*Florida International University*

*School of Computing and Information Sciences*

Software Engineering Focus

Final Deliverable

Agricultural Robotics 3.0

**Team Members: Sean Monroe and Christian Silva**

**Product Owner(s)**: Dr. Leonardo Bobadilla

**Mentor(s)**: Dr. Leonardo Bobadilla

**Instructor**: Masoud Sadjadi

The MIT License (MIT)

Copyright (c) *2016 Florida International University*

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

***Abstract***

*This document presents the information necessary to gain a good understanding of the tools, hardware and software system used to continue development of Agricultural Drone 3.0, a system designed to provide a desktop application to operators of a Parrot AR Drone 2.0 that allows them to deliver waypoint targets and routes to be autonomously executed, as well as manual controls, a live camera and sensor data feed, and live image processing.*

*This project is motivated by FIU’s Senior Project course and the efforts of previous students to work with the FruiTREC project of the University of Florida’s Tropical Research Center. With the ultimate goal of monitoring the development of horticultural characteristics (HC’s) of tropical crops during the use of a new mineral mixture meant to replace fertilizer.*

*Assessing the progress of HC’s requires individual attention to crop yields of fruiting trees. These can be very tall and spread across large areas, requiring an excessive amount of manual labor for proper analysis. The use of unmanned drones in HC data collection can decrease required labor as well as providing a safe alternative to climbing trees and interacting with chemicals being tested. Data entry tasks can also be automated and presented in human-readable form to provide in-depth analysis at a level of detail not possible by human researchers.*

*The objective of this project is to combine these tasks of autonomous data collection of HCs, recording and storing data on a remote server, displaying the data to researchers, and providing an interface for all parts of the system.*

**Table of Contents**

**Introduction**……………………………………………………………………………………….. 5

Current System.…………………………………………………………………………………. 5

Purpose of New System ………………………..………………………………………………... 5

**User Stories**

Implemented User Stories ...……………….…………………………………………………….. 6

Pending User Stories ………………………………………...…………………………………. 12

**Project Plan**

Hardware and Software Resources ………………………………...………………………….. 13

Sprints Plan ……………………...………..……………………………………………………. 14

*Sprint 1*  …….……………………………………………………………………………….... 14

*Sprint 2*  …….……………………………………………………………………………….... 15

*Sprint 3* ..……………………………………………………………………………………… 16

*Sprint 4*  .……………………………………………………………………………………… 16

**System Design**

Architectural Patterns .……………………………....……………………………………….. 17

System and Subsystem Decomposition ………....……………………………………………….. 18

Deployment Diagram ……………………………………………………………...……………. 19

Design Patterns ……………………………………………………………………………........ 19

**System Validation**  ………………………………………………………………………………... 20

**Glossary**  …………………………………………………………………………………………... 30

**Appendix**  ………………………………………………………………………………………….... 31

Appendix A - UML Diagrams …………………………………………………………………... 31

*Class UML Diagrams*  ………………………………………………………………………... 31

*Use Case UML Diagrams*  …………………………………………………………………… 34

*Sequence UML Diagrams*………………………………………………………………..…… 38

Appendix B - User Interface Design ……………………………………………………..……... 43

Appendix C - Sprint Review Reports …………………………………………………..………... 45

Appendix D - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents ………………………………………………………………………………………... 47

**References** …………………………………………………………………………………...……. 52

**INTRODUCTION**

Agricultural Robotics version 3.0 is a software application designed to incorporate features that provide useful data and feedback for agricultural research and analysis. The current versions goal is to develop an interface between the drone and drone operator that will allow aerial coverage of specific map regions for analysis. The drone operator would select a particular region of interest that requires research, and the drone will successfully navigate the region. During navigation the drone will scan the region, using sensor feedback and computer vision algorithms, to detect different types of fruit. Statistics of these fruit such as size, color, and their number, are uploaded to a database. Analysis of these statistics over time would ultimately reveal characteristic changes in fruit development (i.e. growth, health, yield). The data analyzed would be provided to the drone operator in reports and visual data (e.g. thermal mapping).

**Current System**

The current implementation of this application was developed in Agricultural Robotics version 1.0. In this project iteration, the developers used an android mobile application as the interface and a Node/Apache/Django stack to provide the data storage service. Using the Android application the drone would be piloted, by selecting a path, to regions of research interest and with the drones computer vision collect data pertaining to the number of fruit in a tree. This information would then be stored in a database and made available to the operator online.

