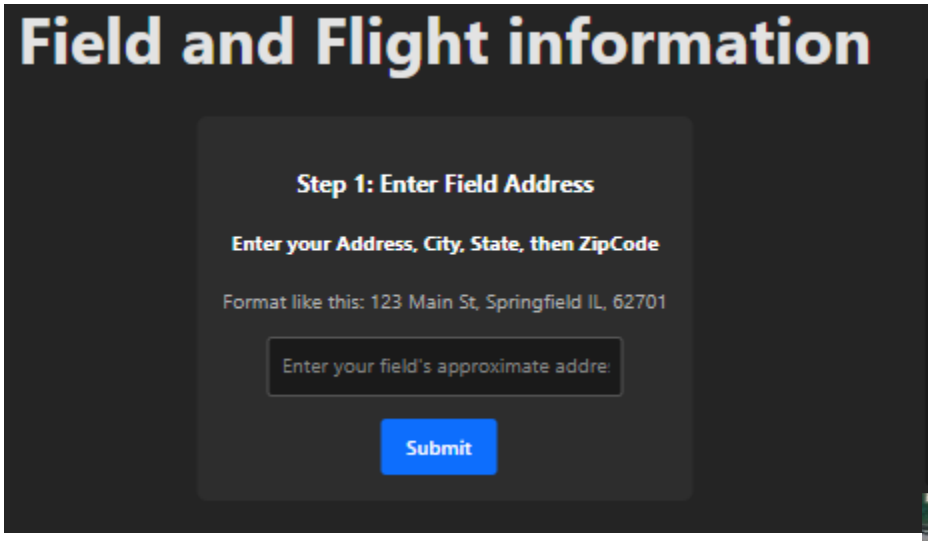


Figure 6.4: Crop carousel page, user can see crop pictures taken by drone here



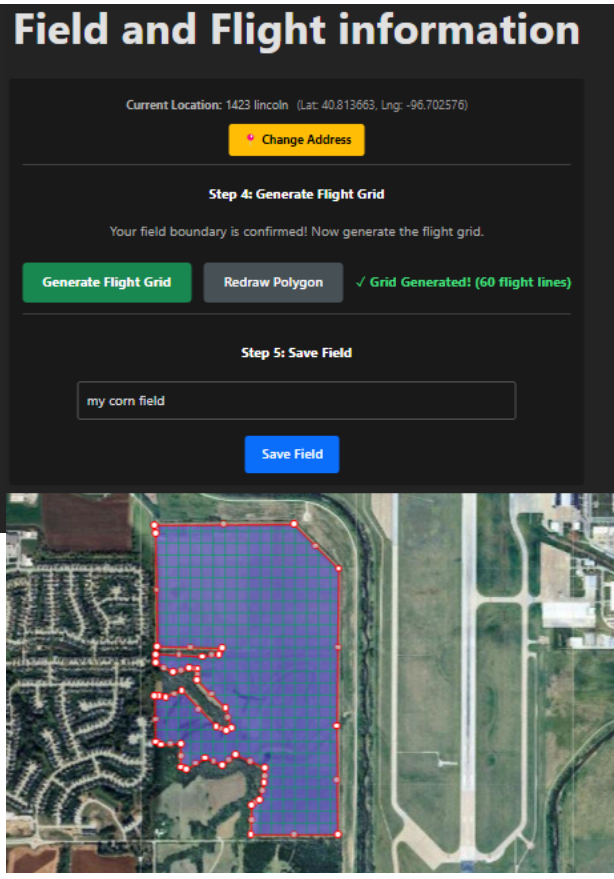


Figure 6.5: (Top) and Figure 6.6: (Right) Flight Creation and outlining thanks to Google Maps API

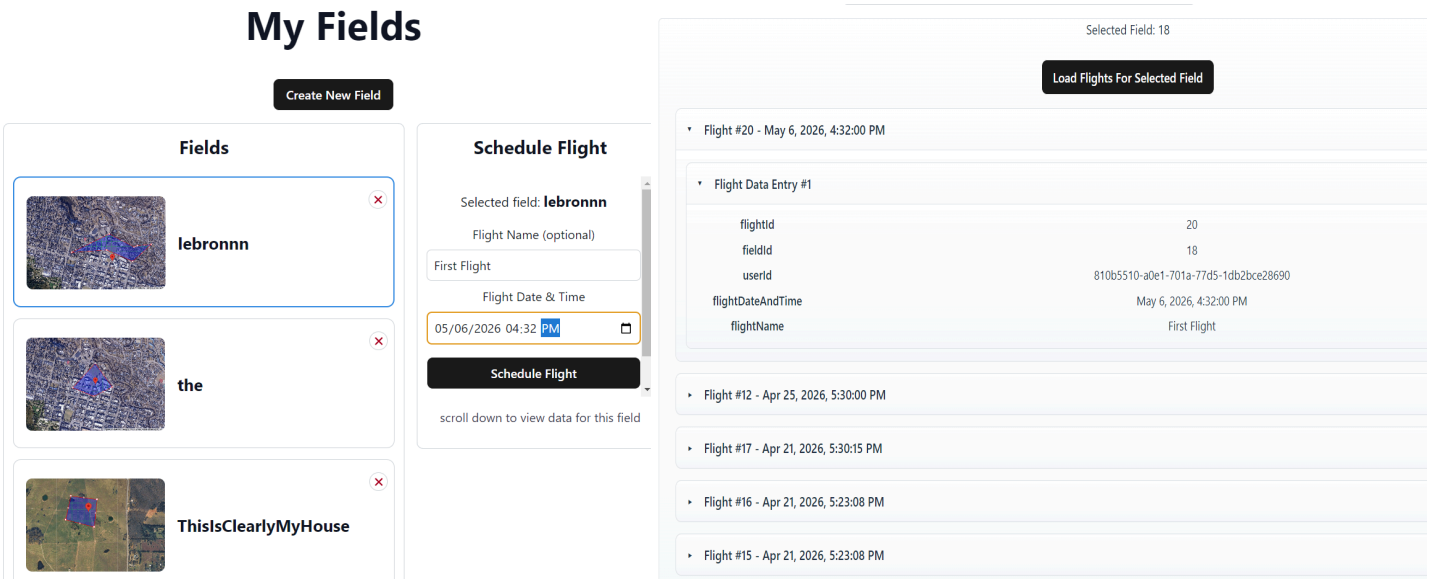


Figure 6.7: Saved field viewing & flight scheduling screen, with flight data information at the bottom of the screen.

