Florida International University
School of Computing and Information Sciences

CIS 4911 — Senior Project
Software Engineering Focus

Final Deliverable

FruiTREC
An Agricultural Robotics Application

Team Members:
Franklin Abodo
Gabriel Barrs
Miguel Chateloin

Product Owner:
Dr. Wagner Vendrame

Mentor:
Dr. Leonardo Bobadilla

Instructors:
Mohsen Taheri
Dr. Masoud Sadjadi

Abstract

This document presents the information necessary to gain a good understanding of the tools and methods used to implement FruiTREC, a software application that serves to inform the stakeholders of agricultural enterprises, from farmers to research scientists, about the states of their crops (e.g. growth, health, and yield) using unmanned aerial vehicles (UAVs) and computer vision to measure and analyze those states, and a mobile/web client to create UAV flight plans, store recorded data, and generate analysis reports.

The FruiTREC project was motivated by researchers at the University of Florida's Tropical Research and Education Center, who are continuously involved in agricultural experiments. Most recently, they have explored the effects of bioavailable minerals on the horticultural characteristics (HC) of tropical crops, including growth, health and yield. The mineral concoction promises to be an environmentally friendly alternative to fertilizer, but would only be marketable if its use were to maintain or increase positive HC metrics. Assessing its effects on small crops that cover a small area of land is easily achievable using traditional methods (i.e. looking with one's human eyes and touching with one's human hands), but scaling such assessments to much larger crops can be a tedious and hazardous task. The use of UAVs in HC data collection can increase coverage and reduce risk when trees are multiple stories tall. It is also possible to reduce data-entry labor by automating analysis of and reporting on the collected data. The objective of this project is to combine these two tasks into a single agricultural robotics software application.

Table of Contents

Introduction

FruiTREC 1.0 is a combination of drone technology, mobile software API's, and web based storage and viewing for agricultural purposes.The idea is to create a system that allows a user to map a desired agricultural field i.e crops on a mobile device. Essentially the drone would fly this field while recording video in order to gather useful information, the core being the fruit yield for this first iteration, by using Computer Vision techniques to essentially apply a scan process that will attempt to count the fruits over the lapse of the video. This useful information would than be stored and viewable on a mobile friendly website with all the pertinent information to the user and update over multiple uses.

Current System

The FruiTREC team worked on the first iteration of this system. Prior to this, there was no system designed to aid in crop data collection for the University of Florida's Tropical Research and Education Center. Farmers at the center collect horticultural-related data manually by walking through crops and recording their observations. They then take those recordings and enter them into digital spreadsheets, where they employ formulas to make estimations about crop yield. This activity is carried out once a month on average.

Purpose of New System

The new system aims to reduce the amount of time it takes for farmers to do their data collection, and to reduce the risk of injury associated with data collection. Both aims, if achieved, can allow farmers and agricultural researchers to survey crops that are very tall and cover a relatively large area of land. The system has not yet achieved the ambitious goal of total automation of drone flights, report generation and decision making. But it nevertheless lays the groundwork for these features to eventually be realized.

The core of the problem, the computer vision component, has been partially solved. By narrowing down the scope of fruits to handle, the system can accurately count the number of sapate and longan in images of trees bearing those fruits. The algorithms used apply color-based inductive techniques and texture-based machine learning techniques. The texture-based algorithm can easily be extended to support avocado and coconut images.

The architecture is a well-thought out client-server approach that is set up with the right data storage and database solution to store large video assets and logs from flights, and graphical information. The client software solution has provided some basic service layer modules for communicating with, and controlling the AR Drone. It displays navigation and sensor related information off of the drone in real-time, and allows piloting the drone manually. Flight-related data is initially stored on the device with the intention of later storing it on the server for review,

and to assist in data analysis (e.g. determining the size of fruits in an image as a function of the known distance of the drone from a fruit tree).
User Stories

The following section provides the detailed user stories that were implemented in this release of the FruiTREC project. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

#117 — Research Drone Hardware Platforms

Description:

As a developer, I want to compile a list set of hardware platforms for drone flight so that I can choose one to commit to working with to solve the problem of autonomously gathering data.

Acceptance Criteria:

1. The research should answer the question of whether or not it is possible to replace human eyes and hands with computer vision when gathering information about yield and health from a grower's crops.
2. The platform chosen must meet the follow qualifications:
    a. It must be inexpensive, less than $300 per unit.
    b. It must be able to communicate over some sort of wireless network.
    c. It must be simple to use. It should not take longer than a day to begin piloting it manually.
    d. It must support video recording, storage and streaming to other devices on its network.
    e. It must be safe enough to use without special clothing or equipment.

Related Tasks:

1. Collect and Store Computer Vision Literature and other References.
2. Compile a list of each drone platform and present a case for and against each one.
3. Begin work on #118 and start thinking about the pros and cons of several possible hardware/software combinations.

#118 — Research Drone Software Application Development Tools

Description:

As a grower, I want to discover the drone application development tools that can be used to speed up the development process, if any exist. There's likely to be some frameworks that.

Acceptance Criteria:

1. It can be claimed definitely that the software platform chosen meets the following requirements:
    a. It's compatible with previously chosen drone hardware platform.
    b. It's compatible with the Android application environment.

Related Tasks:

Compile a list of each software platform and present a case for and against each one in combination with hardware platforms discovered in #118.
Present findings to the group and commit to receiving a unanimous and binding decision about which platforms to commit to working with going forward..

#119 — Install Drone Application Development Environment

Description:

As a developer I wanted to install the combination of software and hardware platforms I have chosen to begin implementing my solution on top of this architecture.
Refer at online resources and guides to install all the necessary tools and dependencies we'd need to build any Android application. After completing installation, we should create a simple "Hello World" application and run it on an Android virtual machine as well as a physical Android device. We should also make it easy for other developers to repeat installation process by creating our own installation guide that that includes all the solutions for issues that were troubleshooted in the process.

Acceptance Criteria:

1. An installation guide must be produced which should aim to reduce other team members' installation time.
2. Development tools are installed without reports of fatal errors that prevent development.
3. Should be able to compile a simple Android application
4. Should be able to run a simple Android application on both a physical and emulated device.

Related Tasks:

1. Research tools that will aid installation such as IDEs, extensions.
2. While performing the installation, write a guide that contains all the instructions and commands that must be run, and links to supporting articles if necessary.

Additional Comments:

User Story #127 Setup Android Development Environment was moved into the scope of this story.

#121 — Research Computer Vision Theory, Techniques, and Applications

Description:

As an algorithm engineer, I want to discover the computer vision techniques that can be used to help automate the task of measuring crop yield and health metrics, if any exist.

Acceptance Criteria:

The team can claim definitively that it is or is not possible to replace human eyes and hands with computer vision when gathering information about yield and health from a grower's crops.

Related Tasks:

Collect and Store Computer Vision Literature and other References

#122 Collect and Store Computer Vision Literature and other References

Description:

As an algorithm engineer, I want to collect, store, and make easily accessible information about computer vision techniques that can be used to help automate the task of measuring crop yield and health metrics, if any exist.

Acceptance Criteria:

The team can quickly access computer vision resource material (mostly hyperlinks) from a shared Google Drive folder.

Related Tasks:

Research Computer Vision Techniques and Applications

#123 — Research Computer Vision Application Development Tools

Description:

As an algorithm engineer, I want to discover the computer vision application development tools that can be used to help build computer vision applications more easily, if any exist.

#124 — Install Computer Vision Application Development Tools

Description:

As an algorithm engineer, I want to install all of the computer vision application tools that can help in building my agricultural robotics platform.

Acceptance Criteria:

One demo application included with or created using each of the software tools can be run.

#128 — Test All Available Drones for Working Functionality

Description:

As a developer, I would like to know which drones are fully functional and support the features I care about, so that I can decide which individual units I should develop with and support.

The Parrot AR Drone was chosen as our hardware platform (See User Story #117). We currently have several AR Drones, both 1.0 and 2.0 models, available in the robotics lab. All the drones we have access to are refurbished so there is a chance that some may be defective. The ones that were bought new may also be defective as a result of work done in previous semesters. It's important that we test whether each one is fully functional. A quick way to do this is to use Parrot's FreeFlight application from the iOS or Android app store. It provides a first-person piloting tool. This will also let us practice flight maneuvers and become more comfortable with using our drones.

Acceptance Criteria:

1. For each drone, we must know whether to allow or to exclude them during development.

Related Tasks:

1. Install Parrot Free Flight Application and confirm it work with the first working drone found.
2. Physically label each drone as it is tested with a usable/not-usable message for development. Include any possible questions about problems with individual components, batteries, blades, etc.
3. Create a list describing each drone, it's yes-or-no decision for usability, and the reasoning for that decision.

#129 — Evaluate React-Native Framework

Description:

As a developer, I would like see a prototype a possible user interface so that I can be confident React-Native is the correct choice for development going forward.

React-Native will help us build the user interface component of the Android application with a lot more ease than native methods. In order to become familiar with the workflow, we should dedicate a day to following Facebook's tutorial guide for building a simple React-Native application and spend some time reading relevant documentation. Afterward, we should try building a small prototype that resembles the collective vision of the UI that we've discussed so far. At the very least, we should prove that it is possible and relatively straightforward to build a UI that contains map of the device's current location, allows drawing a shape on that map (resembling a drone's path), and allows clearing that shape to draw another one.

Acceptance Criteria:

1. Display a map of the device's current location when the application is opened.
2. Must provide an interface to draw and clear at least one polygon at a time over the displayed map.
3. Please note that the acceptance criteria specify use cases for the UI of the application. Do not try to implement a feature such as autonomous flight at this time. We are only trying to prototype the UI so we can get a sense for how to build those components in React-Native.

Related Tasks:

1. Follow Facebook's Getting Started Tutorial to get a working React-Native enabled application running on both a physical and virtual device.
2. Create the prototype UI using Google Maps API and any resources that can be found on the web to speed up arrival at a proof of concept.

#176 — Research Parrot's Navdata and Streaming APIs

Description:

As a developer, I would like to have access to the raw navdata and video stream coming from the drone, so that I can record data from flights for later processing.

We need to get a sense for what the navdata and video feed coming off the drone looks like. Parrot's application does not log any information about GPS coordinates which doesn't allow us to confirm that sensors are working and we can stream them. The attributes we most care

about are GPS coordinates, altitude, velocity, and orientation. We confirm that it is possible to receive an accurate real-time representation of these from the AR Drone 2.0. However, we can't waste time writing throwaway code. The best way to accomplish this would be to find existing CLI solutions to facilitate piloted or autonomous flight while also supporting recordings of these metrics.

Acceptance Criteria:

1. At least one video displaying the front or bottom camera view during a flight.
2. At least one flight's worth of navdata in a textual human-readable representation.

Related Tasks:

1. Find some existing software that can enable us to pilot the drone manually or write some scripts to to fly the drone manually with access to logs afterward.

#181 — Integrate AR Drone SDK into Application Architecture

Description:

As a grower, I would like the user interface components and drone API to be compatible, so that the application can be installed to my device without errors.

Acceptance Criteria:

1. Application should contain our package as well as Parrot's package without conflict.
2. Application should successfully build with Gradle.
3. Application should launch a simple screen with text created by us, to confirm the presence of our main activity.
4. Parrot's FreeFlight application example source code should remain in the codebase for reference, but should never be displayed in the user interface.

Related Tasks:

1. In addition to following developer guides provided by Parrot, research frameworks that may give us development speed advantage in implementing integrating the SDK.

#130 — Provide a NavData Service Module

Description:

> As a farmer, I would like to access the state of the drone including data, such as altitude and GPS coordinates, so that I could reflect those attributes in the user interface.
>
> We already know how to read navdata off the AR Drone 2.0 using desktop CLI tools. Now we have to accomplish the same thing within our Android application. We need a native Android module that provides a subscription service for the UI components to access.

Acceptance Criteria:

1. Must provide at least the data on attitude, altitude, gps, battery, accelero, gyroscope, magnetometer and fly state from our scripted flight tool.
2. All navdata provided must reflect the current state of the drone in flight i.e. should all be real-time, no caching.

Related Tasks:

1. Document how to access each navdata property as well as the meaning behind it. Refer to Parrot's official developer guide for the AR Drone 2.0.
2. Create a class diagram, update package diagram for Android application service layer Write a unit test module for the service.
3. Demo the service using a mock screen in the UI
4. Implement the service on a feature branch in the codebase.

#131 — Provide a Video Streaming Service Module

Description:

> As a developer, I would like access to the video stream coming off of the front and bottom camera of the drone, so that I could serve it to the user interface.

We've verified with our piloted flight CLI tool that we can "see through the eyes" of the drone via the front-facing and bottom-facing camera. We must now provide user interface with a service that returns that stream, so that the user interface can display it somehow.

Acceptance Criteria:

1. The stream must be delivered in real-time
2. Both the front-facing and bottom-facing cameras must be accessible

Related Tasks:

1. Document the service API methods.
2. Demo the service by providing a mock screen in the UI where the video stream can be displayed. Both the bottom and front facing view must be displayed. A trivial flight is not required to receive video from the drone hardware.
3. Write a unit test module for this service.
4. Implement the service in its own feature branch.
5. Create a class diagram, update package diagram for Android application service layer

#179 — Design A Segmentation Workflow/Pipeline for Longan Trees

Description:

Use any combination of filters and algorithms learned from the six texts in the research material collection to isolate longan fruits from image background objects.

Acceptance Criteria:

Demonstrate the efficacy of the process by comparing the output to some hand-crafted ground truth.

#191—Researching modeling Techniques/Technology related to agricultural and topological data

Description:

As a Designer, i would like to know the best field practices for scanning crops including camera equipment and color spectrum,sensors, 3D modeling, point clouds, SVM, drone flight orientation/ergonomics,etc… and they're feasibility/cost

Acceptance Criteria:

1. Will these techniques aid in the accuracy/consistency of our main task(count yield main goal)
2. Are they feasible(some equipment goes in the thousands for both HW & SW)
3. Difficulty of Understanding/Time Consumption

#197 — Setup Web Application Server Development Environment

Description:

As a backend web developer I would like to set up a web application development environment on my laptop to that I can conveniently develop the backend of the FruiTREC software system.

Acceptance Criteria:

1. Decide on a programming language, web application framework, database management software, web server, and an IDE
2. Install aforementioned software and related tools
3. Write installation guide for future developers

Related Tasks:

1. #195 — Write Environment Setup Guide
2. #194 — Configure web server dev environment
3. #193 — Research server-side web application development tools

#199 — Research Existing REST-like Web API Frameworks

Description:

As a backend web developer, I would like to compare and contrast the most popular Python-based REST-like web API frameworks, given that the computer vision module is already being written in Python.

Acceptance Criteria:

Either Django REST Framework or Graphene is chosen to be the framework for the web API server.

#210—Generate Graphs about Crops data

Description:

As a Developer, i would like to generate views the crop data in a convenient graphical(charts) way.Since the farmer is viewing information like yield, it may be worthwhile to create graphs based on this data for faster observation and comparison

Acceptance Criteria:

1. Graphs should be generated/regenerated real time when new data is entered
2. Compatible with mobile API
3. USer can choose graph preferences/style

#213 — Implement the Histogram of Oriented Gradients Detection Algorithm in Python

Description:

As a developer, I want to implement a texture-based object detection algorithm to count fruits that are similar in color to their surroundings.

Acceptance Criteria:

1. Algorithm executes and terminates.

#223 — Provide A Drone Control Service to the User Interface

Description:

As a developer, I would like to have the ability to send command through the drone via a module that abstracts the details of the network layer.

We need an Android module that exposes the drone's AT commands to the react-native user interface. We're going to need this in order to perform flight correction while the drone is flight and allow the user to perform emergency landings. We should abstract as many of the details of the networking layer as possible away from the developer consuming this service. For example, in cases where AT commands need be sent repeatedly before confirming their effects (such as AT*FTRIM) we should make those decisions behind the scenes, leaving the developer to only call an API

Acceptance Criteria:

1. Must expose following 5 AT command set of the AR Drone: AT*REF, AT*PCD_MAG, AT*_FTRIM, AT*CALIB, AT*CONFIG

Related Tasks:

1. Create a use case diagram
2. Create a feature document
3. Create a screen in the UI to prototype/test the module
4. Update package diagrams
5. Create a class diagram
6. Implement the service
7. Create a sequence diagram

#240 — Create Longan Detector using the HOG Algorithm

Description:

As an algorithm engineer, I want to create a longan fruit detector using the texture-based HOG method because the color-based method previously implemented was not effective.