**Purpose of New System**

The new implementation continues to have the same goal as the previous iteration; to provide researchers and growers with usable data to improve the research of tropical agriculture. However, the new system implements a custom interface using Python 2.7 libraries exclusively to perform the combined tasks of computer vision, drone sensor data processing and drone control.

**User Stories**

The following section provides the detailed user stories that were implemented in this iteration of the Agricultural Robotics 3.0 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**

**#262 - GPS coordinate and sensor data acquisition**

Description:

As a drone programmer, I want the drone to acquire its GPS coordinates and compass heading, accelerometer data, and gyroscope data so that I can successfully pilot the drone.

Acceptance Criteria:

1. Clear, sunny weather
2. Open area with no buildings or obstacles

**#263 - Research drone commands**

Description:

As a drone programmer, I want to to command the drone to perform precise movements so that I accurately pilot the drone.

Acceptance Criteria:

1. Commands must use distances and time

**#264 - Flight controller**

Description:

As a drone user, I want to have a graphical interface with the drone so that I can send it commands through button selections, view real-time sensor data, and see live video so that I can more effectively pilot the drone.

Acceptance Criteria:

1. Open area with no obstacles.
2. Python 2.7

Related Tasks:

1. Integrate navigation objects with GUI
2. Add real-time velocity display
3. Add real-time altitude display
4. Add real-time battery status
5. Add Drone controls
6. Add Drone test buttons to GUI
7. Add GUI interactive buttons
8. Create Basic GUI

**#265 - Drone path navigation**

Description:

As a drone programmer, I want to establish the code necessary to have the drone be able to travel a predefined path from one GPS coordinate to another so that I do not have to manually pilot the drone through a path.

Acceptance Criteria:

1. Clear and sunny weather
2. Open area with no obstacles
3. Clear GPS signal

Related Tasks:

1. Design simple flight scripts
2. Research and implement navigation algorithms

**#272 - Verify Equipment**

Description:

As a drone programmer, I want to know what hardware is available to me for development purposes so that I can plan accordingly and order new parts or repair what I have if needed.

Acceptance Criteria:

1. Only drone kits available in discovery lab.

**#273 - Research Python GUI**

Description:

As a drone programmer, I want to be able to provide an interactive graphical interface so that users easily control the drone. A suitable GUI library must be found to provide this.

Acceptance Criteria:

1. Uses Python 2.7
2. Easily implemented

**#289 - GPS Map**

Description:

As a drone user, I want to be given a map with which I can interact and give the drone locations to move so that I can visually determine regions the drone should survey.

Acceptance Criteria:

1. Fully integrated into GUI
2. “Box drawing” to provide the drone with a region of interest (ROI)
3. Uses Google API to provide or derive GPS coordinates

Related Tasks:

1. Static map
2. Area boxing
3. Enable interactive map in GUI
4. Accommodate a map in GUI
5. Research Google Maps API

**#290 - Integrate sensor data into controls**

Description:

As a drone programmer, I want the drone to be able to perform basic responses to its accelerometer, GPS and camera data so that I can develop mid-flight course corrections, object detection and collision avoidance.

Acceptance Criteria:

1. New waypoints inserted into current queue
2. New targets overriding current targets, with optional preservation of old target
3. Live video feed to the GUI with basic color detection
4. Manual control disengages automated control
5. Movement accommodates wind interference

Related Tasks:

1. Image recognition
2. Course correction
3. Dynamic targeting
4. Live video feed

**#292 - Multiple -Waypoint Navigation**

Description:

As a drone programmer, I want the drone to be able to be given a list of waypoints and determine an ideal path through them so that I do not have to manually determine the ideal flight path.

Acceptance Criteria:

1. Uses shortest paths possible
2. Does not use obstacle maps or avoidance

Related Tasks:

1. Refactoring and cleaning
2. GUI Object
3. User-friendly feedback
4. Area navigation
5. Simulation script
6. Drone flight
7. Obtaining targets
8. Waypoint queue
9. Research pathing algorithms

**Pending User Stories**

**#266 - Total Autonomy**

Description:

As a drone operator, I want the drone to perform its tasks without requiring an active network connection such that after only a “begin”-like command, it will act entirely autonomously so that I can prioritize other research while the drone operates.

**#268 - Obstacle Avoidance**

Description:

As a drone operator, I want the drone to be able to travel to a specified point while avoiding obstacles and using the shortest path possible so that I can reduce that risk of manual piloting error.