Field viewing and selection for already created fields, the user can see the name and snapshot of the field, select it and then schedule a flight. Then the user can scroll down and load all of the flights for their selected field, in the screenshots you see how a flight scheduled for field “lebronnn” at 4:32 PM on may 6th, 2026 is scheduled by the user and then instantly available to be viewed by that user as well as all of the pre-existing flights.

The backend authorization and user data management are complete, allowing users to retain their data, and utilize AWS Cognito for security management. Core API functionality has been implemented, allowing crucial information to be stored related to flight planning and navigating, as well as data collection to allow enhanced accuracy for future flight planning and hardware utilization. Lastly, the backend has a multi-environment configuration, that allows for deployment to both dedicated servers, and cloud management through AWS.

The drone hardware implementation is also complete, in addition to a near complete implementation of the pilot software. The factory Flywoo Explorer LR was assembled and tested. Then, the Pi mounts were printed, and the drone was disassembled to install it. Testing then continued independently until the Pi was ready to test with the drone’s flight controller. Then, wiring harnesses were made with JST connectors to the extra ports available on the drone’s flight controller, including a ground/RX/TX harness to the Pi’s GPIO for a UART connection to the drone, and a 5V MicroUSB connector to power the Pi off of the drone’s battery. An RGB LED was soldered together with 220Ω resistors on each power leg to connectors on the Pi’s GPIO. Finally, a new GPS/compass combination was installed on the drone to

improve navigation accuracy, which required soldering leads to the drone flight controller's SDA and SDL leads, together with a 4-pin JST connector to the old GPS connection, all into a 6-pin JST connector on the new GPS. Finally, extensive calibration was performed in the INAV configurator together with many test flights until a successful navigation mission was achieved.

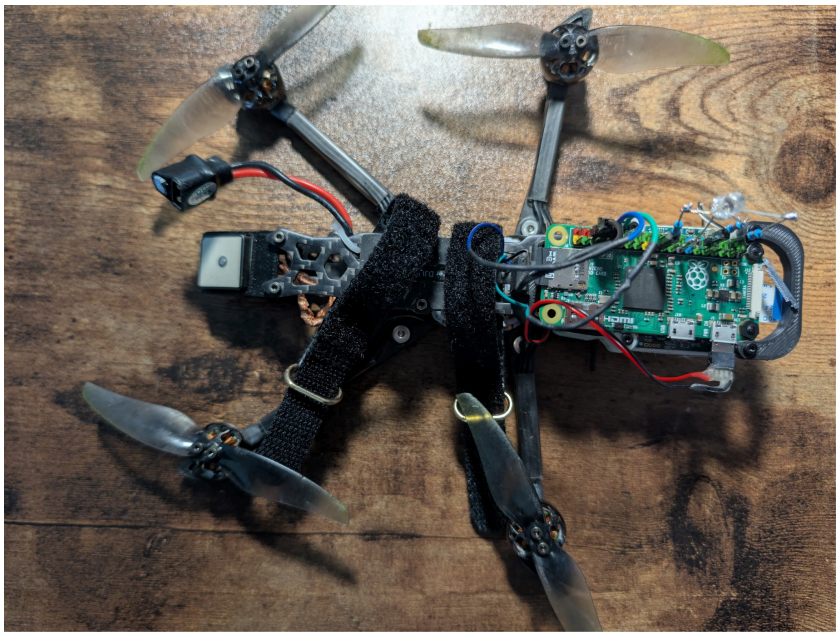


Figure 6.8 Complete drone build

The drone software was also almost completely implemented. The MSP protocol, camera and image functionality, data reading/writing, mock interfaces, simulator support, and a partial flight loop were implemented and tested. The current iteration is able to read the flight plans created by the frontend and send them to the drone's flight controller over MSP, then logging the navigation status until the flight is complete.

```
camden@camden-laptop sdmay26-20/Drone (finish-drone-sw) » ./build/pilot 127.0.0.1 5762
    0.000 INFO  pilot               starting at 1777929796
    0.000 INFO  led                 set color to off
    0.000 INFO  led                 set color to cyan
    0.000 WARN  pilot               simulator mode enabled
    0.000 INFO  fakeuart            trying to connect to 127.0.0.1:5762...
    0.000 INFO  fakeuart            connected
    0.000 INFO  pilot               opening data manager on .
    0.000 INFO  pilot               reading drone config
    0.000 INFO  fakecontroller      ready
    0.000 INFO  fakecontroller          'a' to arm
    0.000 INFO  fakecontroller          'd' to disarm
    0.000 INFO  fakecontroller          'n' to enter normal flight mode
    0.000 INFO  fakecontroller          'g' to enter gcs nav mode
    0.000 INFO  fakecontroller      disarmed, none
    0.000 INFO  pilot               loading flight 1
    0.000 INFO  pilot                   waypoints: 4
    0.000 INFO  pilot                   home: 47.466540, -122.308000
    0.000 INFO  pilot                   night: 0
    0.000 INFO  pilot               setting up camera
    0.000 INFO  pilot               handing off to flight loop
    0.000 INFO  flight_loop         preflight: MSP 0, INAV 2.5
    0.000 INFO  flight_loop         nav wp is flag 11
    0.000 INFO  flight_loop         gcs nav is flag 15
    0.000 INFO  flight_loop         preflight: waiting for GPS fix
```

Figure 6.9: Pilot software running on the X-Plane simulator

The AI implementation was fully realized through the development of a custom data pipeline and a YOLOv8 Convolutional Neural Network. To handle the scale of the G2F UAV dataset, a programmatic data pipeline was developed to automate the image labeling process. This involved writing an *auto_labeler.py* script that utilized Excess Green (ExG) index algorithms to calculate fractional canopy cover, as well as HSV color-space mapping to detect late-stage biological markers like tassels and crop dry-down. To ensure high data integrity, an OpenCV "dirt filter" was integrated to eliminate false positives, preventing non-biological noise such as tire tracks or bare soil from being misclassified as mature corn.

Model training was orchestrated on the university's Nova High-Performance Computing (HPC) cluster. This required developing specialized Slurm batch scripts to sequentially download, process, and clear data batches to overcome strict 10GB storage quotas on the cluster. The YOLOv8 model was trained to classify six distinct biological growth stages, plus a background class. Following 50 epochs of training, the model achieved a 99.3% validation accuracy on the 24,000-image prototype dataset.