Acceptance Criteria:

1. 80% support vector machine classification accuracy

Work Log:

1. Created negative and positive training samples for the SVM
2. Created pre-processing workflow to improve the effectiveness of the HOG feature detector

#241 — Write Computer Vision Application Development Tool Installation Guide

Description:

As an algorithm engineer, I want to document the installation process for all of the computer vision application tools that were selected to build my agricultural robotics platform.

#202 — Design and Implement Data Model for Drone Mission Data

Description:

As a backend web developer I would like to create a data model for all drone mission data (position, attitude, etc.) to serve as the basis of the Python classes that psycopg2 (an object-relational mapper) will map to entities, attributes, and relations in postgres.

#203 — Install REST-like Web API Framework

Description:

Install the chosen framework for facilitating the transfer of data between the client and server in the development environment (laptop VM)

Work Log:

We chose Graphene, the python implementation of the GraphQL single-endpoint query language

Pending User Stories

#163 — Design a simple website to view crop data

Description:

As a developer, I would like design a hub to view my crop data over the web once it has been processed. Since the tablet will be sending Video/Navigation data off site via the Web to a server, a web friendly application would help the user view useful information from the Mission i.e. there crops

Acceptance Criteria:

Website can communicate with Storage/Processing Machine ,Facilitates views typical of database, and Mobile Friendly.

#175 — Traditional Database

Description:

As a farmer, I would like to view the useful info produced from My Mission's and use this data for future planning. Since we are dealing in an agricultural field, it would benefit the user and help organize data for future planning if we store this info in a traditional database after the Video images have been processed for the "useful" information.

Acceptance Criteria:

1. Records are updated per mission after processing
2. "Bad Mission" record can be removed and updated
3. Can organize and set new records for different vegetation, land plots, comparison, etc..

#198 — Provide a Web API to the Mobile Client Application

Description:

As a grower I would like to design and build a REST-like API on the FruiTREC web server so that the FruiTREC client application can easily POST data to and GET data from the server.

Acceptance Criteria:

1. Serialization method for client requests and server responses defined
2. Data model for objects derived from use cases defined
3. RESTful or REST-like API frameworks researched and evaluated
4. RESTful or REST-like API framework chosen and installed
5. Reference guide for RESTful or REST-like API design and implementation written
6. Data model implemented in API framework

#201 — Design and Implement a Data Serialization/Deserialization Model

Description:

As a backend developer, I want to decide how to encode data for transmission over the internet, then decode at the network destination.

#218 — Create Avocado Detector using the HOG Algorithm

Description:

As an algorithm engineer, I want to use the Histogram of Gradients algorithm discovered during the research phase of Release 1 to create an avocado detector.

Work Log:

1. The detector has actually been created, but an image preprocessing pipeline must be written to improve the detector's accuracy.

#239 — Construct Training and Test Data Sets for the Neural Network-Based Detector

Description:

As an algorithm engineer, having produced substantive results using color and texture-based fruit detection, I want to further explore the YOLO neural network framework and compare its performance to the HOG algorithm.

Additional Comments:

Both are machine learning techniques and require large data sets, but the neural network classifier requires *much* larger set to be capable of identifying features in images on its own (as opposed to the support vector machine classifier, which has the features that it looks for pre-computed by the HOG feature descriptor.

Project Plan

This section describes the planning that went into the realization of this project. This project incorporated the agile development techniques and as such required the sprints to be planned. These sprint plannings are detailed in the section. This section also describes the components, both software and hardware, chosen for this project.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

Project Management and Software Development Tools

Google Drive
A file storage and synchronization service created by Google.

Mingle
Mingle was used as a planning and management tool for the various agile development processes

Git
Repository hosting service,provides access control and several collaboration features

PyCharm
A cross-platform IDE for Python development (free for student use).

System Hardware and Software Components

Parrot AR Drone
The Drone the team used throughout the project, special thanks to Discovery lab for providing.

Android
A popular operating system for mobile phones and tablet computers.

Python
Widely used high-level, general-purpose, interpreted, dynamic programming language.

PostgreSQL
A database query language and database server.

Ubuntu
A computer operating system based on the Debian Linux distribution and distributed as free and open source software, using its own desktop environment.

Apache
A public-domain open source Web server .

GraphQL
A query language that provides a common interface between the client and the server for data fetching and updating.
Software Libraries

OpenCV
A computer vision algorithm library implemented in C++ with Python bindings.

Django
A Python web development framework that encourages rapid development and clean, pragmatic design.

Graphene
A Python implementation GraphQL.

Scikit-learn
A Python machine learning library.

React Native
It lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components.

Sprints Plan

Sprint 1

Attendees: Wagner Vendrame, Leonardo Bobadilla, Miguel Chateloin, Franklin Abodo, Gabriel Barrs
Start time: 2016.9.9, 11:40 AM
End time: 2016.9.9, 12:00PM

After discussion, the velocity of the team were estimated to be 60 points per sprint.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- User Story 120 - Demonstrate All Drone Application Development Tools
- User Story #117 Research Drone Hardware Platforms (Off-the-shelf vs Custom)
- User Story #118 Research Drone Software Application Development Tools
- User Story #119 Install Drone Application Development Tools
- User Story #123 Research Computer Vision Application Development Tools
- User Story #124 Install Computer Vision Application Development Tools
- User Story #127 Setup Android development environment
- User Story #128 Test Parrot FreeFlight application with all currently owned AR Drones
- User Story #125 - Demonstrate All Computer Vision Application Development Tools
- User Story #126 - Collect and Store Computer Vision Literature and other References

The team members indicated their willingness to work on the following user stories.

- Franklin

    - User Story 125 - Demonstrate All Computer Vision Application Development Tools
    - User Story 126 - Collect and Store Computer Vision Literature and other References
    - User Story #123 Research Computer Vision Application Development Tools
    - User Story #124 Install Computer Vision Application Development Tools
    - User Story #117 Research Drone Hardware Platforms (Off-the-shelf vs Custom)
    - User Story #118 Research Drone Software Application Development Tools
- Gabriel
    - User Story 126 - Collect and Store Computer Vision Literature and other References
    - User Story #117 Research Drone Hardware Platforms (Off-the-shelf vs Custom)
    - User Story #118 Research Drone Software Application Development Tools
    - User Story #119 Install Drone Application Development Tools
- Miguel
    - User Story 120 - Demonstrate All Drone Application Development Tools
    - User Story #127 Setup Android development environment
    - User Story #128 Test Parrot FreeFlight application with all currently owned AR Drones

Sprint 2

Attendees: Wagner Vendrame, Leonardo Bobadilla, Franklin Abodo, Gabriel Barrs, Miguel Chateloin
Start time: 2016.9.23, 3:10PM
End time: 2016.9.23, 3:30PM

After discussion, the velocity of the team were estimated to be <41>.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- User Story 121 - Research Computer Vision Theory, Techniques, and Applications
- User Story 184 - Design and Implement Image Segmentation Pipelines for All Crops of Interest
- User Story 154 - Create and Launch an Autonomous Flight
- User Story 142 - Provide a Drone Control Service to higher-level navigation modules that need to control the drone
- User Story 131 - Provide a Video Streaming Service to the User Interface

- User Story 130 - Provide Navdata Service to the User Interface
- USer Story 191 - Researching Data modeling Techniques/Tech

The team members indicated their willingness to work on the following user stories.

- Franklin
    - User Story 184 - Design and Implement Image Segmentation Pipelines for All Crops of Interest
    - User Story #241 -Write Computer Vision Application Development Tool Installation Guide
- Gabriel
    - User Story 121 - Research Computer Vision Theory, Techniques, and Applications
    - User Story 191- Researching Data modeling Techniques/Tech
- Miguel
    - User Story 154 - Create and Launch an Autonomous Flight
    - User Story 131 - Provide a Video Streaming Service to the User Interface
    - User Story 130 - Provide Navdata Service to the User Interface
    - User Story 142 - Provide a Drone Control Service to higher-level navigation modules that need to control the drone

Sprint 3

Attendees: Leonardo Bobadilla, Franklin Abodo, Gabriel Barrs, Miguel Chateloin, Virgil, Nick
Start time: 10:40AM
End time: 11:00AM

After discussion, the velocity of the team were estimated to be 116.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority:

- User Story #121- Research Computer Vision Theory, Techniques, and Applications
- User Story #184 - Design and Implement Image Segmentation Pipelines for All Crops of Interest
- User Story #130 - Provide Navdata Service to the User Interface
- User Story #197 - Setup Web Application Server Development Environment
- User Story #175 - Design Traditional Database
- User Story # 163 - Design website to view Crop data

The team members indicated their willingness to work on the following user stories.

- Franklin
    - User Story #121- Research Computer Vision Theory, Techniques, and Applications
    - User Story #184 - Design and Implement Image Segmentation Pipelines for All Crops of Interest
    - User Story #197 - Setup Web Application Server Development Environment
- Gabriel
    - User Story #163 - Design website to view Crop data
    - User Story #175 - Design Traditional Database
- Miguel
    - User Story #130 - Provide Navdata Service to the User Interface

Sprint 4

Attendees: Leonardo Bobadilla, Franklin Abodo, Gabriel Barrs, Miguel Chateloin
Start time: 2016.10.27, 2:00PM
End time: 2016.10.27, 2:10PM

After discussion, the velocity of the team were estimated to be <20>.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- User Story #202 - Design and Implement Data Model for Drone Mission Data
- Uster Story #173 -Establish VM Server Communication
- User Story #199 - Research Existing REST-like Web API Frameworks
- User Story #203 - Install REST-like Web API Framework
- USer Story #142 - Create a Flight Plan
- User Story #204 - Manage Flight Plans
- User Story #210 - Generate graphs of crops

The team members indicated their willingness to work on the following user stories:

- Gabriel:
    - User Story #173-Establish VM Server Communication
    - User Story #210- Generate Graphs of Crops
- Miguel:
    - User Story #142 - Create a Flight Plan
    - USer Story #204 - Manage Flight Plans
- Franklin Abodo :

- User Story 199- Research Existing REST-like Web API Frameworks
- User Story 202 - Design and Implement Data Model for Drone Mission Data
- User Story 203 - Install REST-like Web API Framework

Sprint 5

Start time: 12:15 PM
End time: 12:20 PM

After discussion, the velocity of the team were estimated to be <27>.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- User Story #179 - Design A Segmentation Workflow/Pipeline for the Longan Fruit
- User Story #218 - Create Avocado Detector using the HoG Algorithm
- User Story #213 - Implement the Histogram of Gradients Object Detection Algorithm
- User Story 239 - Construct Training and Test Data Sets for the Neural Network
- User Story #223 -Provide A Drone Control Service to the User Interface
- User Story #211-Continue integrating/consolidating website and schema features into one solid platform

The team members indicated their willingness to work on the following user stories.

- Franklin Abodo
    - User Story #179 - Design A Segmentation Workflow/Pipeline for the Longan Fruit
    - User Story #218 - Create Avocado Detector using the HoG Algorithm
    - User Story #213 - Implement the Histogram of Gradients Object Detection Algorithm in Python
    - User Story #239 - Construct Training and Test Data Sets for the Neural Network *Wagner know that i'll be going back to computer vision and reviving some unfinished work*
- Gabriel:

    - User Story #211-Continue integrating/consolidating website and schema features into one solid platform
- Miguel:
    - User Story #223 -Provide A Drone Control Service to the User Interface

Sprint 6

Start time: 12:15 PM
End time: 12:20 PM

After discussion, the velocity of the team were estimated to be <34>.

The product owner chose the following user stories to be done during the next sprint. They are ordered based on their priority.

- User Story #237 - Unify CV Algorithms in a Single Python Module
- User Story #155 - Persist Autonomous Flight Data to Device
- User Story #240 - Create Longan Detector using the HoG Algorithm
- User Story #154 - Launch an Autonomous Flight from Existing Plan
- User Story #131- Provide a Video Streaming Service to the User Interface
- User Story #238 - Port Final-State Project to Lab Workstation

The team members indicated their willingness to work on the following user stories:

- Franklin:
    - User Story #237 - Unify CV Algorithms in a Single Python Module
    - User Story #238 - Port Final-State Project to Lab Workstation
    - User Story #240 - Create Longan Detector using the HoG Algorithm
- Gabriel:
    - User Story #211-Continue integrating/consolidating website and schema features into one solid platform
- Miguel:
    - User Story #131- Provide a Video Streaming Service to the User Interface
    - User Story #154 - Launch an Autonomous Flight from Existing Plan
    - User Story #155 - Persist Autonomous Flight Data to Device

System Design

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a subsystem component diagram, and the subsystem decomposition is explained. Finally, the design patterns used in the project are discussed.

Overview

FruiTREC uses a client-server architecture. The presentation layer is implemented entirely on the client device. The application layer is distributed between the client, which is primarily responsible for the logic associated with operating UAVs, and server (which is responsible for the data analysis and report generation components of FruiTREC). The presentation layer is implemented using an MVC architecture.

Subsystem Decomposition

The client subsystem is designed to allow concurrent development of mobile applications on two different platforms: Android and iOS. This is more of future-proofing measure as only the Android module was developed during this release. There is a logical separation between the presentation layer and the business layer. The business layer handles the responsibilities of communicating with and commanding the drone, as well as sending a receiving data from the servers. It sits between the presentation and the network layer. The NavData Service Module, Control Service Module, VideoStream Service Module compose the drone-related module of the application.

The server subsystem will be composed of two modules: 1) the web application server and 2) the image analysis server. The web application (Python-Django) will provide the Web Application Programming Interface interface that the client will use to access 1) the database and 2) the image analysis module. This web API will be implemented using an HTTP server (Apache), and a data query language (GraphQL). The data query language will be used to construct requests on the client, interpret requests on the server, construct responses on the server, and then interpret responses on the client. The requests and responses will be transmitted over the internet using HTTP. The responses will contain only data. All UI views will be constructed on the client side given the data. The database is implemented using a PostgreSQL server, with which the web application interfaces using the Django and psycopg frameworks. There frameworks provide

automated mapping between the object models implemented in Python, and the database entities and the queries on them that are written in the PostgreSQL query language.

Hardware and Software Mapping

The server subsystem is distributed across two computers. The web application will be implemented on a virtual machine running the Ubuntu 14.04 operating system, which itself runs on a Dell server featuring an Intel Xeon CPU. The image analysis module is implemented on a Dell Precision T1700 desktop featuring an NVIDIA K600 GPU and Intel Core i7 CPU. This desktop also runs the Ubuntu 14.04 operating system.