**#269 - Maximal Coverage**

Description:

As a drone operator, I want the drone to determine and travel a path of maximum visual coverage within an indicated region so that I can retrieve maximum data for research.

**#271 - Patrolling**

Description:

As a drone operator, I want the drone to determine and repeatedly travel paths of coverage within an indicated region so that I can consistently survey specific regions.

**#291 - Emergency Drone Behavior**

Description:

As a drone programmer, I want the drone to engage in automated behavior in response to any emergency situations and ideally recover from them without requiring a reboot so that I can reduce the risk of equipment loss.

Acceptance Criteria:

1. Runtime error handling established for all existing code
2. GPS, sensor data and network connection loss handled
3. Recovery is attempted, followed by returning home or (in case of GPS loss) simply landing.

**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:

* **Thoughtworks Mingle -** A project management tool that enables the implementation of the AGILE software development process.
* **Google Drive -** Highly accessible cloud based storage center.
* **Github -** A software development platform used for version control
* **Atom -** An Open source text editor developed by Github
* **Vi -** A Unix based screen-oriented text editor
* **Python -** A general-purpose and widely used high-level programming language.
* **OpenCV3 -** An open source computer vision and machine learning software library with easily modifiable code.
* **Mac OS X -** Unix-based graphical operating system
* **Arch Linux -** Minimalistic Linux operating system distribution

Hardware tools:

* **Parrot AR 2.0 Drone -** Smartphone or tablet controlled quadricopter.

**Sprints Plan**

Sprint 1

**Attendees**: Christian Silva, Sean Monroe and Leonardo Bobadilla

**Start time**: 5/23/2017 10:00:00 AM

**End time**: 5/23/2017 10:30:00 AM

After discussion, the velocity of the team was unknown.

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

* User Story #272: Verify Equipment
* User Story #273: Research Python GUI
* User Story #262: GPS coordinate and sensor data acquisition
* User Story #263: Research drone commands
* User Story #264: Flight controller
* User Story #265: Drone path navigation

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

* Christian Silva
* User Story #273: Research Python GUI
* User Story #263: Research drone commands
* User Story #264: Flight controller
* Sean Monroe
* User Story #272: Verify Equipment
* User Story #262: GPS coordinate and sensor data acquisition
* User Story #265: Drone path navigation

Sprint 2

**Attendees**: Christian Silva, Sean Monroe, Tauhidul Alam, and Leonardo Bobadilla

**Start time**: 5/30/2017 2:30:00 PM

**End time**: 5/30/2017 2:40:00 PM

After discussion, the velocity of the team were estimated to be 2.

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

* User Story #262: GPS coordinate and sensor data acquisition
* User Story #263: Research drone commands
* User Story #264: Flight controller
* User Story #265: Drone path navigation

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

* Christian Silva
* User Story #273: Research Python GUI
* User Story #263: Research drone commands
* User Story #264: Flight controller
* Sean Monroe
* User Story #262: GPS coordinate and sensor data acquisition
* User Story #265: Drone path navigation

Sprint 3

**Attendees**: Christian Silva, Sean Monroe and Leonardo Bobadilla

**Start time**: 6/12/2017 08:00:00 PM

**End time**: 6/12/2017 08:28:00 PM

After discussion, the velocity of the team were estimated to be 26.

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

* User Story #289: GPS Map
* User Story #290: Integrate sensor data into controller
* User Story #292: Multiple-Waypoint Navigation

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

* Christian Silva
* User Story #289: GPS Map
* Sean Monroe
* User Story #290: Integrate sensor data into controller
* User Story #292:Multiple-Waypoint Navigation

Sprint 4

**Attendees**: Christian Silva, Sean Monroe, Tauhidul Alam, and Leonardo Bobadilla

**Start time**: 7/17/2017 7:05:00 PM

**End time**: 7/17/2017 7:15:00 PM

After discussion, the velocity of the team were estimated to be 42.