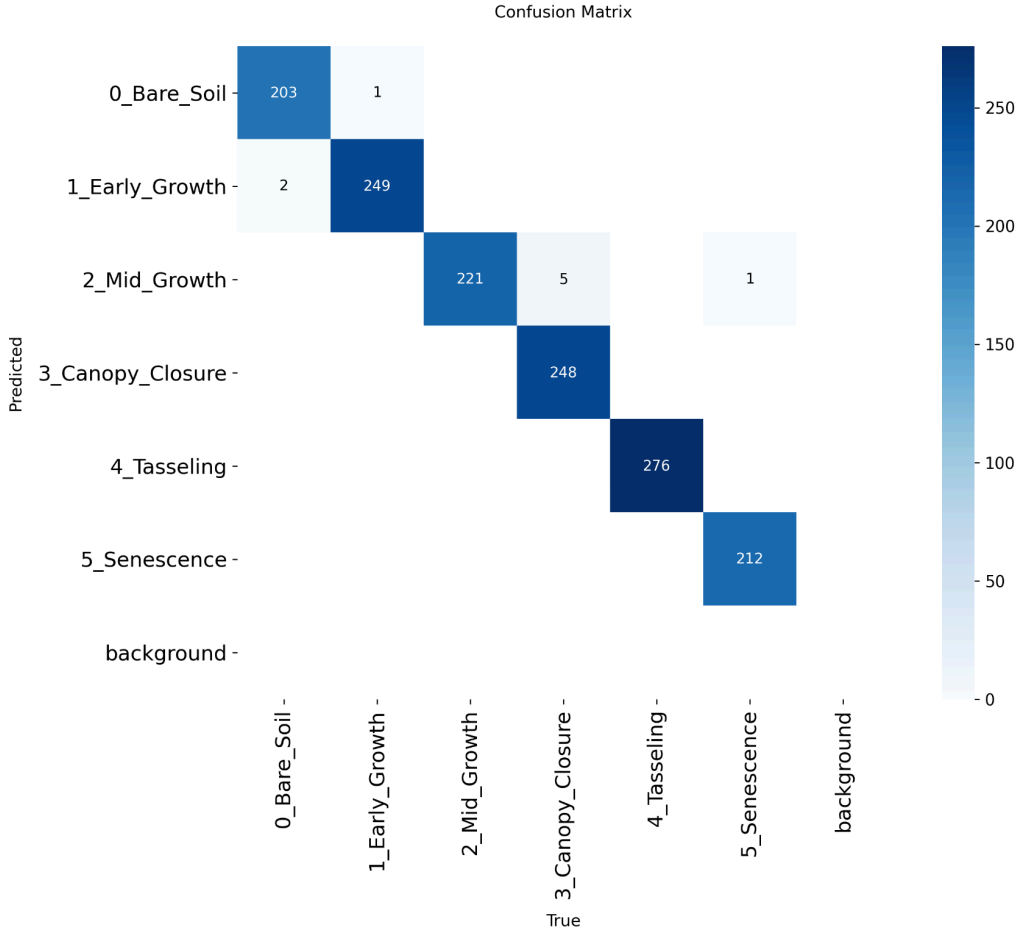


Figure 6.10: Validation confusion matrix for prototype YOLOv8 6-class corn growth model

As demonstrated in the confusion matrix above, the model maintains exceptional precision across the growth stages, successfully filtering out background noise and only exhibiting minor confusion along the immediate boundaries of biologically similar phases. Finally, the optimized model was deployed into a Python microservice, establishing a seamless loop to receive images from the backend API, perform real-time inference, and return analyzed growth data to the user's dashboard.

6.1. Design Analysis

From a design perspective this project has accomplished all of its goals set at the beginning, in some form or another. The full system can be broken down into 4 different parts: hardware, frontend, backend, and AI software.

The frontend has accomplished its goal of creating a functional and secure user interface, offering users an effective way of utilizing the software and creating flight plans. The hardware team properly implemented a functional drone with open source software for flight control, and custom made data collection software to allow the data to be analyzed by the system. The backend properly handles all operations that include data transfer, storage and manipulation, allowing the client app to store data, and communicate with the separate AI service that handled data analysis. Our separate AI analysis service correctly defines different growth stages specifically for corn crops, allowing individuals to track the progress of their crop through the software.

7. Ethics and Professional Responsibility

This project defines engineering ethics and professional responsibility as a commitment to safety, compliance, accountability, and integrity throughout the entire system development lifecycle. Our team adopted a proactive approach to ethical decision making by embedding these principles into planning, implementation, and testing processes. We emphasize responsible innovation, particularly in areas involving hardware integration and real world deployment.

From an ethical standpoint, the development process was guided by careful planning, thorough research, and consistent communication among team members. Each phase of the project was executed with attention to safety, reliability, and transparency. The integration of hardware components, including the use of open source software and commercially available drone parts, required heightened diligence due to the potential risks associated with physical systems. This challenge encouraged the team to adopt more controlled testing procedures and to prioritize safe engineering practices.

Regulatory compliance was a central consideration throughout the project. The team ensured adherence to guidelines established by the Federal Aviation Administration, as well as university policies governing drone usage. Access to a licensed team member with knowledge of aviation regulations significantly improved our ability to conduct testing responsibly and within legal boundaries.

Hardware validation followed a structured and methodical approach. Individual components were tested independently to verify proper functionality and identify defects before system integration. Once validated,



components were combined and tested as a complete system within a controlled and regulated environment. This process reduced risk and ensured that the system met both performance and safety expectations.

At this stage of the project, our ethical framework remains consistent, as the core challenges and responsibilities have not significantly changed. However, our awareness of safety considerations and regulatory requirements has deepened, leading to more disciplined testing practices and stronger adherence to professional standards.