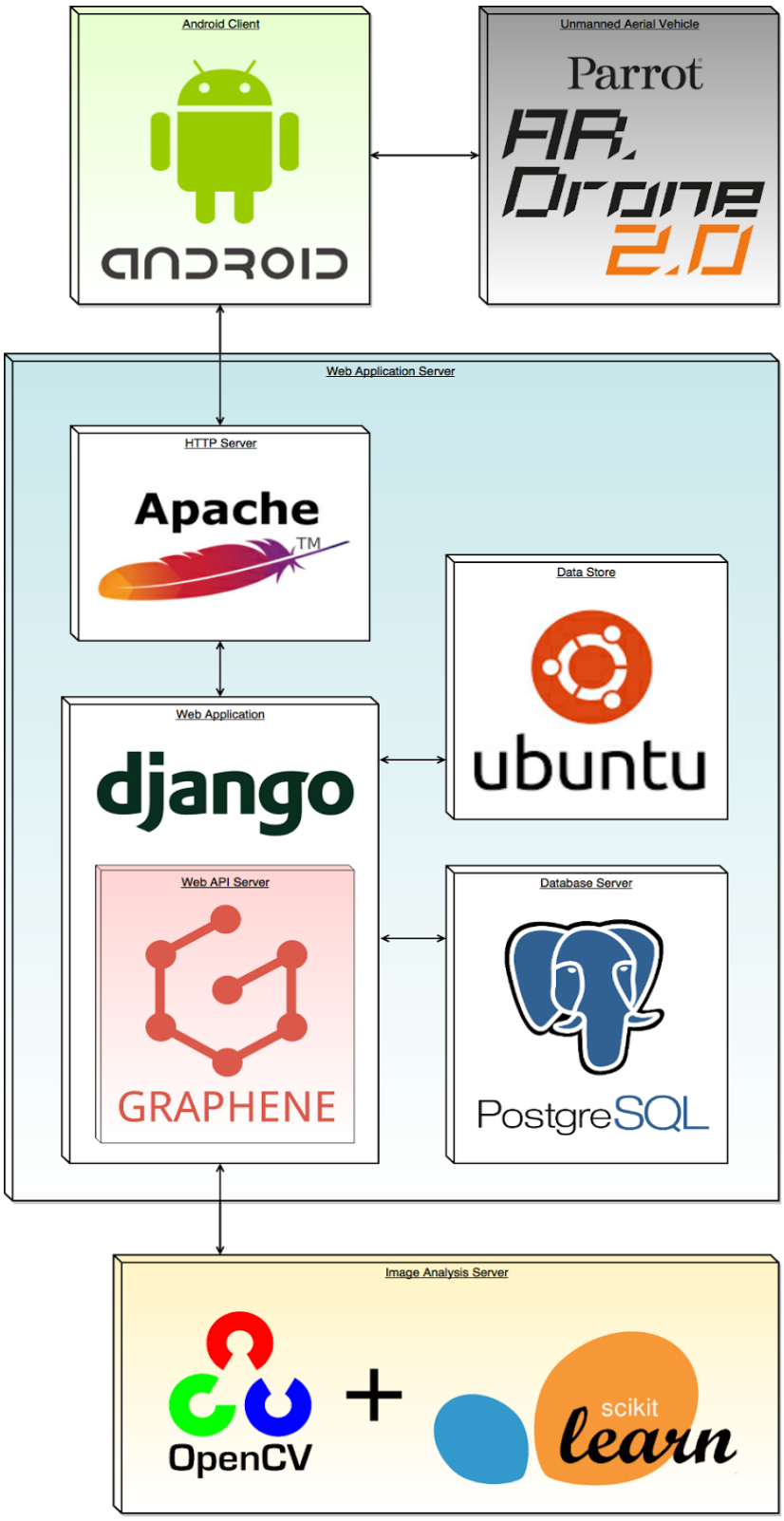


Figure 1. Server Subsystem Component Diagram

Persistent Data Management
The server subsystem's data layer is distributed across the web application server and image analysis server. The system will store two different kinds of data: user profile data and the FruiTREC mission information in a relational database system powered by PostgreSQL

Architectural Patterns

1. **System archtectural pattern:** Client-server

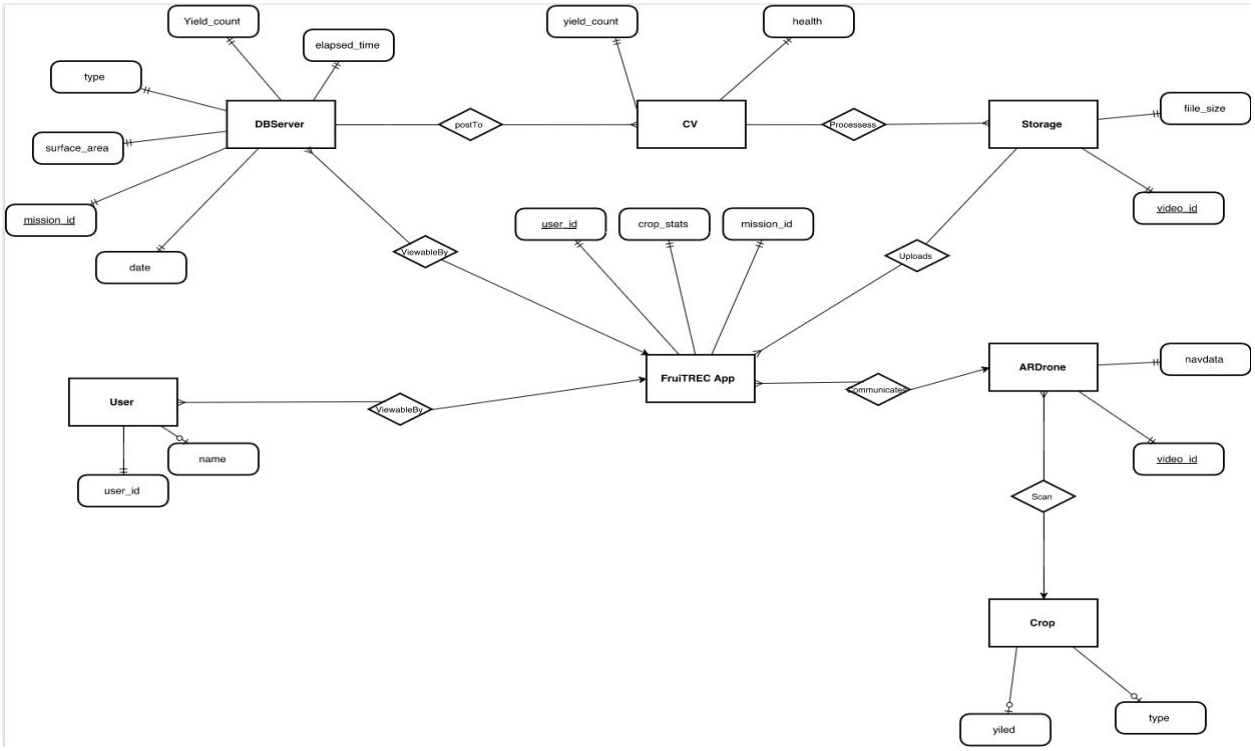
2. **Mobile device architectural pattern: MVC**

- · The MVC architectural pattern allows us to keep our user interface and data access code separate by using a "Controller" object to manipulate the data code stored in a "Model" object, which updates the "View" object that stores the user interface code.
- · Because our software is intended to serve as an extension to various drones models in the future for user convenience, the MVC architectural pattern allows for reusability, meaning that data that resides in our model can be reused for other enthusiasts who wish to implement crop analysis through drones.

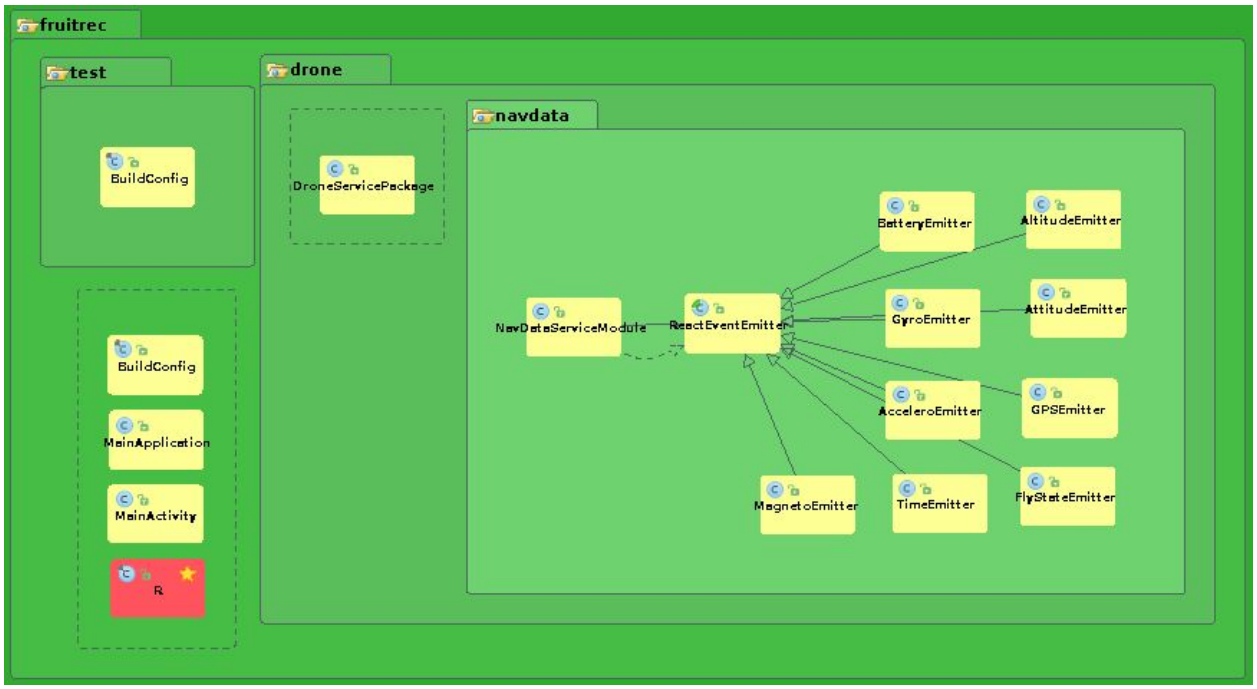
3. **Second architectural pattern: Repository**

- · One benefit of using the repository architectural pattern is that we don't have to think about the technical details as to how we will store our data, since the database is treated as only a detail of the entire application, and hence can be designed at the start of a project.
- The repository architectural pattern allows us to create a one way dependency between domain and data mapping layers since the domain layer does not have to care how the data of our system is stored and only needs to satisfy the requests received.

System and Subsystem Decomposition



Deployment Diagram



Design Patterns

- Singleton : we used this design patterns mostly for convenience and security, such that there is only one instance of each database entity and other functions require just to use and fetch this only entity. The benefits of doing so are to not create redundancy and confusion, it also enable us to create consistency throughout the code so that each object is inserted to his designated collection instance. Moreover, using this design pattern provide with great security and usability as each function and class can look on the collections as one interface and instance.
- Command : we used this design patterns to create common interface and convenience for commands that are frequently used. Such as set menu and set view, that effect the overall functionality and therefore crucial for making independent interface for the commands. The implementation of set view and menu follows this pattern so each function and class call this common interface/class to set the functionality of the command (from common function).
- Abstract Factory : this design patterns is perhaps the most useful and helpful in our project, since it's enabled us to implement more easily and faster the chat for each representative and map region chat. it's enabled us to create entities more easily and consistent in its design/functionality and implementation. In our project (FruiTREC) the abstract factory is designated for creating the mission designation for the drone during the mapping of a location. So the flight and videocapture functionality and characteristics are the same just the context of the mission is different i.e. distance,time, pitch,height, etc.

System Validation

7.1.1. Test Suite USID01
Purpose: To test the functionality of use case**: USID01 - Register New Account**

Test case 1:

Purpose: To test that the system registers a new user correctly

Precondition: 1. The user is at the website homepage.
2. The server is active.
3. The user is not logged in.
4. A user with username "test1234" does not exist in the database.

Input: The user clicks either the "Register" tab or the "Click Here" to register link on the homepage and on the following page enters the following information in the designated fields:
Username: test1234
Email address: test@test1234.com
Password: 1234

After the user has entered the preceding information by typing them on using a keyboard, he/she clicks on the "Submit" button.

Expected Output: The user is logged in and redirected to the Home page which displays the list of products, and the user test1234 is added to the database with the entered information.

Test case 2:

Purpose: To test that the system does not create a user if one with the same Username exists

Precondition: 1. The user is at the website homepage.
2. The server is active.
3. The user is not logged in.
4. A user with username “test1234” does exist in the database.

Input: The user clicks either the “Register” tab or the “Click Here” to register link on the homepage and on the following page enters the following information in the designated fields:

Username: test1234
Email address: test@test1234.com
Password: 1234

After the user has entered the preceding information he clicks on the “Submit” button.

Expected Output: The system will present the user with a message that states “Username already exists” and clear all the fields of the register page.

Test case 3:

Purpose: To test that the system does not register the user if a field is missing

Precondition: 1. The user is at the website homepage.
2. The server is active.
3. The user is not logged in.
4. A user with username “test1234” does not exist in the database.

Input: The user clicks either the “Register” tab or the “Click Here” to register link on the homepage and on the following page enters the following information in the designated fields:

Username: test1234
Email address: test@test1234.com
Password:

After the user has entered the preceding information he clicks on the “Submit” button.

The user leaves the “Password” field blank, and clicks the “Submit” button.

Expected Output: An error note appears above the field stating “This field is required.”

7.1.2. TEST SUITE TUSID02

Purpose: To test the functionality of use case**: USID02 - Login account**

Test case 1:

Purpose: To test that the website performs a login when a correct username and password are input

Precondition: 1. The server must be running and connected
2. The Login Page is active
3. The User has registered an account with :
Username:test
password:1234

Input: The user presses the “Submit“ button with:
Username:test
password:1234

Expected Output: Login is performed, view is changed to the product list page .

Test case 2:

Purpose: To test that the website doesn’t perform a login when a username and password are not input.

Precondition: 1. The server must be running and connected

2. The Login Page is active

Input: The user presses the “Submit“ button with:
Username:
password:

Expected Output: Login is not performed, and an “Invalid Login” message should appear.

Test case 3:

Purpose: To test that the website does not perform a login when an incorrect username or password are input.

Precondition: 1. The server must be running and connected
2. The Login Page is active
3. The database should NOT contain an account with: :
Username:badtest
password:4321

Input: User attempting to login presses the “Submit” button with:
username: badtest
password: 4321

Expected Output: Login is not performed, and an “Invalid Login” message should appear.

7.1.3. TEST SUITE TUSID04

Purpose: To test the functionality of use case**: USID04 - Logout**

Test case 1:

Purpose:	To test "logout" when a user is logged in to the system.
Precondition:	1. The django server must be running and connected to its communication port. 2. The user needs to have an active account. 3. The tester user needs to be logged in with the Name "test" and Password "1234". 4. The Logout button is active.
Input:	The tester user presses the "Logout" button right after logging in to the system.
Expected Output:	1. The page is subsequently reloaded to the login page. 2. The active user session is terminated.

Test case 2:

Purpose: To test if pressing the "Back" button on the browser logs the user back once the user logged out.

Precondition:	1. The server must be running and connected to its communication port. 2. The user needs to have an active account. 3. The user has logged out. 4. The "Back" button on the browser is visible..
Input:	The user presses the "Back" button the browser once logged out..
Expected Output:	The user shouldn't be signed in and instead be redirected to the Login page.

Glossary

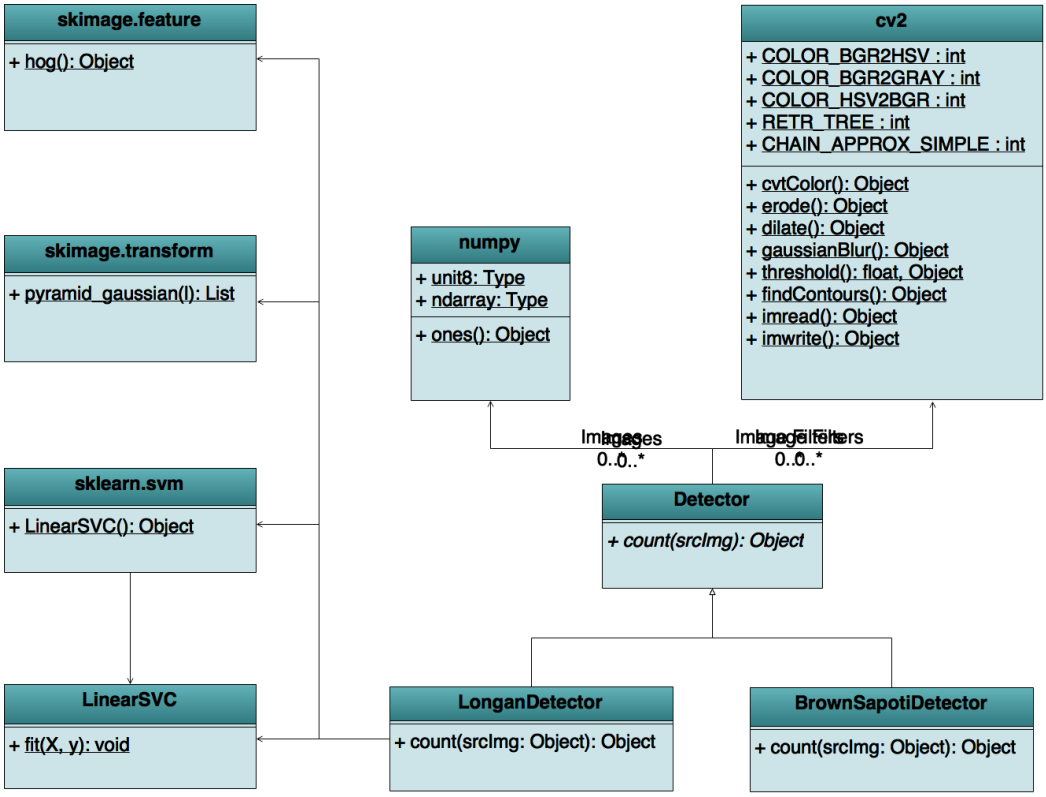
AR Drone: The quadracopter drone used for this project.