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

* None for next sprint - Working on documentation

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

* Christian Silva
* N/A
* Sean Monroe
* N/A

**System Design**

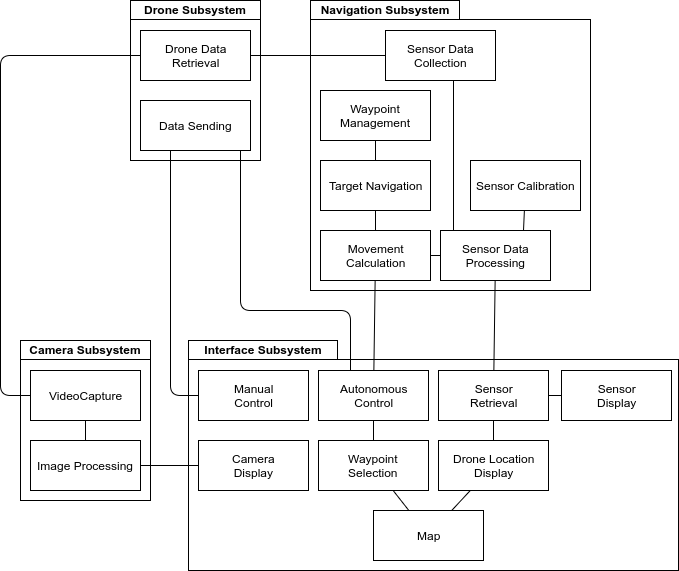
This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

**Architectural Pattern**

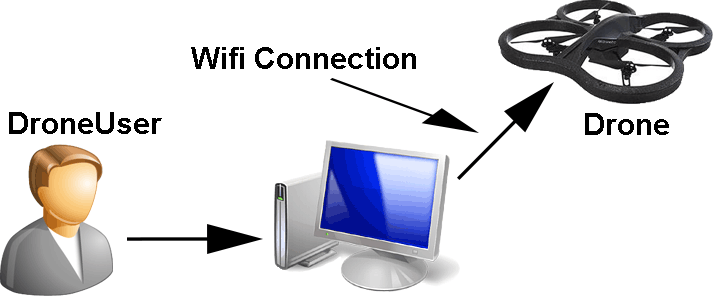
**System Architectural Pattern:** Client-server

* The Drone and Camera objects act as servers to the Navigator object, which in turn is a server for the GUI object.
* The Drone and Camera objects asynchronously act to fetch sensor data to hold until called for. The Navigator object also continuously fetches data from the Drone, ready to prepare any requested sensor data.
* When called on, the Camera and Navigator objects prepare data to be sent, only performing any processing when necessary.

**System and Subsystem Decomposition**



**Deployment Diagram**



**Design Patterns**

* Module: Demonstrated by the Drone class, being designed to be an all-in-one API for interacting with the Parrot AR 2.0 Drone. It contains communications protocols for getting and setting configuration and navigation data as well as providing basic functions that are passed through the comms protocols to command the drone to perform actions.
* Proxy: The Navigator and Viewer classes contain behaviors that closely resemble those of the Drone class and are functionally similar. However, they provide the additional feature of being easily accessible and conceptually separate from each other. This makes them easier to access, debug, and expand.
* Singleton: All objects are necessarily created only once; they each encapsulate a specific set of behaviors and functions and are used by the GUI object that pools their resources to perform commands issued by the user. Consequently, there is a high amount of coupling between these objects, and while by the nature of their implementation they are relatively stable, the relationships between them must be kept in mind when introducing new functionality.

**System Validation**

Test Suite ID 01

**Purpose:** To test the functionality of use case: **Manual Flight**

**Test case 1:**

Purpose: Perform flight action with move command

Pre-condition: Drone is hovering

Input: User selects a move command button.

Expected Results: Drone performs command while no new command is transmitted.

Actual Result: Drone performed selected command continuously until new command was

given.

Status (Fail/Pass): Pass

**Test case 2:**

Purpose: Perform flight action with land command

Pre-condition: Drone is hovering

Input: User selects the “Land” command button.

Expected Results: Drone breaks current action and performs land action.

Actual Result: Drone halted its current action and performed land action.

Test Suite ID 02

**Purpose:** To test the functionality of use case: **Waypoint Navigation**

**Test case 1:**

Purpose: Command the drone to perform a series of simple pre-made actions.

Pre-condition:

1. Successful connection to the drone.
2. Acceptable weather conditions.

Input: User selects a move command button.

Expected Results: The drone should fly forward for two seconds, backward for two seconds, left

for two seconds, right for two seconds, upward for two seconds, downward

for two seconds, rotate left for two seconds, rotate right for two seconds, and

land.

Actual Result: Drone performed all expected motions.

Status (Fail/Pass): Pass

Test Suite ID 03

**Purpose:** To test the functionality of use case: **Render rectangle and markers**

**Test case 1:**

Purpose: Render single point marker.

Pre-condition: Waypoint navigation selected.

Input: User selects a location on the map.

Expected Results: Map displays marker on selected location and adds point to waypoint list.

Actual Result: Each map selection generated map marker in selected location and list was

updated with correct x,y values.