7.1. Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	Definition	Relevant Item from ACM CoE
Work Competence	The skills and effort to complete work at a high quality and with efficiency.	2.1 and 2.2, we're all trained in some ways to help contribute to different areas of the project. We've been making progress in all areas of our project while maintaining a high level of work. As our project has continued to become more integrated and advanced, it's pushed us to a higher level of work competence.
Financial Responsibility	Take care of the parts and pieces that we spend money on, be frugal when purchasing.	1.5, by respecting the work required to make the parts for our drone, we are also respecting the money it costs to buy it. We took many precautions during assembling and testing of the drone to ensure that no damage comes to the drone, these precautions were successful, showing we take financial responsibility seriously.
Communication Honesty	Have open communication between each other and especially with Maruf about project progress.	1.3, we update Maruf every week with the progress on our project. If we're stuck or need help we tell him. We have a shared discord server where we constantly check in and update each other on progress, adding another layer to our communication honesty. This has only gotten better over time, as a team we're much better at communicating to each other as integration is a key part of success for our project.
Health, Safety, Well-Being	Learn from other drone safety issues, take our drone build and testing serious while prioritizing safety over progress.	1.2, We've been in meetings and discussions with Dr. Jones about drone safety and what it looks like in our project. We've learned how to slowly iterate to test functions of our drone so that no one gets hurt.
Property Ownership	Don't damage buildings or crops with the drone.	1.2, avoiding harm is broad, but here we're applying it to avoiding harm of property. We've taken many steps in testing and deployment to ensure that our drone won't cause permanent damage to buildings or crops. In testing this was done by using a rope, ensuring our protocols in place for the drone function properly, then after that we are able to rely on our drone's backups & protocols to ensure no damage.
Sustainability	Shut off the drone	1.1, part of contributing to society also includes not leaving a

	when done and don't have debris fall off the drone.	trace. Our drone won't leave debris, pieces, or anything of that sort while flying over fields and taking pictures.
Social Responsibility	Only gather pictures of crops from farmers who consent to it.	1.6, while we haven't been able to take any pictures of crops yet due to the full flight loop not being done, we will respect the privacy and wishes of farmers and only use the drone on their fields that they want us to.
Confidentiality	Users should be able to see their information and their information only.	1.7, confidentiality was a huge concern for us as we implemented user login & authentication, without proper confidentiality there were concerns that malicious users could backdoor some information as user IDs are used in API requests. To mitigate this, we used AWS Cognito to ensure that no data is leaked to any users, user IDs aren't available in plain text and that all users are protected.

Table 7.1: Table for areas of professional responsibility

An area that our team is performing well in is work competence. We're working in around five different areas of our project and all five areas are making significant high-quality progress. We've got specialists in every area to help ensure that we are producing competent work. One area that our team needs to improve in is communication honesty. This is not because there has been any dishonesty or communication, but rather there is the most room to improve on this area. We need to perform more thorough code reviews and also update each other more often on our progress. Our communication with Maruf is flawless so far. This continued to be the case for the second semester, we had great communication in our team and with our advisor, while pushing our work competence levels even higher.

Data Integrity and Licensing (AI Model): As the project evolved to include custom AI training pipelines, we strictly adhered to ethical data sourcing. We ensured that all UAV imagery sourced from public repositories (such as the G2F dataset via CyVerse) was used in compliance with their research licensing and contained no Personally Identifiable Information (PII).

7.2. Four Principles

	Beneficence	Non-Malificence	Respect for Autonomy	Justice
Public health, safety, and welfare	By improving crop health, can increase protection for crops.	Drone has safety features to prevent harm.	Drone will take commands at request of user. Apps will not take user data.	Healthy crops promote safe food production for multiple farms.
Global, cultural, and social	Food can be produced faster.	Follow flying laws.	Communities can modify drone.	Higher quality crops can be obtained for more



				farmers.
Environmental	The project helps the crops by identifying their potential diseases.	Reduction of chemicals that is used on crops, electricity for drone instead of heavy machinery.	Farmers can efficiently choose how they want to grow crops with the information obtained from drone.	Healthier crops will lead to healthier farms across communities.
Economic	Increase yield and reduce waste or crops that have be thrown out.	Reduce crop failure risk.	Drone is not proprietary, and user can modify as needed.	The jobs of people that may go out and perform this labor would be replaced by the drone.

Table 7.2: Table of the four principles

The most important context-principle pair for this project is safety and non-maleficence. In regards to an Autonomous Drone, there needs to be precautions taken to not cause any harm to the user. The blame would fall entirely on the team if there is an error that causes the drone to potentially severely injure a user or anyone nearby. We have working to ensure safety by implementing safety features such as a kill switch and rigorously testing the flight capabilities of the drone to catch any unexpected flight patterns.

A broad context-principle pair in which the project could improve is global and non-maleficence. While the team has taken steps to ensure that the process of flying a UAV is being met, there could be improvements to really secure the flight control so that nothing unexpected occurs with flight and local and national flying laws.

7.3. Virtues

Three virtues that are important to the team are cooperation, friendliness, and reliability. Cooperation is the sharing of knowledge and willingness to work with others. Cooperation has been supported in the team within the different portions of the project such as hardware, frontend, and backend. All these sub-teams work with each other to create a better version of the project than if we had not cooperated well. This extends to the entire team, as there has been a willingness among the team to help out when needed. Friendliness is sharing a bond with others that is not just a camaraderie, team type of relationship, but one that exceeds that, where it is fun to work with the members of the group. This is always being highlighted whenever the team meets for the project, as well as the “group bondings” that the entire team participates in. The stronger bond that is shared between us allows us to open up to each other more about any help or vulnerabilities that we have. The last virtue of reliability is the setting of expectations with the team which is then lived up to by each member. Reliability has appeared through the continuous attendance of team meetings by each member as well as the technical deadlines being met.

- Alex
    - Initiative



        - I improved on this this semester by being more willing to take ownership of tasks without waiting to be directly told to do them. Instead of only focusing on what was immediately due, I tried to look at what would help the team make progress overall and then start working on it. I also made sure that when I noticed something that needed to be fixed or added, I would take the first step rather than waiting for someone else to bring it up.
    - Foresight
        - I improved on this by paying more attention to the future goals of the project and thinking through what work would need to be done before we got there. I tried to plan ahead more, whether that meant preparing documentation, thinking about possible issues with implementation, or making sure my work would be useful for later parts of the project. This helped me stay more aware of how my individual work fit into the bigger picture of the team's progress.
- Caleb
    - Attentiveness
        - Attentiveness is important because without prompt response from each member of the team, code reviews and feature changes would take longer, delaying the project exponentially.
        - I have demonstrated attentiveness by responding to requested frontend code reviews that pass our CI/CD pipeline within 24 hours of the request. I have also begun work on features that interact with the backend as soon as they are ready to proceed. In these ways, I help minimize downtime and help keep the team moving forward
    - One virtue that I still have to work on is moderation
        - Improving my moderation is important because I oftentimes struggle when the frontend UI isn't responding or reacting how I want it to. This can lead to me making the situation worse by trying a bunch of random ideas to see if any of them work, and then adding bloat to the project by forgetting what I've tried and what packages I might've accidentally added by trying those solutions.
        - To improve my skills in moderating myself, I recognized the need to take a step back when the project isn't adapting how I think it should be, and take a moment to analyze what I've already done, then proceed with those new changes in mind.