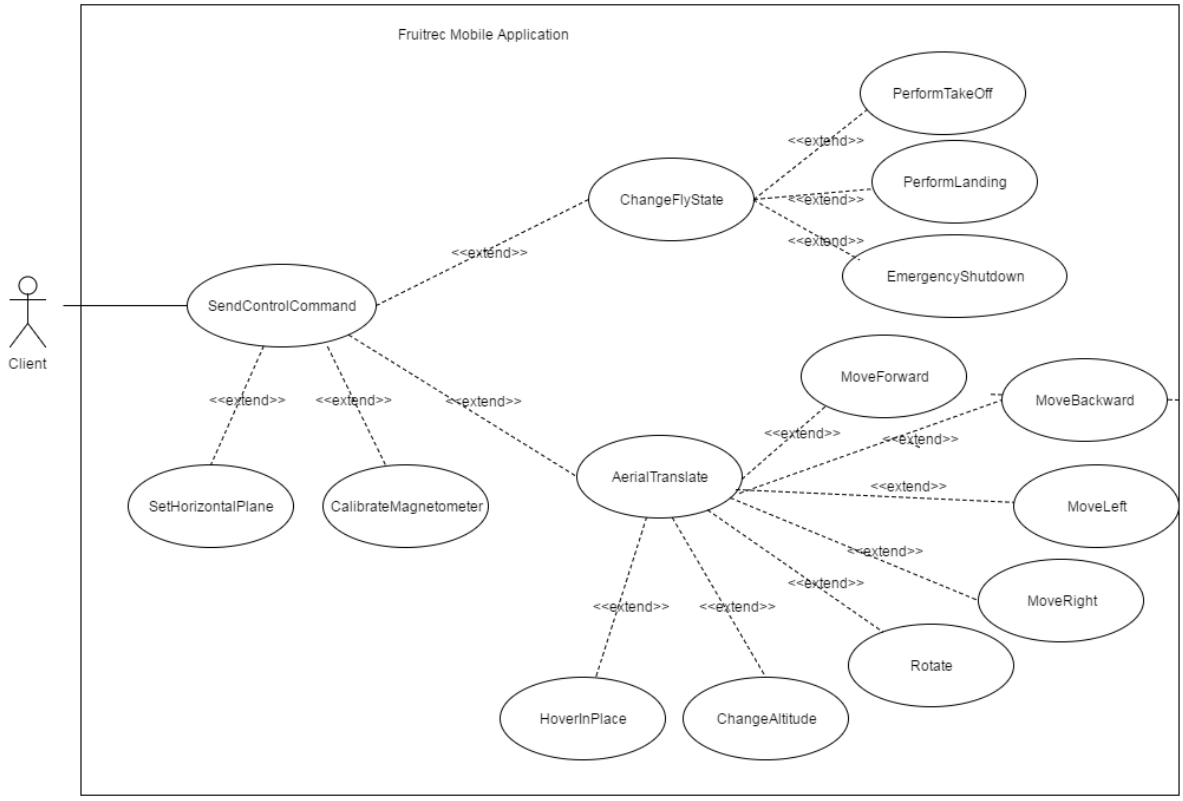
AT Command: Parrot's proprietary format

Quadracopter: A drone with 4 blades. Hexacopters and Octocopters had 6 and 8 blades respectively. The number of blades usually correlates to the stability of the drone while in flight.

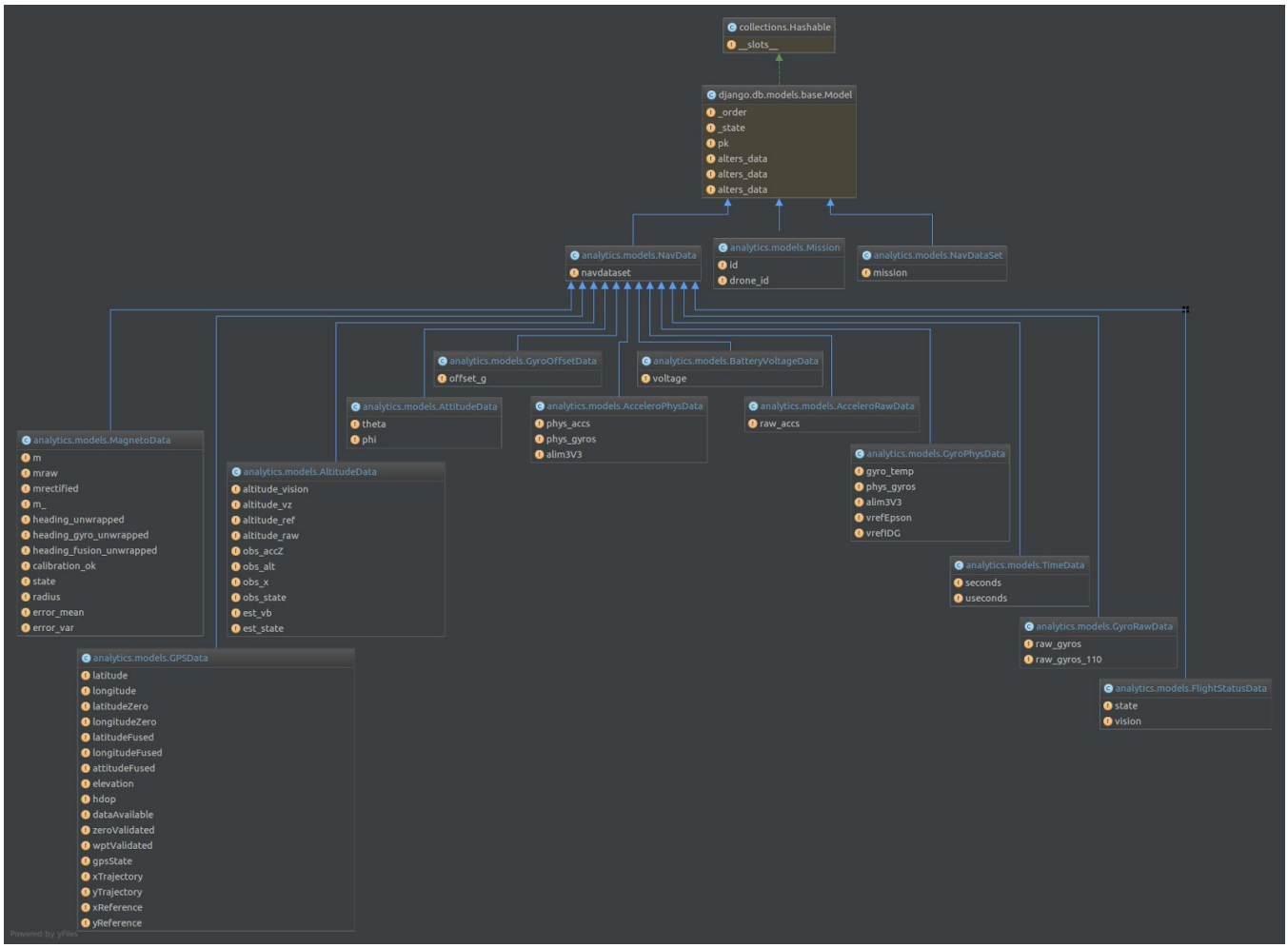
Appendix
Appendix A - UML Diagrams

Image Analysis Subsystem

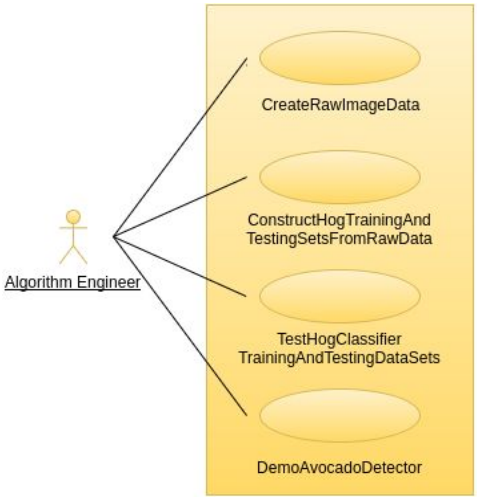




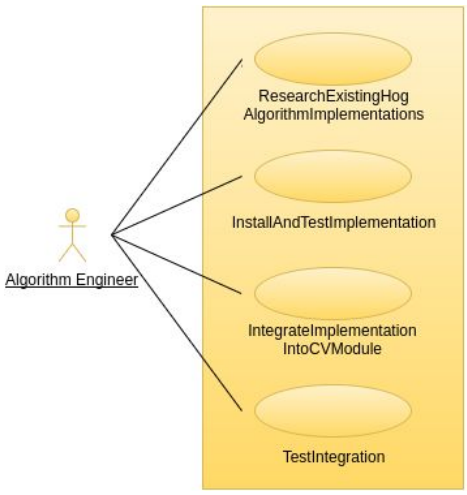
#223 Use Case Diagram



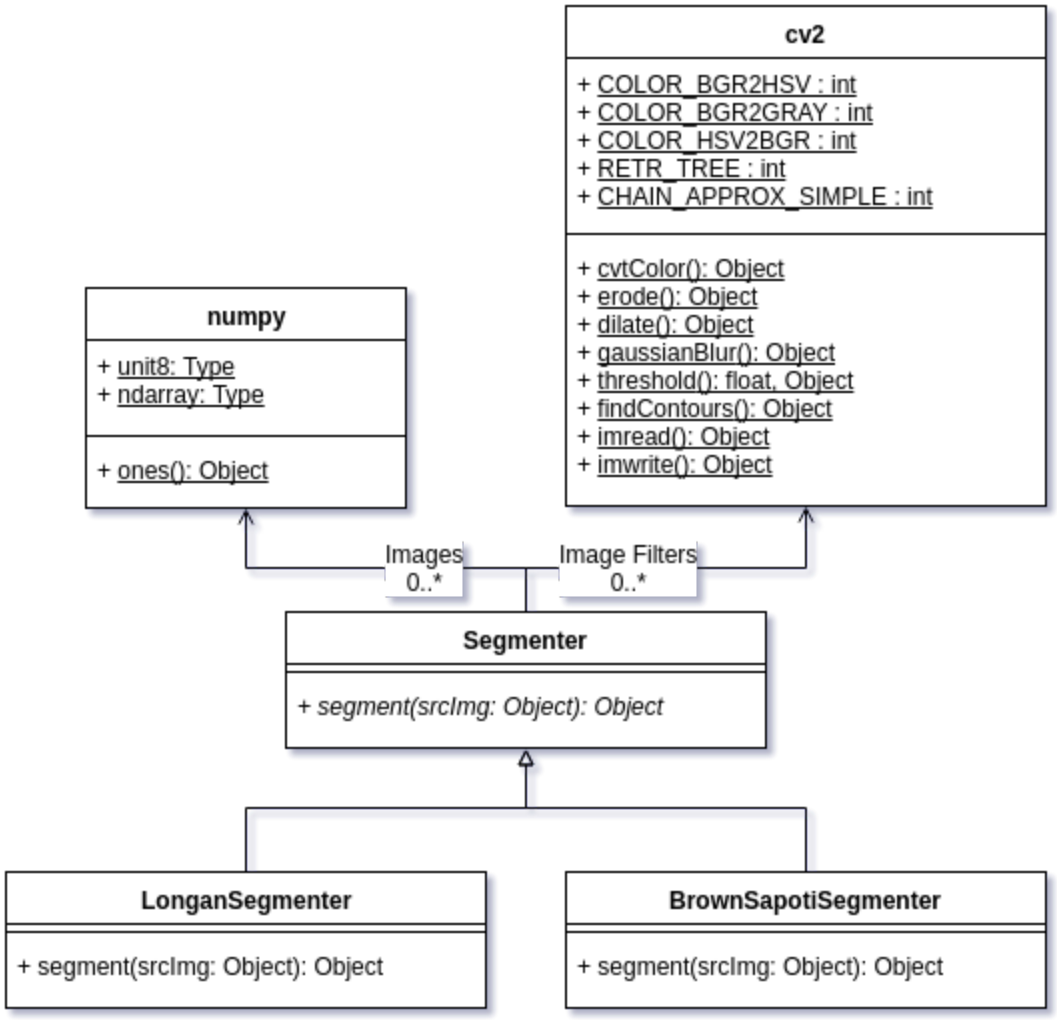
Models.py Class Diagram for User Story #198