Status (Fail/Pass): Pass

**Test case 2:**

Purpose: Render rectangle and markers.

Pre-condition: Region of interest navigation selected.

Input: User selects two locations on the map.

Expected Results: Map displays rectangle and updates waypoint list.

Actual Result: Generated a rectangle with markers at each vertex. Waypoints list was

updated to contain four vertex coordinate pairs.

Status (Fail/Pass): Pass

Test Suite ID 04

**Purpose:** To test the functionality of use case: **Render Path**

**Test case 1:**

Purpose: Render path of single selections.

Pre-condition: Waypoint navigation option selected

Input: Drone operator selects map location.

Expected Results: GUI map renders edge from drone to selected vertex and selection added to

waypoint list.

Actual Result: GUI map rendered edge from drone to new vertex. Each new map selection

renders edge from current vertex to new vertex and new selection is added to

waypoints list.

Status (Fail/Pass): Pass

**Test case 2:**

Purpose: Render region path.

Pre-condition: Region of Interest navigation option selected

Input: Drone operator selects two map locations.

Expected Results: GUI map renders row wise path creating short edges along the perimeter and

long edges along the interior from opposite edges. Navigator generates all

path vertices and adds them to waypoints list.

Actual Result: GUI map rendered row wise path and waypoints list was populated with all

generated vertices.

Status (Fail/Pass): Pass

Test Suite ID 05

**Purpose:** To test the functionality of use case: **Live Image Feed**

**Test case 1:**

Purpose: Run camera in test program.

Pre-condition: Drone is turned on and connected

Input: Drone programmer runs test program.

Expected Results: Camera feed connected to and displaying in window.

Actual Result: Camera feed displays in window

Status (Fail/Pass): Pass

**Test case 2:**

Purpose: Camera feed in GUI.

Pre-condition: Drone is turned on.

Input: Drone programmer clicks “Connect” button.

Expected Results: Drone connects and live video shows in GUI.

Actual Result: Drone connects and live video shows in GUI.

Status (Fail/Pass): Pass

**Test case 3:**

Purpose: Color detection in live feed.

Pre-condition: Drone connected and live stream on.

Input: Drone programmer clicks “Blue” button.

Expected Results: Blue objects are outlined in the image.

Actual Result: Outlines around blue objects are present.

Status (Fail/Pass): Pass

**Test case 4:**

Purpose: Manual override of autonomous functions.

Pre-condition: Drone is performing an autonomous flight command.

Input: Drone user clicks “Hover” button.

Expected Results: Drone halts and feedback from autonomous command stops.

Actual Result: Drone halts and feedback from autonomous command stops.

Status (Fail/Pass): Pass

Test Suite ID 06

**Purpose:** To test the functionality of use case: **Navigate List of Waypoints**

**Test case 1:**

Purpose: Determining appropriate movement speeds returned by Navigator object.

Pre-condition:

1. Drone is turned on and connected
2. Selection mode is “Waypoint”.

Input:

1. A single marker is placed.
2. “Start Route” is clicked.

Expected Results: Navigator object returns an appropriate list of speeds that would be given to

the Drone’s “move” function.

Actual Result: Forward and sideways speeds appear to be random and don’t correlate with

marked waypoint.

Status (Fail/Pass): Fail

**Test case 2:**

Purpose: Determining appropriate movement speeds returned by Navigator object.

Pre-condition:

1. Drone is turned on and connected
2. Selection mode is “Waypoint”.

Input:

1. A single marker is placed.
2. “Start Route” is clicked.

Expected Results: Navigator object returns an appropriate list of speeds that would be given to

the Drone’s “move” function.

Actual Result: Forward and sideways speeds appear to be random and don’t correlate with

marked waypoint.

Status (Fail/Pass): Fail

**Test case 3:**

Purpose: Determining appropriate movement speeds returned by Navigator object.

Pre-condition:

1. Drone is turned on and connected
2. Selection mode is “Waypoint”.

Input:

1. A single marker is placed.
2. “Start Route” is clicked.

Expected Results: Navigator object returns an appropriate list of speeds that would be given to

the Drone’s “move” function.

Actual Result: Appropriate speeds given for selected waypoint.