- Srdan
    - Curiosity
        - Throughout this project I let curiosity drive my contributions from a technical and non-technical perspective for the project. Taking on a technology I have never used previously drove my passion for researching, implementing, and testing various systems and approaches to solving the problems our team set as a goal.
        - Over the course of this project, my curiosity drove me to consistently put out suggestions for the rest of the team, try to learn how different groups within our team were implementing their solution, and what improvements we can make in various parts of our system that would benefit us in the end.
    - Self-Control
        - Having self-control is important in order to maintain professionalism, but also help better guide the rest of the team. I had lacked self-control in certain cases



when I got ahead of myself, and wanted to implement various functionalities for this project that were not realistic within the timeline we were working with.
        - In the future, I intend to do a better job hearing my teammates' thoughts on why something might not be the best fit for us or why we should take a different approach to implement something differently from what I had imagined, which will lead to a better overall development experience.

- Andres
    - Humility
        - This is important to me because I like to view myself as never being the smartest person in the room so I can continuously learn from others that are far greater than myself. With this perspective, it allows me to ask questions and multiple of them, while staying humble to my own work and knowledge to reciprocate any knowledge given towards me in a group project.
        - I have received guidance from my peers for multiple topics of the project. Since I am not familiar with many aspects of the project, they have taught me how the project works in regards to their expertise. Even in my own area of hardware, others in this project have shown me a lot that inspires me and helps me out.
        - I think I still did well to preserve humility within the project. I was always still happy and open to my friends talking to me even when they wanted me to do something quickly. I wish I could have lived up to them better than I have.
    - Dedication
        - Dedication is important to me because it shows that there is passion into a project and when there is passion, a great project can be made. While I was and still am interested in the project, other classes and a lack of discipline has not led me to working as much on the project as I could have this semester. This leads to me not showing that much work in what we could have shown at the end of this semester.
        - To demonstrate dedication I need to put time into the project and even if there is a hold in the project that is outside of our hands like what has gone on during the semester, I need to continue to surround myself with ideas of the project and give a helping hand to other sections of the project.
        - Well, well, well, Andres did not live up to his dedication that he thought he would. After spring break rolled around, I was losing momentum to finish the project. I wish I hadn't since last week would have been a lot nicer if there was a lot more work done before then. Would have not have had to do anything towards the end!

- Jongmin Lee
    - Perseverance
        - Why is it important to you?
          Proactive communication is important because our project relies heavily on cross-team coordination, especially between the AI, backend, and frontend groups.



        - How have you demonstrated it?
          I improved at proactive communication by sharing consistent progress updates in our team channels, especially when running long model training experiments on the cluster or discovering issues with our dataset. By signaling blockers, such as the 2019 file corruption, earlier to the team, I helped prevent integration delays and ensured the backend team knew exactly what JSON formatting to expect from the AI microservice.
    - Proactive Communication
        - Why is it important to you?
          Proactive communication is important because our project relies heavily on cross-team coordination, especially between the AI, backend, and frontend groups. When updates or blockers are communicated early, it prevents wasted work and makes integration much smoother. Improving this virtue ensures the team stays aligned and avoids bottlenecks.
        - What might you do to demonstrate that value?
          I can demonstrate more proactive communication by sharing consistent progress updates in our team channels, especially when running long experiments, discovering issues with data, or identifying changes that may affect the backend or UI. I also plan to signal blockers earlier and request help instead of trying to solve every issue alone. This will improve team workflow and reduce integration delays.

- Camden
    - A virtue I have demonstrated is craftsmanship.
        - This is important to me because I always want to produce the best possible work I can, spending lots of time researching and making sure everything is well-thought-out, maintainable, and reliable – least likely to cause any issues down the road, and most likely to help the rest of the group in their own work.
        - I have worked hard this semester to produce well-documented systems that show an attention to detail in their design, making sure the team can understand them. I have also done lots of additional work to improve the development experience with thorough CI pipelines, well-automated and tested systems, and lots of documentation.
    - A virtue I have improved in is collaboration.
        - I have a tendency to work heads-down on a task I have through to completion, but in a large group it's important that we all spend time working through designs together and splitting up tasks to ensure that the work we individually complete will work well with the larger product and that everyone gets to have a part in it.
        - Throughout the year, I got better at supporting group members with tasks, and discussing progress and implementations with the group in our weekly meetings, so that each part of the project could be refined and made better in the context of the whole project.
- Z
    - A virtue I have demonstrated is willingness to fail.
        - This virtue is important to me because it shows that I want to learn and grow as a developer. One of the necessary traits to have as a software engineer is to want



to learn. The chase for knowledge and skills is a major motivator for me as an engineer. I am happy to have demonstrated this virtue throughout this project.
        - I haven't worked on a React frontend before so being assigned to frontend was a completely new experience and there has been a lot of failing, but also a lot of learning. I think the biggest way I've shown this virtue is the progress I've made on the frontend and the questions I've been asking of my teammates that are more familiar with React, throughout the project I've solo implemented some critical and complex functionality for the frontend. Google Maps API integration, and field snapshotting were my two pieces that took a lot of failing, but I've shown that I'm better because I wasn't afraid to fail, and in the end, both work flawlessly.
    - A virtue that I improved upon this project is moderation.
        - Moderation is an important virtue because lack of moderation means that you are either putting in way too much effort or way too little. In my case it's a bit of both. Due to my schedule outside of school I have weeks where I contribute a lot to the project and weeks where I do a lot less. This leads to me being an inconsistent teammate which I don't want, which is why I chose this virtue to be better at.
        - I have improved greatly in my moderation this last semester, and I've shown it by being more consistent, and setting time aside to work on the project at the beginning of the week. I had much more frequent work sessions, but they weren't as long. I also tracked my improvement in moderation by keeping a record of my merge requests. The first semester I had one huge merge request, this semester I had 5 smaller ones. I believe consistency is key in life and I want to strive to be better in that aspect. I believe I've shown improve in my moderation which is exactly what I set out to do.

8. Conclusions

8.1. Summary of Progress

By the end of this project we have developed a functional Drone -> Frontend <-> Backend <-> AI model pipeline for image transfer, as well as established a method of creating fields, assigning and planning flights, and analysing images for information. We are also able to display almost all of this information to the user in an app format that most users should be familiar with. The main feature that is left to be tested remains to be images taken on an autonomous flight. There could also be improvements to the flight. Features there were not implemented were corn health or soybean health and crop monitoring. We also can not capture night time data.