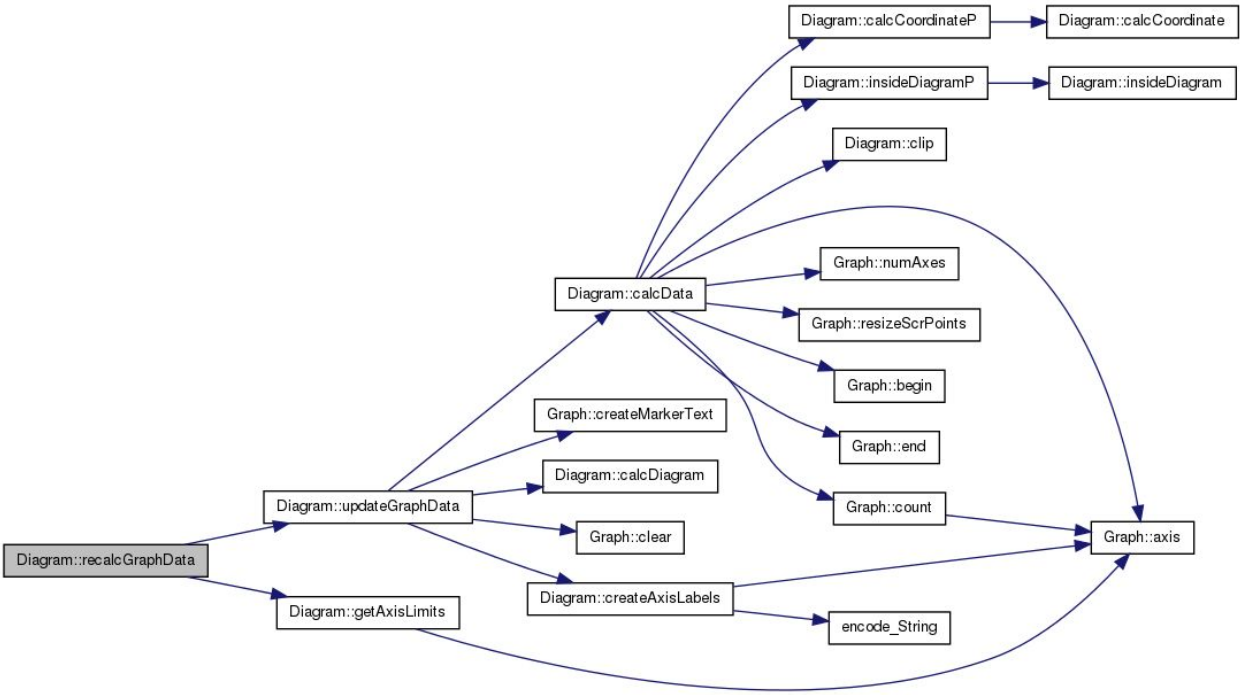
User Story #218 Use Case Diagram



USer Story #213 Use Case Diagram

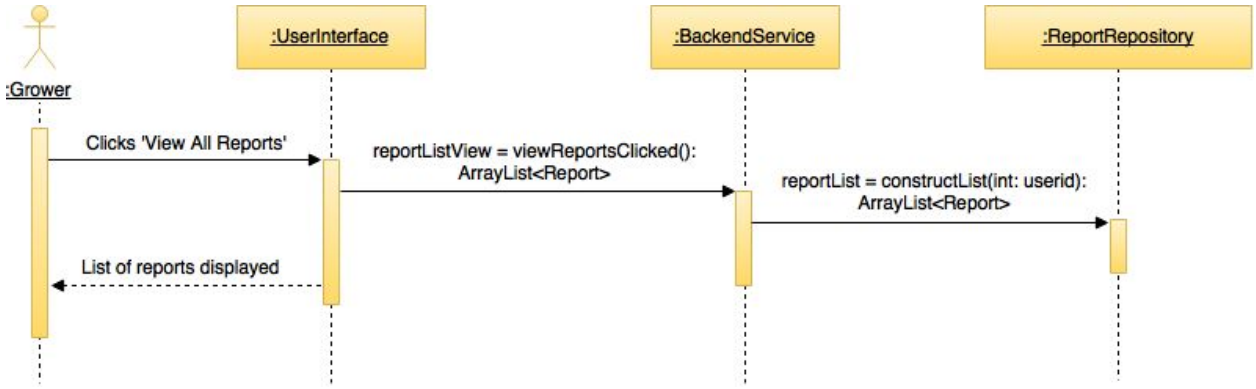


User Story #179 Class Diagram

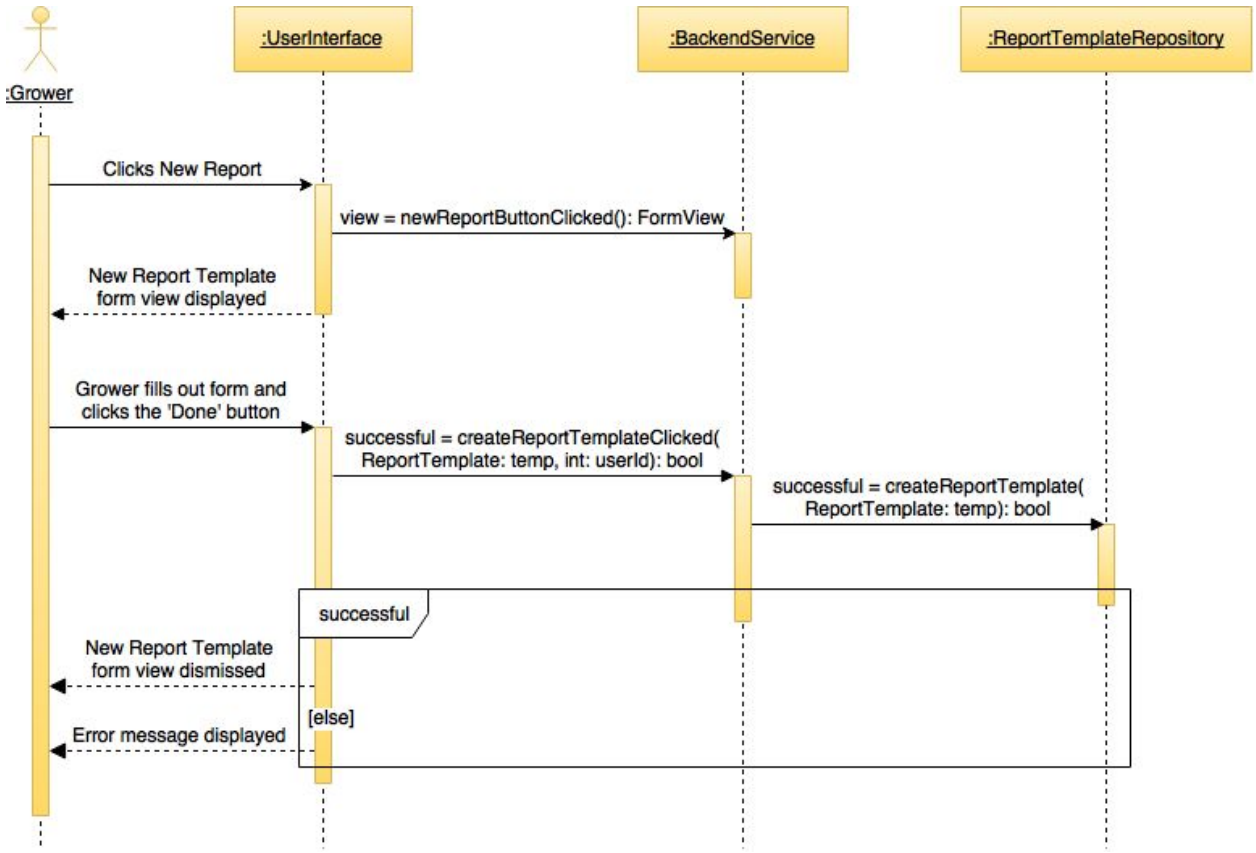


User Story #210 Minimal Class Diagram

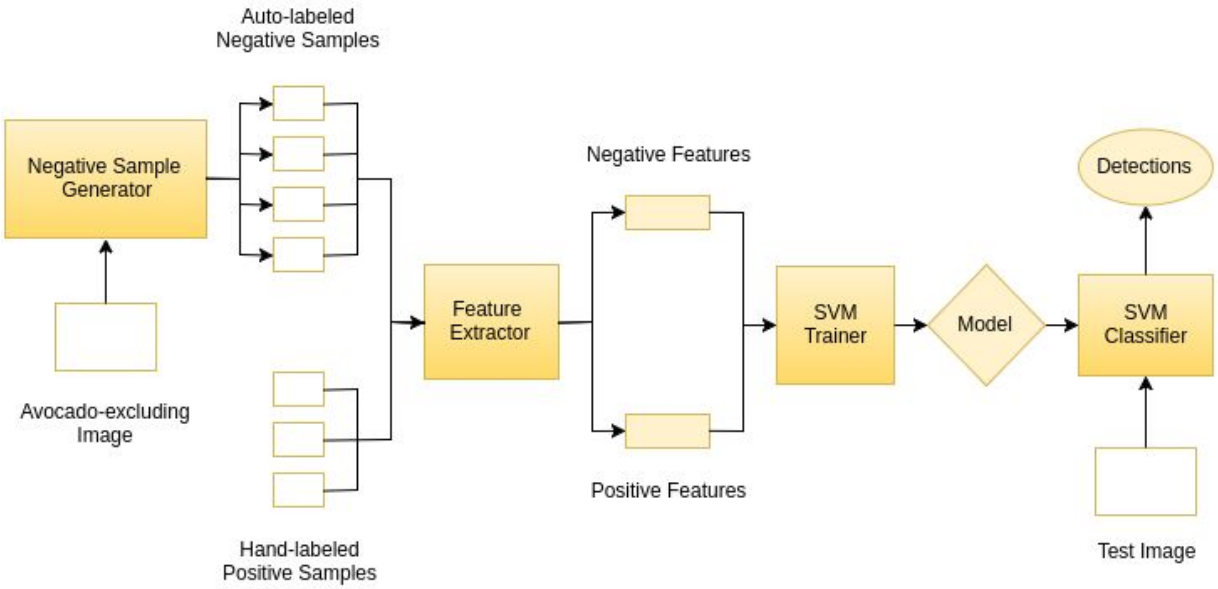
Dynamic(sequence and flow chart i.e step specific action jot general overview):



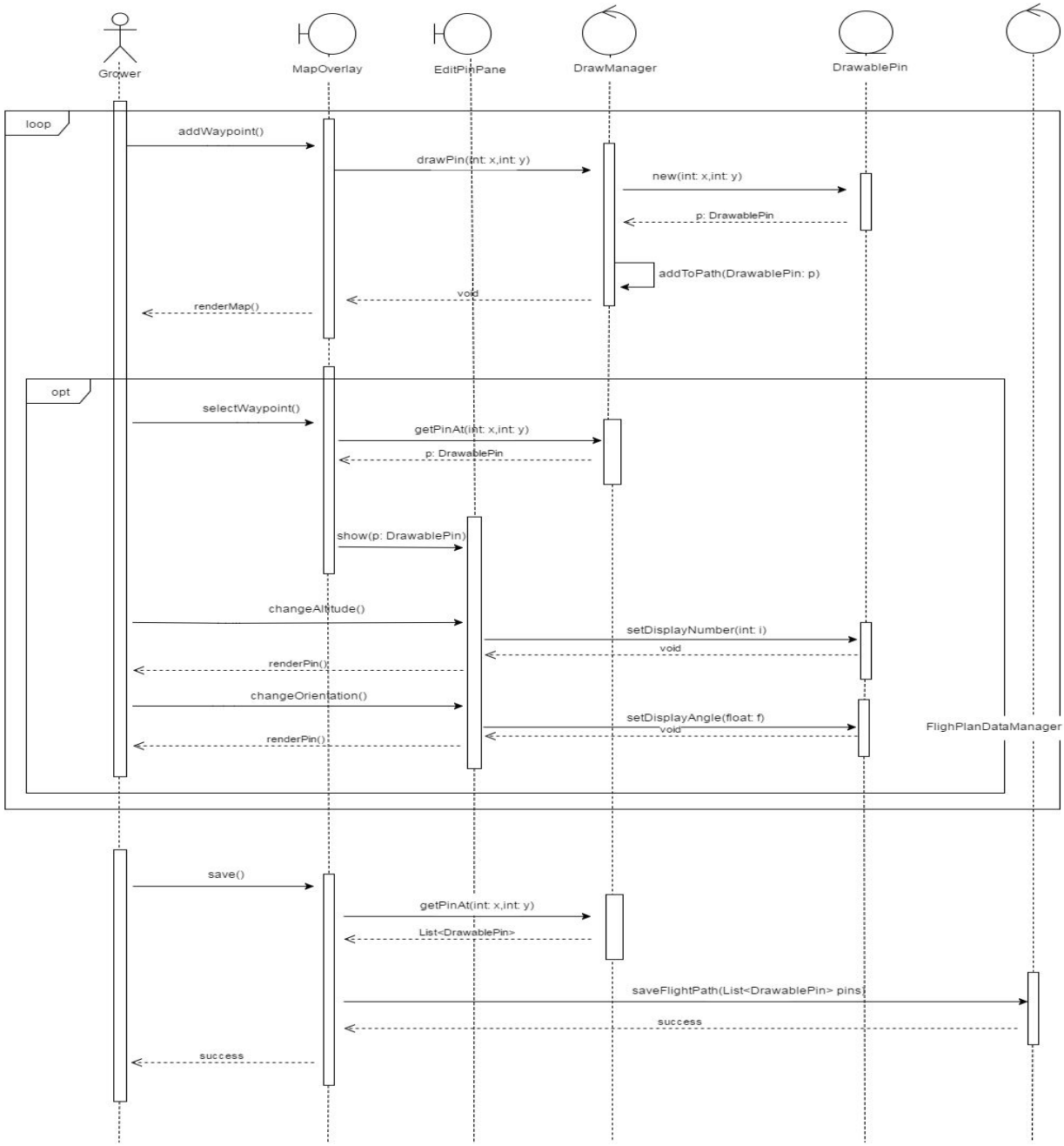
Sequence Diagram(1 of 2) of User Story #198



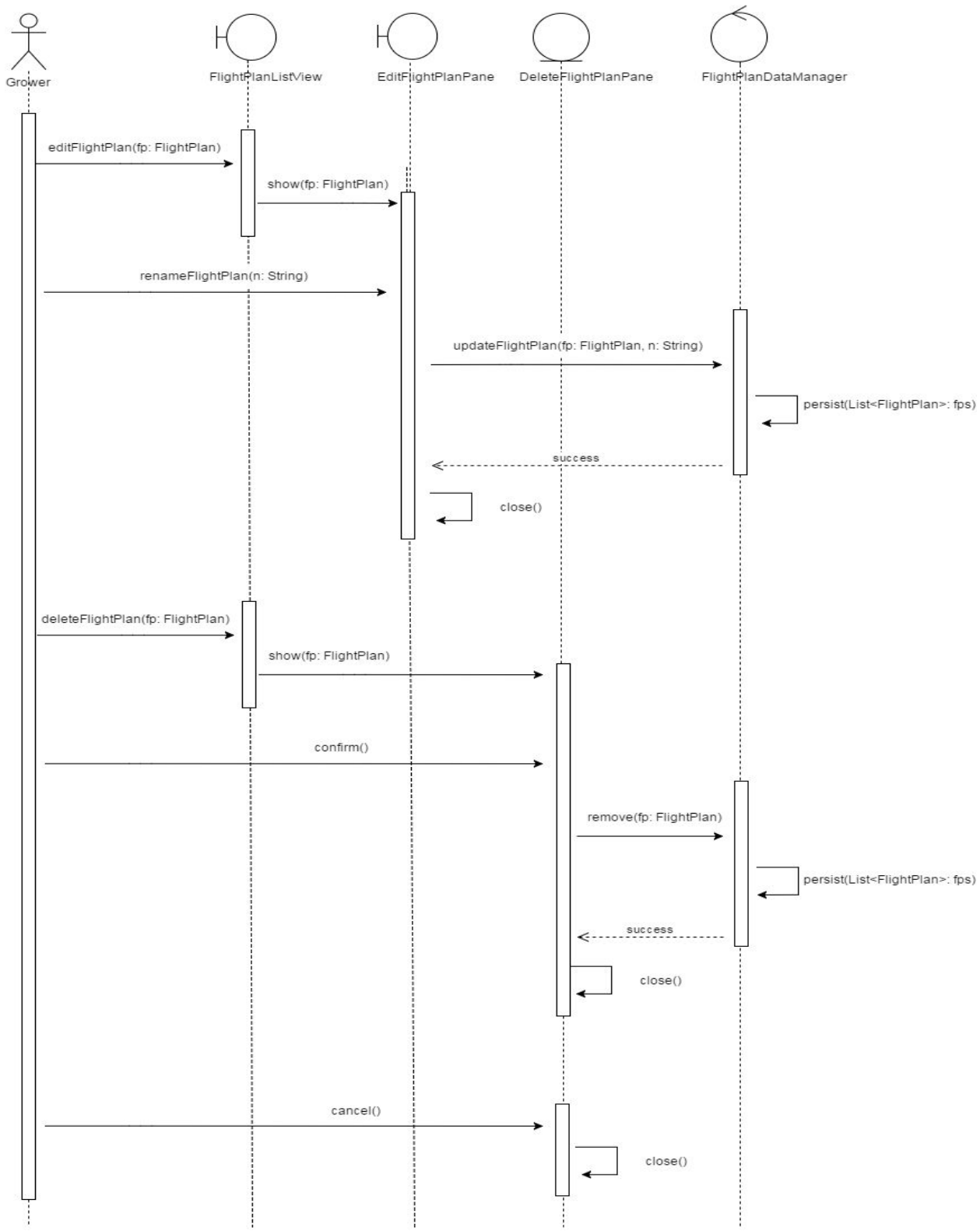
Sequence Diagram(2 of 2) of User Story #198