Status (Fail/Pass): Pass

**Glossary**

Parrot AR.Drone 2.0 - Quadcopter drone used for Agricultural Robotics 3.0

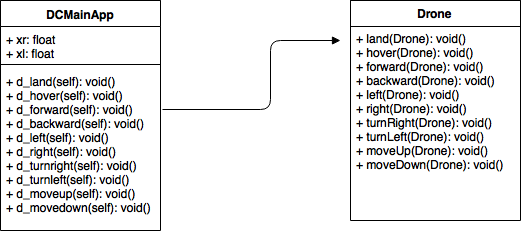
PS\_Drone - Parrot AR.Drone 2.0 API written in Python

Quadcopter - Helicopter with lift and propulsion from four rotors.

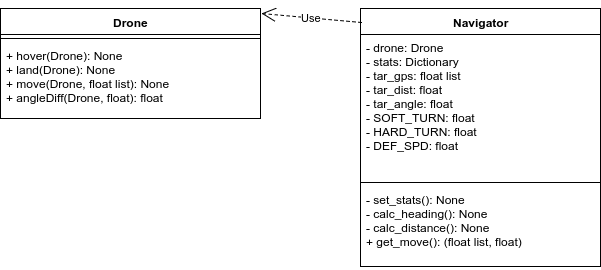
**Appendix**

**Appendix A - UML Diagrams**

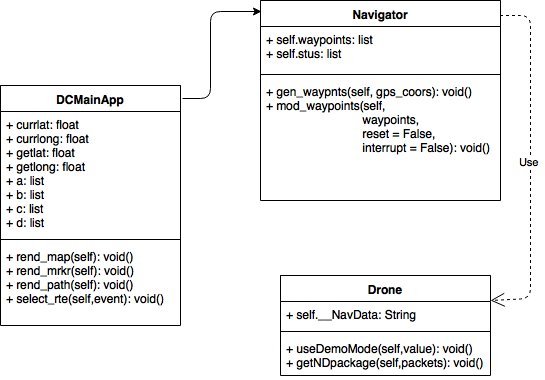