8.2. Value Provided

The goal of our project was to make a drone and software system that could fly and take pictures of the farmer's field, and then analyze those images for growth stages and pests. Currently, we are able to do all of those things to varying degrees of accuracy. Our drone is fully capable of flying to a designated GPS

location, and we can pass those locations from the frontend in the form of fields that can be stored and retrieved from the backend API, as well as take image captures and analyze them with an AI model.

8.3. Next Steps

Future projects building off of our work could take place during the spring/summer seasons, enabling them to reach out to farmers and collect image samples from actual fields as well as get user feedback for UI/UX usage. They could also work on cleaning up the frontend UI to make it look nicer while still maintaining usability. Additionally, future projects might train the AI model using crop images taken from the drone, or use larger pre-analyzed datasets, as any place that we attempted to reach out to declined to provide us with any of their data. The drone sensors can be further calibrated to ensure that the flight is smoother and then test it out on a field, where there are no obstacles. We would absolutely love to hear about any progress made to the design of this project, and would be thrilled to assist in any way possible to the future members of the project.

9. References

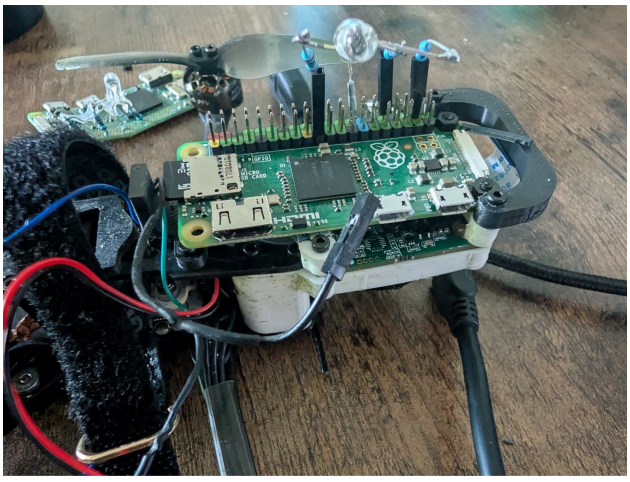
List technical references and related work / market survey references. Do professional citation style (ex. IEEE). See link:

https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf

[X] Glenn Jocher, Ayush Chaurasia, and Jing Qiu, "Ultralytics YOLOv8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

10. Appendices

We created a number of CAD files to mount the extra components to our Flywoo Explorer LR drone. They were 3D printed in TPU filament. All files are available in the Git repository under the `Docs` folder. An assembly guide is also included in the repository.



Camera mount. This attaches in place of the factory DJI O4 mount, and mounts to the four screw holes on the camera.

Pi mounts. These are two small mounts that go over the camera screws and provide attachment points for the Pi.

Battery stopper. This was added after we noticed the battery slipping towards the Pi, eventually putting significant pressure on the SD card slot.



Camera cable guard. This was added after the drone hit an object directly over the Pi's camera cable, causing it to snap off.

The INAV documentation was extensively used in this project, in implementing MSP commands, learning how to calibrate and control the drone, and debugging issues. See https://github.com/inavflight/inav.

10.1. Appendix 1 – Operation Manual

Development Setup

Before operating the drone, several system components must be running. The following steps will set up each of these systems under a development environment. Additional development details are available in the Git repository's documentation, https://git.ece.iastate.edu/sd/sdmay26-20.

Setup

- Clone the Git repository.
  ```
  git clone https://git.ece.iastate.edu/sd/sdmay26-20
  cd sdmay26-20
  ```
- Install Task.
  https://taskfile.dev/docs/installation

Backend

- Deploy the Terraform resources in the Terraform folder with `terraform apply`.
  - Record the Cognito region, user pool ID, and client ID from Terraform or the AWS console.
- Install .NET >= 9.
- Configure a PostgreSQL user and database.
- Modify `Backend/AutomationDrone.Backend/AutomationDrone.API/appsettings.Development.json`:
  - `ConnectionStrings -> FlightDB` with the PostgreSQL host, username, database, and password.
  - `Authentication -> Cognito` with the Cognito details found above.
- Start the backend.
  ```
  task backend:api:run
  ```

AI Service

- Install Python >= 3.9.
- Visit the AWS console (after Terraform has been applied with the backend) to find the following:
  - The Cognito endpoint
  - The Cognito client ID
  - The Cognito client secret



- Create a `.env` file in `Backend/AI` with the following environment variables set:
  - `COGNITO_ENDPOINT`
  - `COGNITO_CLIENT_ID`
  - `COGNITO_CLIENT_SECRET`
  - `COGNITO_SCOPE = ai-service/work`
  - `API_URL` (full URL where the backend can be reached)
- Start the AI service.
  `task backend:ai:run`

Frontend

- Install Node.JS.
- If the backend URL has changed, set it on line 11 (`const api`) of `Frontend/AgDrone/src/functions/APIRequests.ts`.
- Start the frontend.
  `task frontend:run`

Drone Pilot Software

- Install gcc, cmake, make, libgtest-dev, libturbojpeg-dev, and nlohmann-json3-dev.
  - If you're running on the live drone Pi, you'll also need libcamera-dev and WiringPi. If you're using a pre-built Pi image (see below for pipeline setup), these are preinstalled.
- Run the drone software.
  `task drone:run`

Production Setup

The entire system is intended to be deployed to production automatically via GitLab CI pipelines. These trigger whenever code is merged into the main branch, and will create frontend binaries, deploy the backend, or generate ready-to-use drone software images whenever those components have changed.

If the CI pipelines are not ready, the following steps can be taken to set them up.

Prerequisites

- A server to run the backend and AI model. We chose a 2CPU/8GB RAM server from ETG running Ubuntu.
- For Raspberry Pi image building, any 64-bit OS Raspberry Pi.
  - This is possible to do with the Pi Zero 2 W on the drone, however it will be very slow.
  - We used a group member's Pi 5 for this, which resulted in sub-10 minute OS build times.