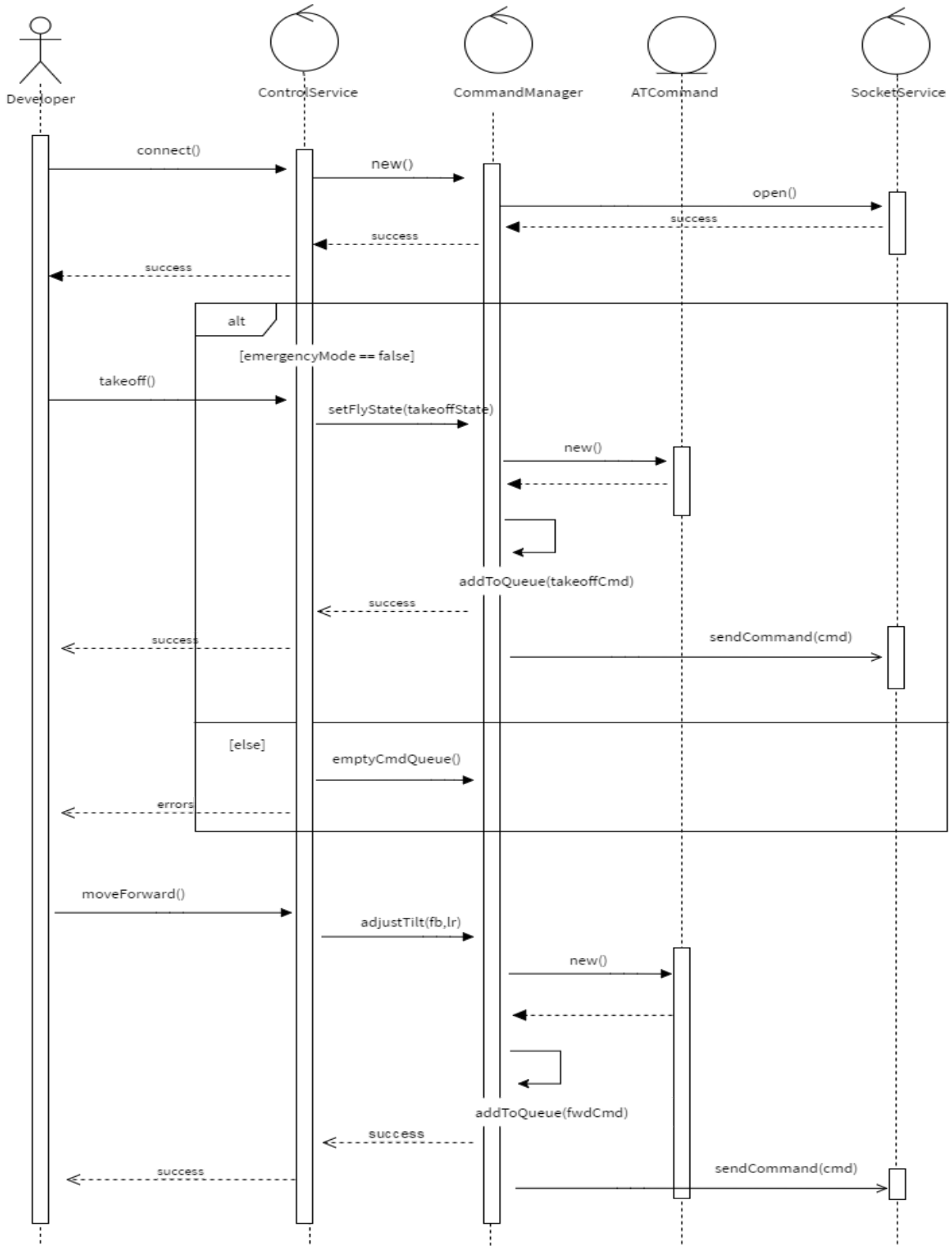
User Story #218 Work Flow Chart



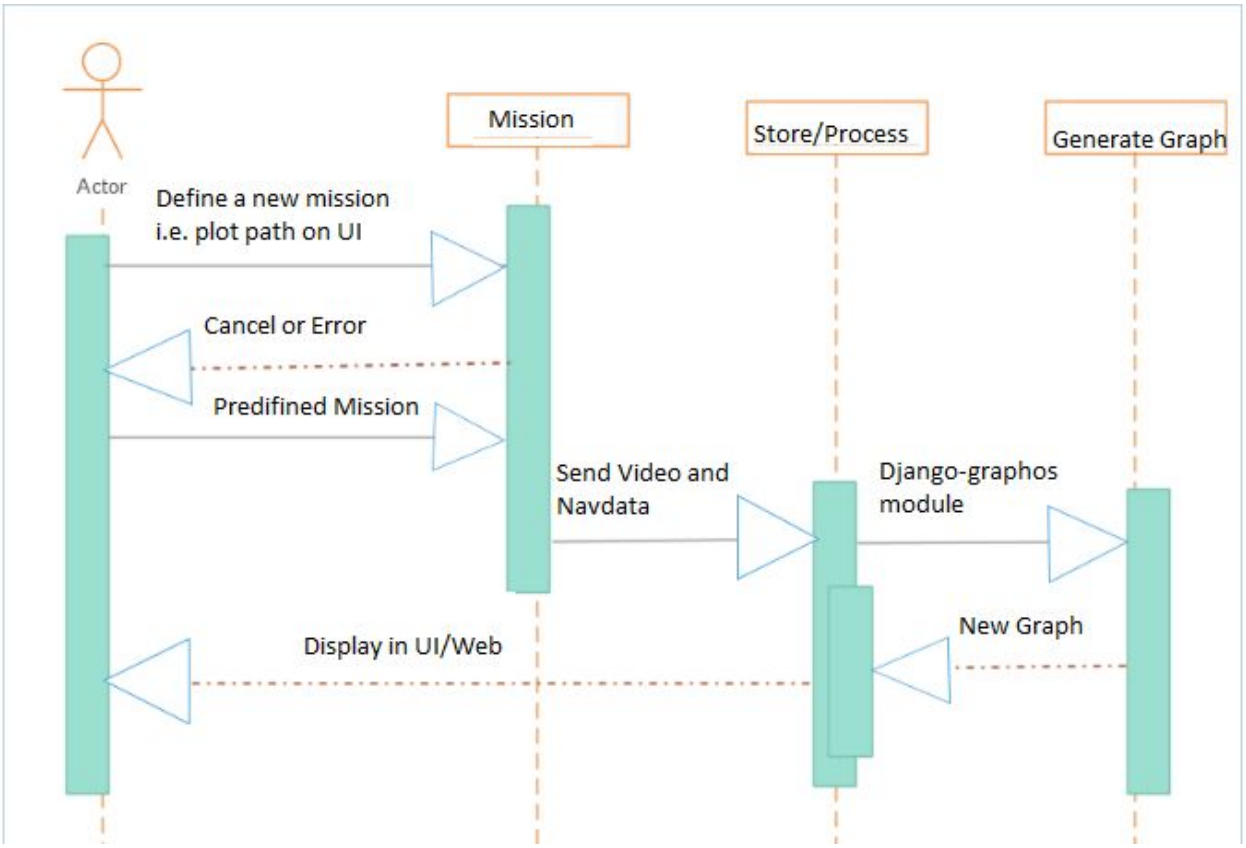
#142 Sequence Diagram



#204 Sequence Diagram

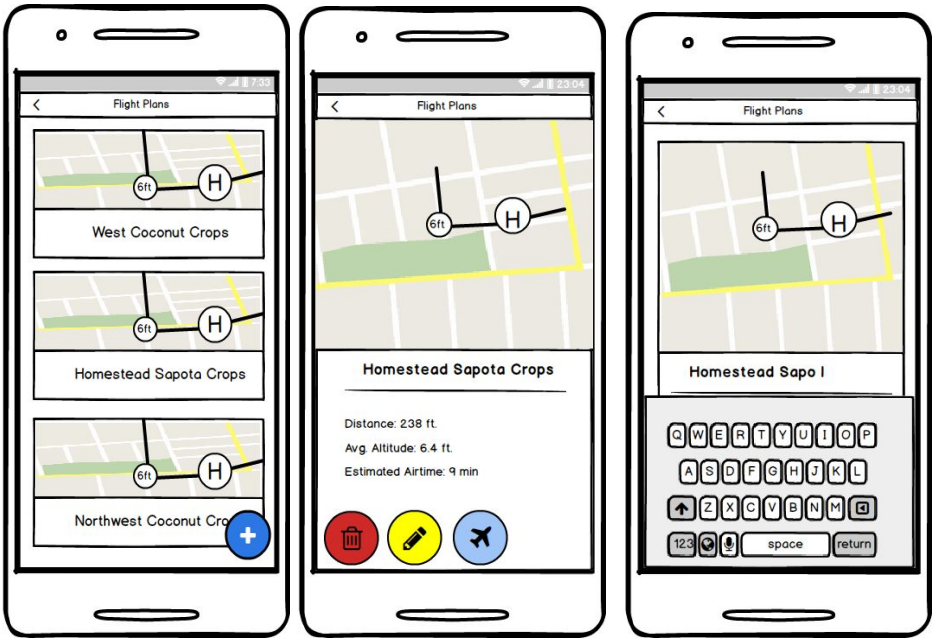


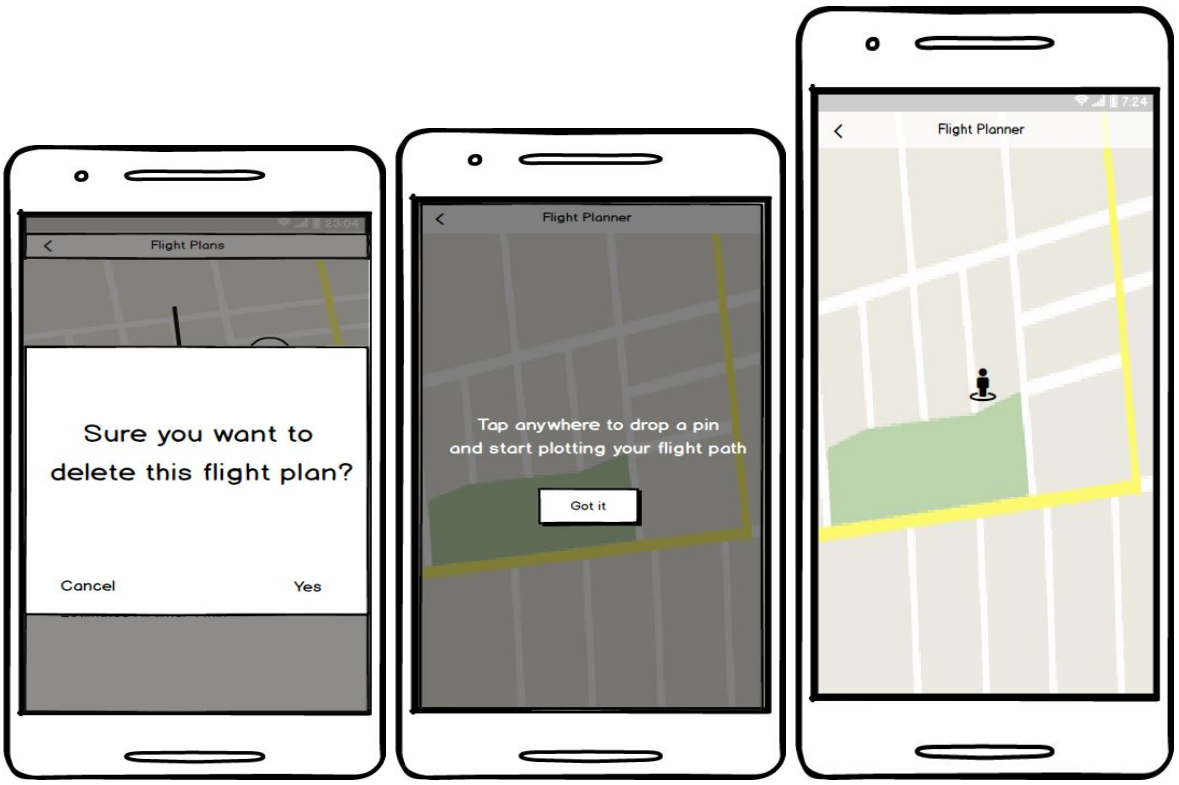
#223 Sequence Diagram

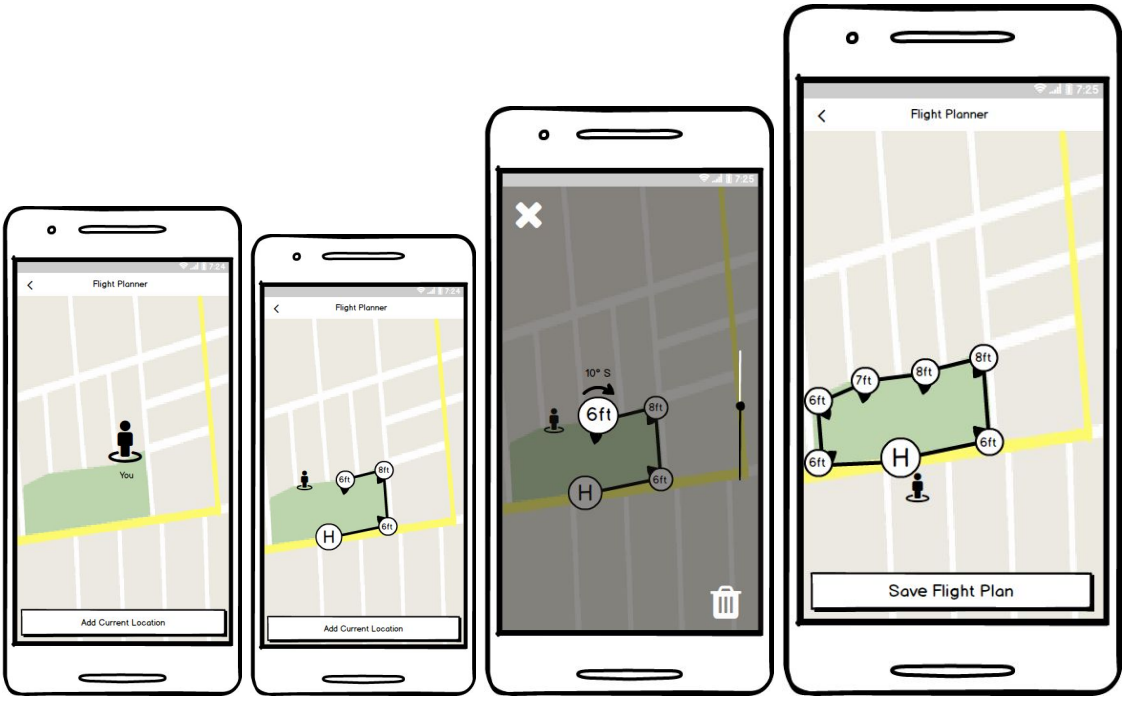


USer Story #210 Sequence Diagram

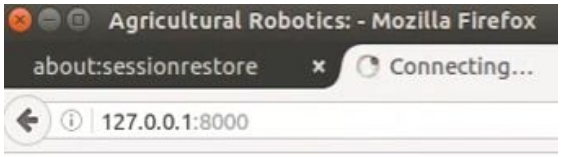
Appendix B - User Interface Design




Flight Plans
Sure you want to delete this flight plan?
Cancel
Yes
Flight Planner
Tap anywhere to drop a pin and start plotting your flight path
Got it
Flight Planner


Flight Planner
You
Add Current Location
Flight Planner
Add Current Location
10° S
6ft
8ft
H
Flight Planner
7ft
8ft
6ft
H
Save Flight Plan


SAMSUNG
8:11
UI: 60.0 fps
5 dropped so far
0 stutters (4+) so far
JS: 41.4 fps
pitch: -1333
yaw: 50971
roll: 756
altitude: 0
flyState: 0
gps: 25.7587713, -80.3736838
battery: 83%
accelero: 33.88211441040039, -130.3089141845703, -958.7865600585938
gyro: 0.21654601395130157, 0.14405499398708344, -0.23421655595302582
magneto: -21, 101, -37
windCompensationThetaAngle: 0
windCompensationPhiAngle: 0



Crop Stats

- My Crops
- Logout

13 hours ago by @test2
1 week, 1 day ago by @gabe
1 week, 1 day ago by @gabe
1 week, 1 day ago by @gabe
1 week, 1 day ago by @gabe
1 week, 1 day ago by @gabe
1 week, 5 days ago by @gabe /via

Appendix C - Sprint Review Reports

Sprint 1

Attendees: Wagner Vendrame, Leonardo Bobadilla, Miguel Chateloin, Franklin Abodo, Gabriel Barrs
Start time: 2016.9.9, 11:00AM
End time: 2016.9.9, 11:20 AM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: Wagner Vendrame.

- User Story 117 - Research Drone Hardware Platforms (Off-the-shelf vs Custom)
  - We found an off-the-shelf product that met all of our acceptance criteria; the ARDrone 2.0.

- User Story 118 - Research Drone Software Application Development Tools
  - We found one manufacturer-provided software development kit for mobile platforms (Android and iOS), and one open source Python library for operating the ARDrone.

- User Story 119 - Install Drone Application Development Tools
  - Miguel installed both the SDK and the Python library on his computer, and wrote an installation guide for the SDK.

- User Story 121 - Research Computer Vision Theory, Techniques, and Applications
  - Franklin found and started studying a bunch of literature and online courses on the topic.

- User Story 122 - Research Computer Vision Software Application Development Tools
  - Franklin found OpenCV, SimpleCV, and a bunch of convolutional neural network libraries for image classification and object detection.

- User Story 123 - Install Computer Vision Application Development Tools
  - Franklin has installed the SimpleCV and OpenCV python libraries, the OpenCV C++ library, the Darknet library (neural network programming), and the nvidia CUDA development kit (GPU programming).

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story 120 - Demonstrate All Drone Application Development Tools
- User Story 125 - Demonstrate All Computer Vision Application Development Tools
- User Story 126 - Collect and Store Computer Vision Literature and other References

Sprint 2

Attendees: Wagner Vendrame, Leonardo Bobadilla, Franklin Abodo, Gabriel Barrs, Miguel Chateloin
Start time: 2016.9.23, 2:30AM
End time: 2016.9.23, 2:50AM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story 122 - Collect and Store Computer Vision Literature and other References
- User Story 125 - Demonstrate Installation of All Computer Vision Application Development Tools
- User Story 176 - Research Parrot's Navdata and Streaming APIs
- User Story 181 - Integrate Parrot AR Drone 2.0 SDK into a React-Native Android Application

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story 120 - Demonstrate All Drone Application Development Tools
    - We demonstrated everything except the actual parrot ARDrone 2.0 API, which we ended up using for the basis of the app development (after exploring many other options that we did demo)
- User Story 129 - Prototype a User Interface via Research on React-Native Framework
    - It was decided that establishing a connection from the client device to the drone was more important than designing and prototyping the UI (too premature)

Sprint 3

Attendees: Leonardo Bobadilla, Franklin Abodo, Gabriel Barrs, Miguel Chateloin, Virgil, Nick
Start time: 10:00AM
End time: 10:20AM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners:

- User Story 197 - Setup Web Application Server
- User Story 184 - Design and Implement Image Segmentation Pipeline for Fruits
- User Story 121 - Research Computer Vision Techniques
- User Story 130 - Provide NavData Service to Client UI
- User Story 175 - Traditional Database

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story 163 - Design a website to view Crop data
    - How this should be reflected on the user story definition in Mingle:
        - No changes, just put on backlog to prioritize server API

Sprint 4

Attendees: Leonardo Bobadilla, Franklin Abodo, Gabriel Barrs, Miguel Chateloin
Start time: 12:00
End time: 12:20

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story User Story #202 - Design and Implement Data Model for Drone Mission Data
    - Uster Story #173 -Establish VM Server Communication
    - User Story #199 - Research Existing REST-like Web API Frameworks
    - User Story #203 - Install REST-like Web API Framework
    - USer Story #142 - Create a Flight Plan
    - User Story #204 - Manage Flight Plans
    - User Story #210 - Generate graphs of crops

Sprint 5

2016.10.27
Attendees: Franklin Abodo,Gabriel Barrs, Miguel Chateloin, Wagner Vendrame
Start time: 1:45PM
End time: 2:0PM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story #179 - Design A Segmentation Workflow/Pipeline for the Longan Fruit
- User Story #218 - Create Avocado Detector using the HoG Algorithm
- User Story #213 - Implement the Histogram of Gradients Object Detection Algorithm
- User Story 239 - Construct Training and Test Data Sets for the Neural Network
- User Story #223 -Provide A Drone Control Service to the User Interface
- User Story #211-Continue integrating/consolidating website and schema features into one solid platform
- User Story 199 - Research Existing REST-like Web API Frameworks

Sprint 6

2016.11.3
Attendees: Wagner Vendrame, Gabriel Barrs, Miguel Chateloin, Franklin Abodo
Start time: 11:00AM
End time: 11:15AM

After a show and tell presentation, the implementation of the following user stories were accepted by the product owners: All.