**Class Diagram Use Case #264**



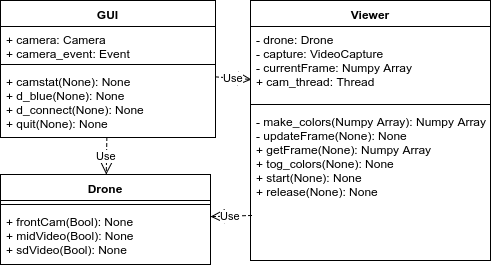
**Class Diagram Use Case #265**



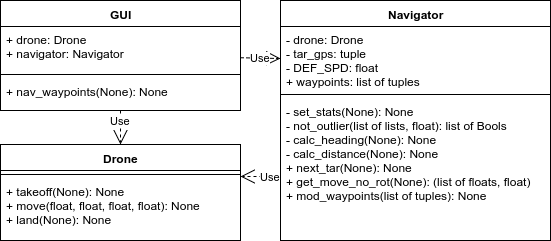
**Class Diagram Use Case #289**



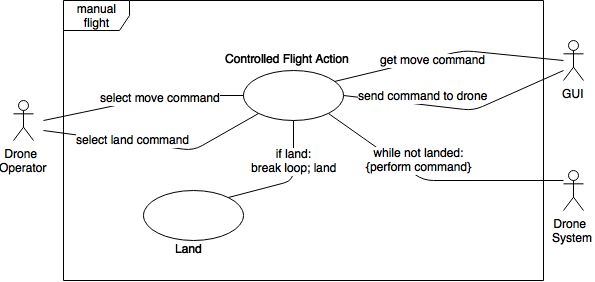
**Class Diagram Use Case #290**



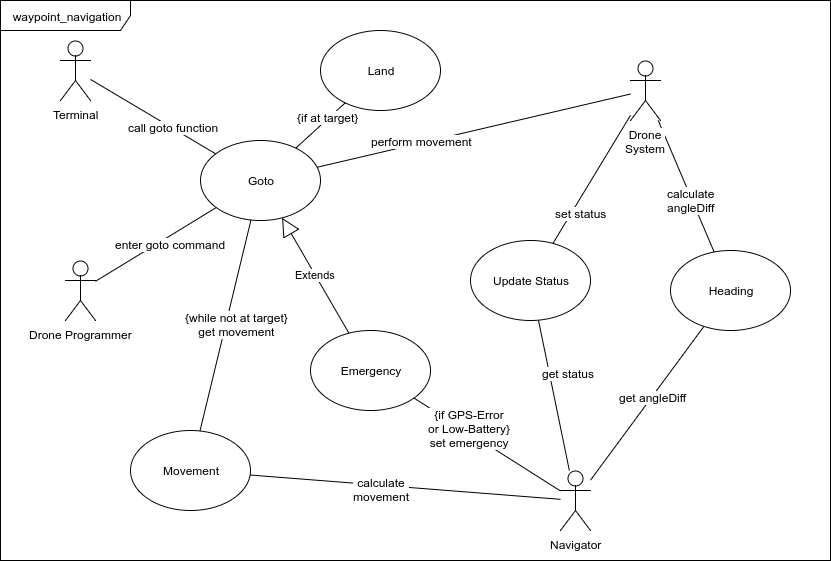
**Class Diagram #292**



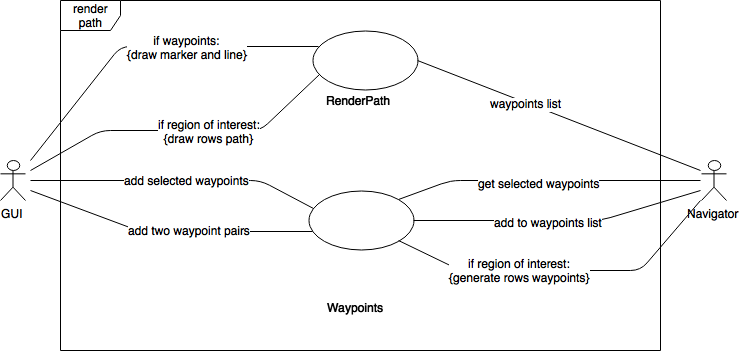
**Use Case Diagram #264**



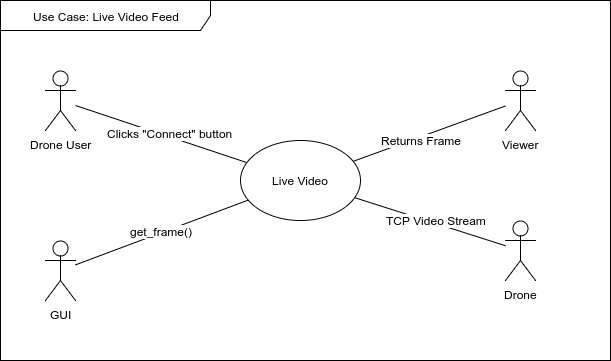