Steps

- Terraform pipeline.
  - Create an AWS service account for use with Terraform.
  - Set the following GitLab CI variables:



    - AWS_ACCESS_KEY_ID
    - AWS_REGION
    - AWS_SECRET_ACCESS_KEY
- Backend deploy pipeline.
    - Set up a shell-based GitLab Runner on your backend server. See https://docs.gitlab.com/runner/install/.
    - Set the runner's tag to `deploy`.
    - Install .NET >= 9.
- Raspberry Pi OS image building pipeline.
    - Set up a shell-based GitLab Runner on your Pi. See https://docs.gitlab.com/runner/install/.
    - Install `rpi-image-gen` dependencies by cloning https://github.com/raspberrypi/rpi-image-gen on the Pi and running `./install_deps.sh`.
    - Set the runner's tag to `pi`.
    - Set a GitLab CI variable named `PI_PASSWORD` with the password to set for the `pilot` user account on built images.
    - The ECE GitLab instance does not allow uploading artifacts as large as the Pi OS image. The current pipeline contains a workaround that uploads the artifacts to a Google Cloud Storage bucket so they are available. This can be easily adapted to an AWS S3 bucket paired with Terraform by updating line 82 in `Drone/ci/ci.yml`. Otherwise, a Google Cloud bucket must be configured and signed into on the Pi and the bucket name on line 82 must be changed to point to that bucket.
- Frontend build pipeline.
    - The ECE GitLab instance does not allow uploading artifacts as large as the built frontend. The current pipeline contains a workaround that uploads the artifacts to a Google Cloud Storage bucket so they are available. This can be easily adapted to an AWS S3 bucket paired with Terraform by updating line 15 in `Frontend/ci/ci.yaml` (to install new dependencies) and replacing `Frontend/ci/upload_artifact.js`. Otherwise, a Google Cloud bucket must be configured, and the GitLab CI `ARTIFACT_KEY` (service account key) and `ARTIFACT_BUCKET` (GCS bucket name) variables must both be set.

XPlane Simulation

INAV contains a SITL (Software in the Loop) simulator that can link to the XPlane 11 flight simulator software to test the drone pilot code before running it directly on the drone. The drone software is configured with a number of mock interfaces (fake controller, camera, UART connection, and LED) that enable fully testing the software with XPlane and INAV SITL.

Prerequisites

- Get XPlane 11. It is cheapest to buy it from Steam during any Steam sale ($10).
- Install INAV Configurator version 7.1.2.
- Follow the development setup for the drone pilot software above on your local machine.



Setup

- Set up the INAV XPlane HITL plugin.
- Start XPlane.
- Start a flight with the INAV Surfwing plane.
- Open INAV Configurator.
- Go to the SITL tab.
- Set the XPlane port to 49000.
- Start SITL.
- If you haven't configured it yet:
    - Select SITL as the device in the dropdown next to the Connect button in the top right.
    - Connect to the drone.
    - Configure as recommended by INAV.
- In XPlane, enable the INAV OSD (optional):
    - ESC -> Plugins -> INAV HITL -> Link -> Port 5760
- Disconnect the INAV configurator if you connected it to configure the drone, or set up a third TCP UART connection.
- Start the drone software.
  `task drone:run 127.0.0.1 5761`
- Use the fake controller to arm the drone or change modes (see terminal documentation after starting, press a key and hit enter).

Drone Configuration

The drone runs INAV 7.1.2. To configure it, install INAV Configurator 7.1.2. Connect via USB with the default baudrate.

Drone Operation

SD Setup

- Download the latest built Raspberry Pi image (see pipeline setup above).
- Use an imaging program to write the image to an SD card
- Plug the drive into the drone's Pi, then turn it on
- Wait for it to reboot, then the status LED on top of the Pi to turn on
- Shut down the drone

Controller Mappings

As the drone is configured now, the following buttons change between drone modes:

- SA: arms the drone (enables motors, allows flight)
    - If the blue status LED on the drone's flight controller is blinking, arming is impossible as INAV has detected an issue. Usually, this is because it has too few GPS signals, it's not level, or another mode is turned on.
- SB: switches between normal drone modes
    - Off: normal (acro) mode, full user control



    - Mid: NAV ALTHOLD, which maintains altitude while allowing the user to control position
    - High: NAV POSHOLD, which maintains a full 3D position lock
- SD: toggles NAV WP mode, which begins data collection if the Pi is connected

Drone Use

This is the intended flow for user operation, if you are reading this as a future member of the project or are an aspiring drone operator you most likely are smarter than us but take this precaution still, please make sure to operate the drone with heavy safety considerations in mind. While it's fun to fly a drone and especially when it's autonomous, it's not fun when anybody gets hurt. With that in mind:

- Plug the drone SD into a computer
- Start the frontend
- Go to the Data Manager tab
- Create a new flight
- Configure a home waypoint and all intended image waypoints
    - Altitude is relative to where the drone was first armed.
- Plug the SD card into the drone's Raspberry Pi
- Plug the battery into the drone
- Wait for the Pi to start and prepare the drone for flight
    - The status LED on top of the Pi communicates the flight status:
        - Off: program has not yet started
        - Cyan: Pi is communicating with the drone
        - Green: Ready for takeoff
        - Yellow: Takeoff condition not met (requires reboot)
            - No flight configured, latest flight already completed, drone not configured properly, etc.
        - Red: Unhandled program error
- Once the LED turns green, arm the drone, apply a bit of throttle to get it in the air, then enable NAV WP mode
- The drone will take pictures and return to the configured home location
- Disarm the drone, then turn it off
- Plug drone SD into a computer
- Visit the Data Transfer tab and select the SD card to view the flight log
- Visit the API Testing tab, configure the flight ID, and press "upload flight images" to run AI analysis in the backend

10.2. Appendix 2 – Alternative/initial version of design

Since beginning work, our project has largely stuck to the initial design without any significant changes. Some major changes occurred during the planning phase as we considered the best way to meet the client's requirements within our budget and with maximal compatibility and safety.

The primary major change, detailed in section 4.2, was the drone we were using. We had initially planned for a large custom build for optimal battery life and customizability. After safety discussions and meetings, we decided to pivot to a small pre-built drone to ensure that the drone could not cause any harm or damage. This caused significant delays, but our team was ultimately able to adapt to this, finding a drone



within our budget with an acceptable battery life, running an open-source drone flight control system that we were able to overwrite and configure to work with our drone. We finally had to adjust our designs to find creative solutions to mounting and wiring our extra components to the drone without risking damage to any of those components or throwing the drone off balance.

Another change to our initial design was how the frontend was developed. We had initially planned on using .NET MAUI, but at the time of the project starting MAUI did not support Linux. This caused issues as it would prevent us from creating CI pipelines on the Linux-based GitLab runners, and would prevent a group member from developing the frontend. We chose to pivot to Electron with React to resolve this, and ultimately were able to continue development without any noticeable impact to our timeline.

10.3. Appendix 3 – Other considerations

- The drone crashed a number of times, and around 6 of them were somewhat hard crashes with one being in water. No damage occurred to the drone itself, and we did not break a single propeller. However, one crash directly on the Raspberry Pi snapped it almost in half and made us install a replacement.
- Bill Gates is stage 3 corn growth, with only about a 50% accuracy. Our drone's mock camera interface returns a tiny image of Bill Gates when run, which when fed through our AI model results in Stage 3, Canopy Closure.