- User Story 179: Design A Segmentation Workflow/Pipeline for Longan Trees
    - User Story #237 - Unify CV Algorithms in a Single Python Module
    - User Story #155 - Persist Autonomous Flight Data to Device
    - User Story #240 - Create Longan Detector using the HoG Algorithm
    - User Story #154 - Launch an Autonomous Flight from Existing Plan
    - User Story #131- Provide a Video Streaming Service to the User Interface
    - User Story #238 - Port Final-State Project to Lab Workstation

The following ones were rejected and moved back to the product backlog to be assigned to a future sprint at a future Spring Planning meeting.

- User Story 212: Design A Segmentation Workflow/Pipeline for Coconut Trees

- How this should be reflected on the user story definition in Mingle:
  - The story actually doesn't change.

Appendix D - User Manuals, Installation/Maintenance Documents, Shortcomings/Wishlist Document and other documents

D.2 Installation and Maintenance Documents

D.2.2 Computer Vision Algorithm Engineering Environment Installation Guide

1. **Download and install the latest VirtualBox**

   1.1. Download VirtualBox here.

2. **Download and install Ubuntu**

   2.1. Download Ubuntu 14.04 (Trusty) 64-bit (same as the FIU web server VM) here.

3. **Download and install virtualenv:**

   In order to avoid tampering with the system, we will install all of the dependencies of our algorithm engineering environment in a python virtualenv. This allows us to install and use the latest version of Python2 if desired. Whenever we launch the virtualenv, we can simply declare Python 2.7.x to be our Python runtime environment.

   3.1. Download virtualenv here, and then extract the archive to your Downloads folder.

   3.2. To install virtualenv in /user/local/lib/python2.7/dist-packages from the Terminal, run:

      3.2.1. cd ~/Downloads/virtualenv-15.1.0
      3.2.2. sudo python setup.py install

   3.3. To create the algorithm engineering virtual environment in a dedicated directory:

      3.3.1. mkdir ~/VirtualEnvironments

3.3.2. cd ~/VirtualEnvironments
3.3.3. virtualenv cvdevenv (the pip package manager will be installed by default)

3.4. To confirm that the pip package manager is up to date, run:

3.4.1. cvdevenv/bin/pip install --upgrade pip

3.5. Next, we enter our cvdevenv using:

3.5.1. source cvdevenv/bin/activate (we should now have an updated prompt starting with "(cvdevenv) username@hostname:")

4. Download and install OpenCV:

4.1. Download the latest source code from GitHub:

4.1.1. Open a second Terminal tab/window, then run:
4.1.2. git clone https://github.com/opencv/opencv.git ~/Downloads
4.1.3. Open a third Terminal tab/window (while the download is in progress), then run:
4.1.4. git clone https://github.com/opencv/opencv_contrib.git ~/Downloads
4.1.5. Return to the first tab/window (while the downloads are in progress) to install OpenCV's dependencies.

4.2. Download and install OpenCV's dependencies:

4.2.1. sudo apt-get install build-essential
4.2.2. sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
4.2.3. sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev

4.3. Install OpenCV (once the downloads have completed), run:

4.3.1. cd ~/Downloads/opencv
4.3.2. mkdir release
4.3.3. cd release

4.3.4. cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D WITH_EIGEN=ON -D BUILD_DOCS=OFF -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_PERF_TESTS=OFF -D BUILD_TESTS=OFF -D INSTALL_C_EXAMPLES=ON -D BUILD_EXAMPLES=ON -D WITH_CSTRIPES=ON -D WITH_OPENCL=ON -D WITH_NVCUVID=ON -D WITH_CUDA=ON -D BUILD_opencv_gpu=OFF -D BUILD_opencv_gpuarithm=OFF -D BUILD_opencv_gpubgsegm=OFF -D BUILD_opencv_gpucodec=OFF -D BUILD_opencv_gpufeatures2d=OFF -D BUILD_opencv_gpufilters=OFF -D BUILD_opencv_gpuimgproc=OFF -D BUILD_opencv_gpulegacy=OFF -D BUILD_opencv_gpuoptflow=OFF -D BUILD_opencv_gpustereo=OFF -D BUILD_opencv_gpuwarping=OFF -D OPENCV_EXTRA_MODULES_PATH=~/Downloads/opencv_contrib/modules ..

4.3.5. sudo make

4.3.6. sudo make install

4.4. To make the installation visible to the Python runtime environment, open ~/.bashrc in your favorite text editor and append this line to the end of the file:

4.4.1. export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.7/site-packages

4.5. To make the change effective without having to restart your user session, run:

4.5.1. source ~/.bashrc

4.6. To reenter the virtual environment, run:

4.6.1. source cvdevenv/bin/activate

5. Download and install scikit-learn and scikit-image:

5.1. pip install -U scikit-learn

6. Download and install scikit-image:

6.1. pip install scikit-image

7. Download the Agricultural Robotics Ver 1.0 project from GitHub:

7.1. cd ~
7.2. mkdir GitHub
7.3. cd GitHub
7.4. git clone https://github.com/FIU-SCIS-Senior-Projects/Agricultural-Robotics-Ver-1.0.git

8. Download and install the IntelliJ PyCharm IDE for Python Development

8.1. PyCharm depends on Java, so we will first need to install the latest JDK. Download the JDK here.
8.2. cd ~/Downloads
8.3. Extract the archive jdk-NuNNN-linux-x64.tar.gz in the Downloads folder.
8.4. mkdir ~/opt_bank
8.5. mv jdk1.N.N_NNN ~/opt_bank

8.6. To make the installation visible to PyCharm, open ~/.bashrc in your favorite text editor and append this line to the end of the file:

8.6.1. export JDK_HOME=/home/YOURUSERNAME/opt_bank/jdk1.N.N_NNN

8.7. To make the change effective without having to restart your user session, run:

8.7.1. source ~/.bashrc

8.8. Download the IDE here. The commercial version is free for students.
8.9. Extract the archive pycharm-professional-NNNN.N.tar.gz in the Downloads folder.
8.10. mv pycharm-NNNN.N ~/opt_bank
8.11. cd ~/opt_bank

8.12. To launch and configure PyCharm for the first time:

8.12.1. ./pycharm.sh

8.13. Once you've registered the software using the FIU email address you provided to the developers at the time of the download, you may 'log in' and start a project. Be sure to lock the application to your dock so that you don't have to run the shell script again!

8.14. From the File menu, select New Project.

8.15. Be sure that Python 2.7 is the selected interpreter, and then select the Location to be /home/YOURUSERNAME/GitHub/Agricultural-Robotics-Ver-1.0

9. Install Darknet (optional):

The Darknet neural network framework optionally supports the NVIDIA CUDA API. It is imperative that any neural network included in FruiTREC's computer vision module take advantage of the computing power of a graphics processing unit (GPU).

9.1. You can install CUDA using the instructions here: http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#post-installation-actions

9.2. You can download the local installer here: https://developer.nvidia.com/cuda-downloads

Note: instead of running the two "export" commands mentioned in the guide at the command-line, you can have them execute automatically upon login by adding the commands to your ~/.bashrc file. In order for the change to take effect the first time, you must either start a new terminal session, or execute the following command:

source ~/.bashrc

9.3. You can install Darknet by following the guide here: http://pjreddie.com/darknet/

10. Have fun!

D.2.2 Backend and Web Development Environment Installation Guide

0.1 If you haven't already, clone the git repo:

https://github.com/FIU-SCIS-Senior-Projects/Agricultural-Robotics-Ver-1.0

0.2 Install Python3 virtual environments (optional):

sudo apt-get install python3-pip
sudo pip3 install virtualenv
virtualenv ~/.virtualenvs/fruitrecwebdev

To activate the virtual environment (needed each new terminal session):

source ~/.virtualenvs/fruitrecwebdev/bin/activate

1.1 Install Apache

DOWNLOAD the latest .tar.gz file here: https://httpd.apache.org/download.cgi

VERIFY the integrity of the files as described on the webpage. You may need to install pgpgpg using apt-get first.

INSTALL in the default location:

./configure;
make;
sudo make install;

To install in a custom location (as per https://httpd.apache.org/docs/2.4/install.html):

./configure --prefix=/home/fabodo/GitHub/Agricultural-Robotics-Ver-1.0/backend/apache;
make;
sudo make install;

TO CONFIGURE APACHE:

open conf/httpd.conf and replace ServerName www.example.com with ServerName your-servers-fully-qualified-domain-name:80

TO START APACHE:

sudo bin/apachectl -k start

*note that root permissions are required to bind to a reserved port (in the range 0-1024)

TO START APACHE AUTOMATICALLY ON BOOT:

Edit the /etc/rc.local file to add

/usr/local/apache2/bin/apachectl -k start

[update 2016.10.27] In order to run the apachectl status commang, the lynx web browser must be installed:

Sudo apt-get update
sudo apt-get install lynx

Then edit the conf/httpd.conf file to add:

```
<Location "/server-status">
   SetHandler server-status
</Location>
```

1.2 Install mod_wsgi

./configure
--with-apxs=/home/fabodo/GitHub/Agricultural-Robotics-Ver-1.0/backend/apache/bin/apxs
--with-python=/usr/bin/python3;

make;

sudo make install;

edit the conf/httpd.conf file to include LoadModule wsgi_module modules/mod_wsgi.so in an appropriate location

restart apache using:
sudo your-server-path-prefix/bin/apachectl restart

2.1 Install PostgreSQL

Note: We will install postgres 9.5.4 from source because 9.5 is the most recent version supported by psycopg2.

Download the source files from here: https://www.postgresql.org/ftp/source/v9.5.4/

Navigate to the unarchived postgres folder inside your downloads folder.

Run:

./configure
--prefix=/home/YOUR_USERNAME/GitHub/Agricultural-Robotics-Ver-1.0/backend/pgsql
make
make install

2.2 Setup PostgreSQL

To configure PostgreSQL for the first time (as detailed here: https://www.postgresql.org/docs/9.5/static/runtime.html):
Create a postgres-specific UNIX user account:

```
sudo adduser postgres
```

Change the working directory of the terminal to the default postgres install location:

```
cd /usr/local
```

Set the owner and group of the pgsql installation to the new postgres user and group:

```
sudo chown -R postgres:postgres pgsql
```

Now that the postgres user “owns” pgsql, we can configure using that account.

Create a database cluster:

su postgres -c 'pgsql/bin/initdb -D pgsql/data’

Open /home/postgres/.bashrc in a text editor and append the following line:

export PGDATA=/usr/local/pgsql/data

To take advantage of the new environment variable immediately:

source /home/postgres/.bashrc

Create a logfile for PostgreSQL:

su postgres -c 'touch pgsql/logfile'

To have the server run automatically on startup, open /etc/rc.local and append the following line:

/usr/local/pgsql/bin/pg_ctl start -l /usr/local/pgsql/logfile -D /usr/local/pgsql/data

2.3 Testing the database connectivity

By default, psql will use your Linux username as default value for the database-username, so it’s important that we were logged in as postgres when running the initdb command. Otherwise, we’ll need to use the -U flag to specify our own UNIX username when manipulating the db.

To run the server in the background (if the PGDATA env var is not set, use the -D flag to specify the path to your database cluster):

su postgres -c 'bin/pg_ctl start -l logfilestatic'

A simple first test of a proper configuration is to simply list the three databases that are installed by default:

su postgres -c 'bin/psql -l’

To shut down the server (as detailed at https://www.postgresql.org/docs/9.5/static/app-pg-ctl.html):

su postgres -c 'pg_ctl stop’

2.3.3 Creating a new db within our data cluster

To create a new db for the site (as detailed https://www.postgresql.org/docs/9.5/static/tutorial-createdb.html):

su postgres -c 'createdb fruitrecdb'

Make sure that we our db creation succeeded by running:

su postgres -c psql fruitrecdb'

3 Install psycopg2

Download the source files from here: http://initd.org/psycopg/

Follow the instructions here, under the heading Use the source package:
http://initd.org/psycopg/docs/install.html#install-from-source

Note that when installing psycopg2, we will build using a non-default pg_config location:

In ~/.bashrc, add the line:

export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python3.4/dist-packages

Next, run:
source ~/.bashrc
python3 setup.py build_ext --pg-config
/home/fabodo/GitHub/Agricultural-Robotics-Ver-1.0/backend/pgsql/bin/pg_config build

sudo python3 setup.py install

4 Install Django (ignore italicized, block-quoted part)

pip3 install -e /path/to/your/local/clone/django/

4.1 Configure Django

Replace:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
   }
}
```

With:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.postgresql',
      'NAME': 'fruitrecdb',
      'USER': 'postgres',
      'PASSWORD': 'pg967je',
      'HOST': 'localhost',
      'PORT': '5432', #we will need to change the port at some point
   }
}
```

Replace: TIME_ZONE = 'UTC'
with: TIME_ZONE = 'America/New_York'

Now we want to test the connection between django and postgres by creating databases for each of the apps that are installed with django by default (see)

Now, lets continue following tutorial 2:

```
python3 manage.py migrate
```

Let's confirm the success of the migration by entering the postgres command prompt:

```
sudo -u postgres pgsql/bin/psql fruitrecdb
```

And then:

```
\dt
```

5 Install Graphene (GraphQL API framework)

sudo pip install "graphene-django>=1.0"

5.1 SWAPI demo graphsql-python demo dependencies

sudo pip3 install django-filter
cd ~/GitHub/Agricultural-Robotics-Ver-1.0/demos
git clone https://github.com/graphql-python/swapi-graphene.git

D.2.3 Web Application API Design Guide

After setting up the development environment, we're ready to create our app. Below are some resources you may find useful for learning about the chosen development tools.

Django, PostgreSQL, and the JSONField:

http://stackoverflow.com/questions/22340258/django-list-field-in-model
https://www.youtube.com/watch?v=xGkH2yoy2MY
https://docs.djangoproject.com/en/1.9/ref/contrib/postgres/fields/#django.contrib.postgres.fields.JSONField

The Django project structure:

http://www.revsys.com/blog/2014/nov/21/recommended-django-project-layout/

Django, GraphQL, and Graphene:

https://www.youtube.com/watch?v=BG1H6IrNbAk
https://www.youtube.com/watch?v=ND9GWSkbUGM&spfreload=5
https://www.youtube.com/watch?v=UBGzsb2UkeY
https://www.youtube.com/watch?v=bPygoeZVL5g

JSON:

http://www.json.org

http://jsonapi.org

D.2.4 Android Development Environment Installation Guide

The following instructions assume we're all developing on an Ubuntu system.

Prerequisites:

- JDK 1.8:
- node.js (version 4 or newer)
- NPM (Node Package Manager)

1 - Verify your JDK default version is 1.8.

We need this to run Android Studio. Verify the default version by running to the following in your terminal:

```
$ java -version
java version "1.8.0_101"
Java(TM) SE Runtime Environment (build 1.8.0_101-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
```

If your output shows a different version of the JDK, run the following to check if JDK 1.8 is still somewhere else on your machine:

```
$ dpkg --list | grep -i jdk
ii  oracle-java8-installer        8u101+8u101arm-1~webupd8~2      all       Oracle Java(TM) Development Kit (JDK) 8
ii  oracle-java8-set-default      8u101+8u101arm-1~webupd8~2      all       Set Oracle JDK 8 as default Java
```

If you don't see it there, refer to the prerequisites section to install it. If it is there, make it the default by running the following:

```
$ sudo update-alternatives --config java
```

You'll be prompted to select the JDK version. Type out the number corresponding to JDK 1.8 in that list and hit enter.

2 - Download Android Studio from here.

This is the IDE we will be using to develop the application. Once your zip file is finished downloading, follow the installation instructions from this page: https://developer.android.com/studio/install.html

When Android Studio finishes installing, don't create a project yet. Stay on the welcome screen. We need to first find the location of the Android SDK that Android Studio just kindly installed for us.

3 - Add SDK path to an environment variable.

From the Android Studio welcome screen, bottom right, go to **Configure > Project Defaults > Project Structure**. You should see the "Android SDK Location" field. Copy the path from that field.