10.4. Appendix 4 – Code

https://git.ece.iastate.edu/sd/sdmay26-20

10.5. Appendix 5 – Team Contract

Team Members

1. Caleb Zea
2. Andres Lopez
3. Srdan Kopunovic
4. Z Harvey
5. Camden Senneff
6. Alex Pallan
7. Jongmin Lee

Required Skill Sets for Your Project

- Frontend:
    - Typescript usage
    - CSS usage
    - How to use and implement backend APIs
    - Data security implementation
    - Data security best practices



- Backend:
    - C# usage
    - API creation
    - API implementation
    - Database connection
    - AWS knowledge
    - Data security implementation
    - Data security best practices
    - Terraform knowledge
- Drone hardware:
    - Part compatibility
    - Safety procedures
    - Raspberry Pi programming knowledge
    - Raspberry Pi hardware knowledge
- AI:
    - Model building
    - Data Collecting and labeling
    - Model training
    - Backend integration

Skill Sets covered by the Team

Data security, AWS setup, Terraform: Alex

Backend API, User Authorization, AI backend integration: Srdan

Frontend API integration, Typescript and CSS, frontend-backend API integration: Caleb

AI model building, dataset training, dataset labeling, dataset organizing: Jongmin

Part building, safety procedures: Andres

AI backend integration, Raspberry PI code, Raspberry Pi hardware: Camden

Typescript, CSS, frontend-backend API integration: Z

Project Management Style Adopted by the team

We are using an Agile management style



Initial Project Management Roles

Camden: Technical planning and organization

Andres: Hardware

Alex: Cyber Security (encryption, auth, etc)

Caleb: Frontend

Jongmin: AI, Frontend

Z: Software jack of all trades

Srdan: Backend, Middleware

Team Contract

Team Members:

1) Caleb Zea 2) Camden Senneff

3) Srdan Kopunovic 4) Z Harvey

5) Jongmin Lee 6) Alex Pallan

7) Andres Lopez

Team Procedures

- Day, time, and location (face-to-face or virtual) for regular team meetings:
    - Advisor meeting on Wednesdays at noon
    - Full team meetings 11am Saturdays, in-person preferred (location TLA), online as needed

- Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    - Discord

- Decision-making policy (e.g., consensus, majority vote):
    - Majority vote



- Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    - Take turns, alphabetical order (Andres > Caleb > Camden > Min > Srdan > Z > Alex)

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

- Be there and be on time, unless communicated with enough advance notice, ideally notifying the group at least 2 hours prior to meeting.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

- Commit to the team assignments that are requested each week and successfully accomplish them. However, if they cannot be completed, report why, and be prepared to have teammates help you resolve any issues.

3. Expected level of communication with other team members:

- Asking for help from the team when/before it is needed. Team members should communicate how they are planning to complete their team assignments with their role members and when they have completed their assignments to the team.
- Communicate if not able to attend meeting/lecture (at least 2 hours prior).

4. Expected level of commitment to team decisions and tasks:

- Actively participate in meetings and decision making.
- Ensure all tasks are completed by the set due dates, and if to communicate any issues that could potentially slow team progress

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction,

individual component design, testing, etc.):

Camden: Technical planning and organization



Andres: Hardware

Alex: Cyber Security (encryption, auth, etc)

Caleb: Frontend

Min: AI

Z: Software jack of all trades

2. Strategies for supporting and guiding the work of all team members:

- Hold each other accountable
- Keep track of each other's contributions and reach out if someone is falling behind
- In the weekly meetings, talk about what we want to get done and what we need from each other to accomplish that

3. Strategies for recognizing the contributions of all team members:

- Have team members discuss their contributions of the week and successes of or why they have not accomplished their task
- Check off sections of the planned timeline for the project, including each contributor's accomplishments.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

- Andres - Embedded Systems, made drones fly before, taking machine learning. I like to code in C. I like hardware as well as background with circuits.
- Srdan - Embedded Systems, solid amount of backend experience, skilled in Python for processing large data sets, good hardware knowledge and an ability to properly set up separate systems to communicate with one another.
- Caleb - Embedded Systems, backend database knowledge, frontend functions, some security knowledge.
- Min - AI, specifically language model and machine learning, Python for AI and data, frontend experiences, preferred language is Java, and Python.
- Camden - I have a lot of experience developing secure backend systems and APIs, and I am additionally comfortable writing software for embedded systems, developing frontend applications, and working with AI. I am most experienced in Java, Python, C/C++, and Go.



- Alex - Python for data, Cyber Security concepts and applications, Infrastructure management (Cloud, physical systems infrastructure), Frontend Development, FAA Certified Remote Pilot.
- Z - Embedded Systems, frontend and backend experience in Java, Angular, C++, Verilog. No matter what stage of development we're in I'll be able to help further our progress. Confident in everything except pure hardware.

2. Strategies for encouraging and support contributions and ideas from all team members:

- Create a concrete plan of execution
- Support each team member when they need help
- Collaborate for proper implementation
- Communicate well with members working on connected parts of the project

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

- Bring it up during a team meeting, and the group as a whole can discuss how we can fix the problem
- If there is a larger-scale issue, then report to Dr. Ahamed

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

- Having a prototype that has some if not all of the features implemented to some capacity.
- Collaborating together to create a design that we can be proud of

2. Strategies for planning and assigning individual and team work:

- Define frontend/backend roles to separate into smaller teams for better coordination.
- Breakdown tasks within those teams and assign them to individuals.
- Assign larger tasks as a whole team

3. Strategies for keeping on task:

- Creating a rough timeline of each task, and goal.



- Working on the project in phases, with the goal of implementing smaller sections of the program with each other, before moving on to other sections.
- Proper communication, use of version control software, and anything else that can support us in completing our work.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

- Any repeated failure to adhere to the contract will be discussed in our weekly team meeting with the group to attempt to come to a resolution.

2. What will your team do if the infractions continue?

- Discuss repeated infractions with Dr. Ahamed

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the

consequences as stated in this contract.

1) Caleb Zea	DATE 12/5/2025
2) Camden Senneff	DATE 12/5/2025
3) Jongmin Lee	DATE 12/5/2025
4) Srdan Kopunovic	DATE 12/5/2025
5) Z Harvey	DATE 12/5/2025
6) Alex Pallan	DATE 12/5/2025
7) Andres Lopez	DATE 12/5/2025