We need this path in an environment variable. Assuming you're using bash, you can do this by editing your bash profile file in your home directory. It will either be **.bash_login**, **.bash_profile**, or **.profile** (usually .profile). Note that if you have more than one, they override each other in that order.

$ nano ~/.profile

At the bottom of the file write the following, pasting in your path after the '=' sign.

export ANDROID_HOME=/your/android/sdk/path

Mine looks like this:

export ANDROID_HOME=/home/miguel/Android/Sdk

Save, and close the file (Ctrl+O, Ctrl+X). Run the following to reload your new profile.

$ source ~/.profile

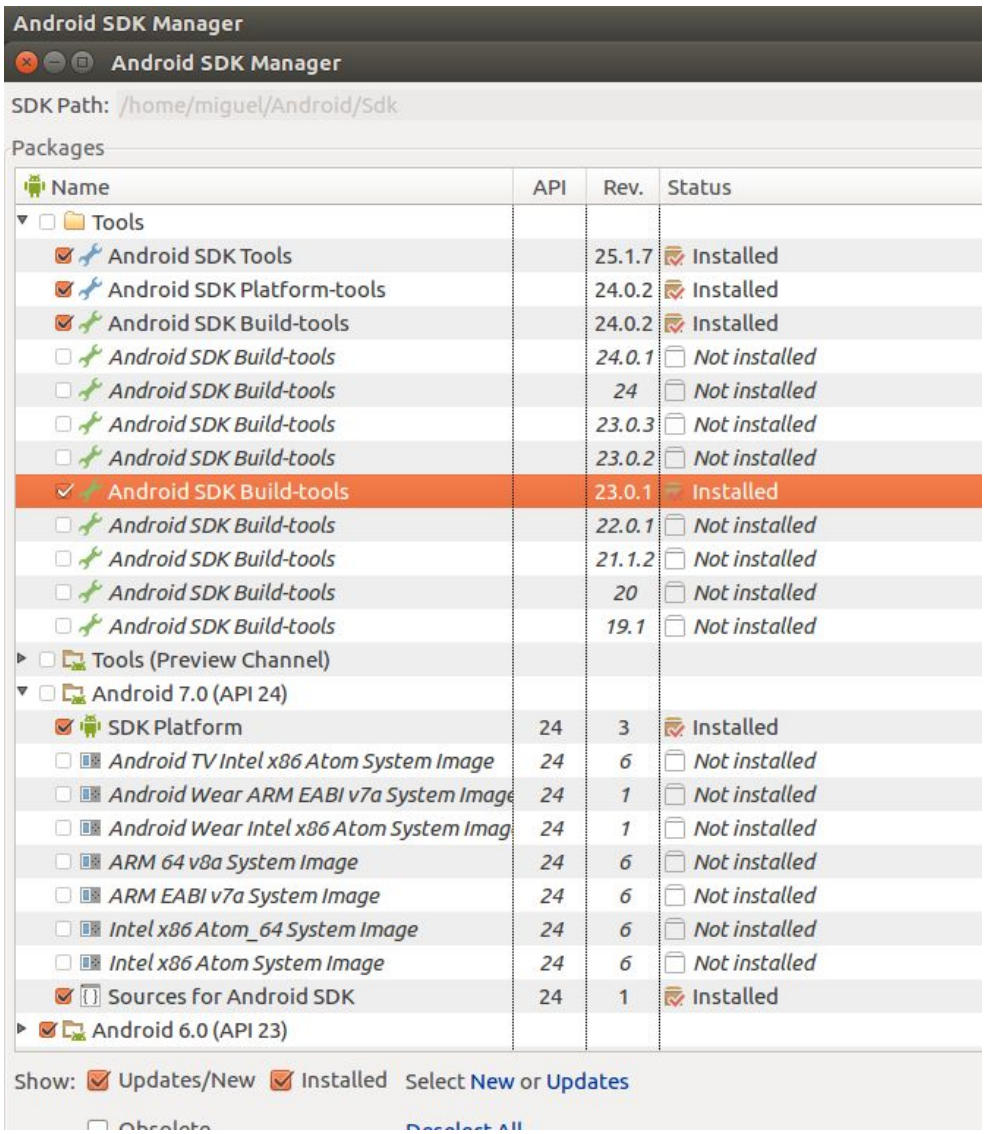
Check your variable does in fact exist:
$ echo $ANDROID_HOME
/home/miguel/Android/Sdk

4 - Install additional libraries for Android SDK

Run the Android SDK Manager which is located at <YOUR_SDK_PATH>/tools/android

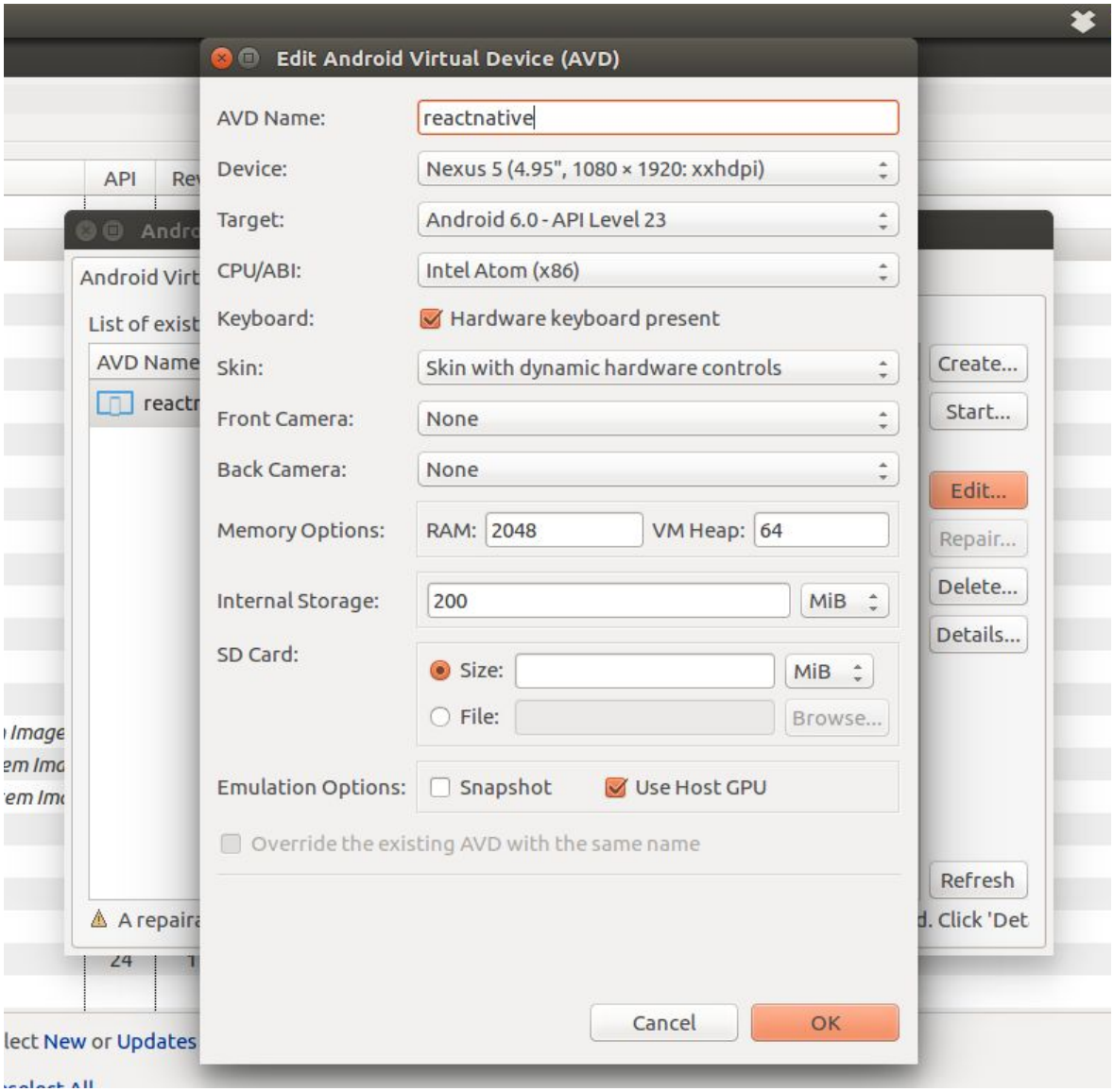
$ ./home/miguel/Android/Sdk/tools/android

Check the following and hit **Install packages** at the bottom right. Now go get coffee, lunch, Netflix, or whatever, because this will take a while. Don't close the manager when you're done.



5 - Create an Android Virtual Device

From the Android SDK Manager's top menu bar, go to **Tools > Manage AVDs > Android Virtual Devices** tab **> Create** button. Make sure all fields are entered exactly the same as shown below. The AVD Name can be different.



After you're finished creating it, select your device from the list in the **Android Virtual Devices** tab and click the **Start** button. Your virtual device should start up. Leave it running for now.

Confirm that you see a device listed by running:
$ /home/miguel/Android/Sdk/platform-tools/adb devices

If you don't see it, close Android Studio, and run the following:

$ /home/miguel/Android/Sdk/platform-tools/adb kill-server

Then try listing the devices again.

6 - Install React Native via NPM

- Follow the instructions here to install React Native and create your first project, but **do not run it with the run-android command yet!**

When your project directory is ready, cd into it and replace everything in the **package.json** file with the following:

```
{
  "name": "YOUR_PROJECT_DIRECTORY_NAME",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "build": "(cd android/ && ./gradlew assembleDebug)",
    "start": "node node_modules/react-native/local-cli/cli.js start",
    "bundle-android": "react-native bundle --platform android --dev false --entry-file
index.android.js --bundle-output android/app/src/main/assets/index.android.bundle
--sourcemap-output android/app/src/main/assets/index.android.map --assets-dest
android/app/src/main/res/",
    "flow": "flow; test $? -eq 0 -o $? -eq 2"
  },
  "dependencies": {
    "react": "15.3.1",
    "react-native": "0.32.0"
  },
  "devDependencies": {
    "flow-bin": "^0.31.1"
  }
}
```

There's some extra dependencies here that This should set us up to build automatically with Gradle, as files change in our project.

From your project root run the following:

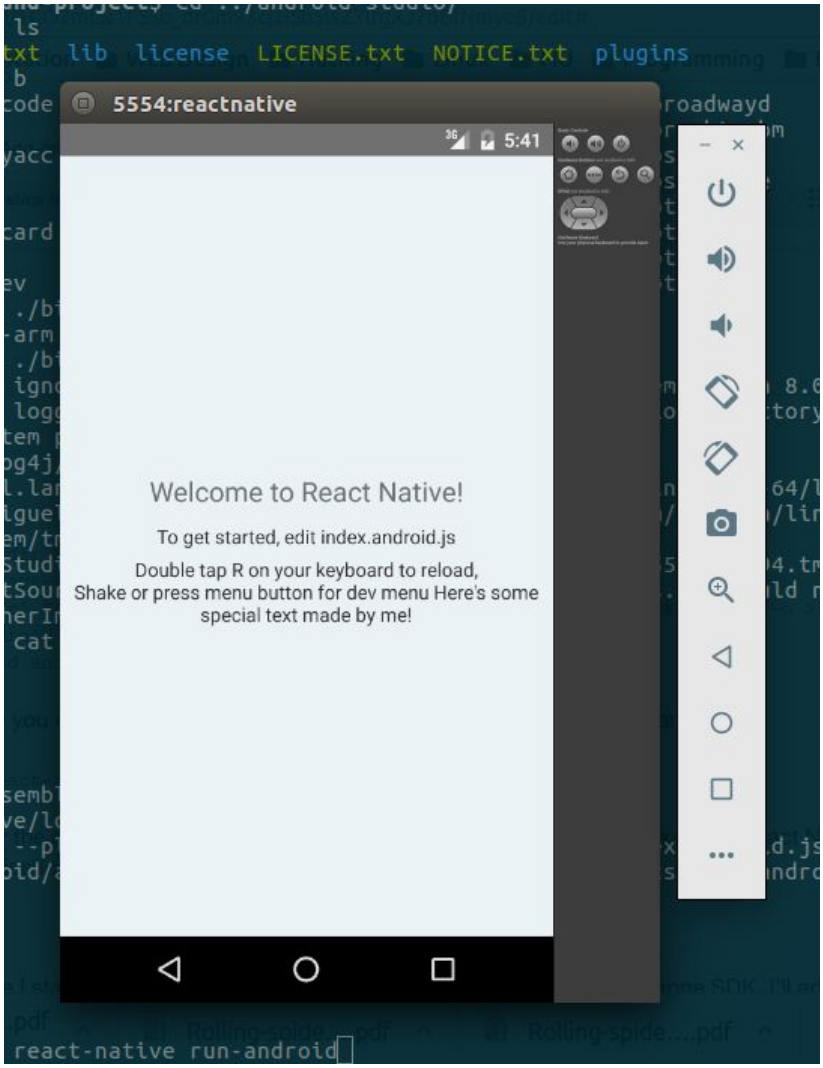
```
$ mkdir android/app/src/main/assets
$ react-native start > /dev/null 2>&1 &
```

$ curl "http://localhost:8081/index.android.bundle?platform=android" -o >
"android/app/src/main/assets/index.android.bundle"
$ (cd android/ && ./gradlew assembleDebug)

Now you can start your React Native project with the following command:

$ react-native run-android

After the output stops, your virtual device should be loaded with the example React Native application. This is what you should see:



From now on, you can edit the index.android.js file and double tap “R”, to reload and see your changes. You also don’t need to run all the commands we just went through each time you start

development. Just start up your emulator from Android Studio or the AVD manager and then run the react-native run-android command to bring up your application.

Instructions for working with physical device coming soon!
7 - Enabling the Gradle Daemon
That last build took a while. To cut down build times during development, we're going to enable Gradle, which has already been installed on your machine by Android Studio. From the Gradle website:

"The Gradle Daemon is a background process that does the heavy lifting of running builds, then stays alive between builds waiting for the next build. This allows data and code that is likely to be required in the next build to be kept in memory, ready to go. This dramatically improves the performance of subsequent builds. Enabling the Gradle Daemon is an extremely cheap way to decrease build times."

Run the following command to enable Gradle:

touch ~/.gradle/gradle.properties && echo "org.gradle.daemon=true" >>
~/.gradle/gradle.properties
8 - Developing native code in Android Studio

Finally we can actually open our project in Android Studio. From the welcome screen, select **Import project** and select the **android** folder in the root of the React Native project you created earlier. Do not import it from the project root directory. Android Studio will not recognize all the project settings for tools like Gradle if it attempts to import the root directory as a project.

9 - Developing React Native code in Nuclide/Atom

So we'll actually need to use two IDEs: Android Studio for the native application code that will run on Android, and Nuclide for the React Native components. Nuclide was developed by Facebook specifically for React Native. It's highly specialized, and has the best support for debugging.

Run through the installation here. I recommend you skip the apt-get upgrade though. It's actually optional. And of course, skip the initial Atom installation if you already have Atom.

Note that Nuclide is just a giant package/plugin built on top of Atom, which is the real IDE underneath. Launching Atom will always launch Nuclide from now on because its package has

been enabled in the IDE. You can always disable the Nuclide package it if you need to keep doing work in plain old Atom outside of our project.

You might get an error when you start Atom saying “Unable to watch path: config.cson.” Follow the instructions here to fix this.

Flow was not found when attempting to start it in '/home/miguel/Development/android-playground-project'.low syntax

Now connect your tablet to your machine with a micro-USB cable. Now try running the react-native run-android command again.

References

Agricultural Robotics at the University of Minnesota:

http://agricultural-robotics.cs.umn.edu/

Robotic Surveying of Apple Orchards:

https://www.cs.umn.edu/sites/cs.umn.edu/files/tech_reports/15-010.pdf

Existing Agricultural Robotics Applications:

https://www.sensefly.com/applications/agriculture.html

https://www.dronedeploy.com/

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwjq25Orz4LPAhVDHR4KHcRGBZMQFggqMAE&url=http%3A%2F%2Fwww.mdpi.com%2F1424-8220%2F16%2F8%2F1222%2Fpdf&usg=AFQjCNFe_eX0b1t4zyVaVnuajskbnjVXwQ

http://cdn.intechopen.com/pdfs-wm/9521.pdf

Normalized Difference Vegetation Index:

http://earthobservatory.nasa.gov/Features/MeasuringVegetation/measuring_vegetation_2.php

Computer Vision:

OpenCV: http://opencv.org
scikit-image: http://scikit-image.org

Machine Learning:

scikit-learn: http://scikit-learn.org/stable/

Data Mining Techniques and Applications to Agricultural Yield Data:

http://ijarcce.com/upload/2013/september/31-h-dantam%20ramesh%20-data%20mining%20techniques%20and%20application%20to.pdf