**Use Case Diagram #265**



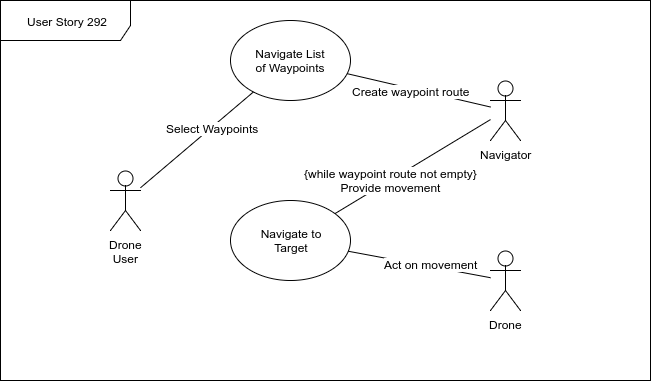
**Use Case Diagram #289**



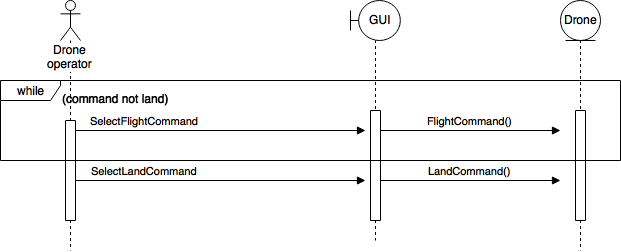
**Use Case Diagram #290**



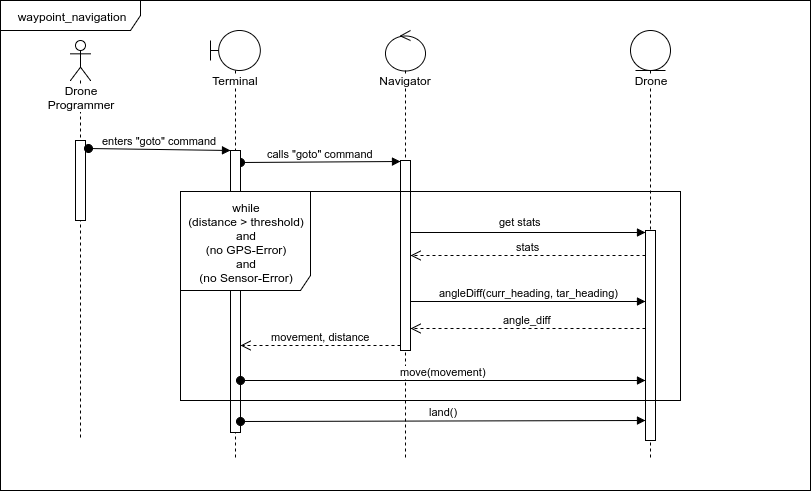
**Use Case Diagram #292**



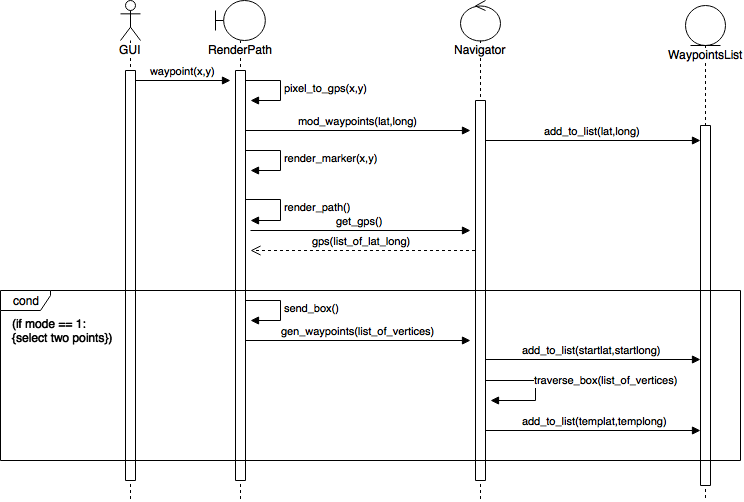
**Sequence Diagram Use Case #264**



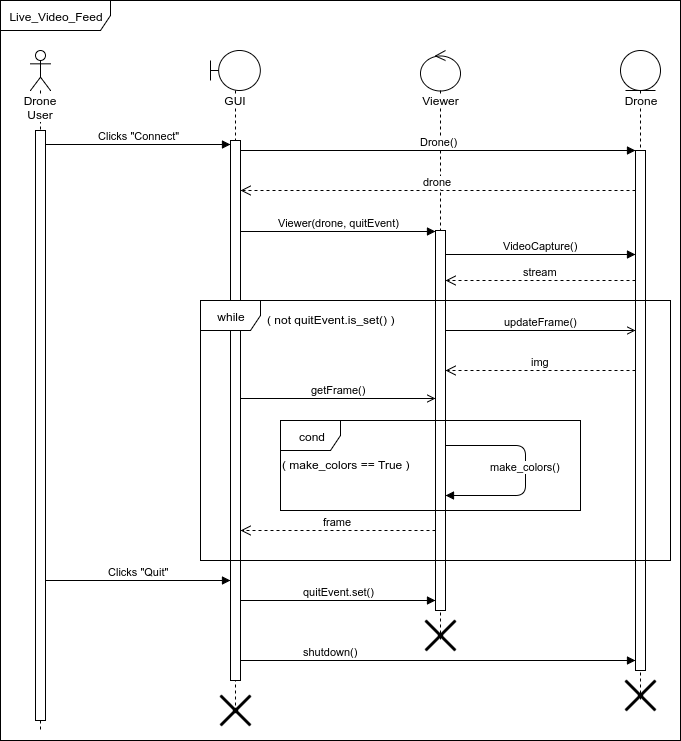
**Sequence Diagram Use Case #265**



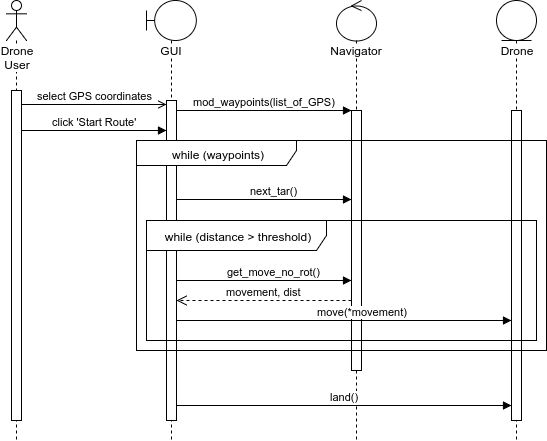
**Sequence Diagram Use Case #289**



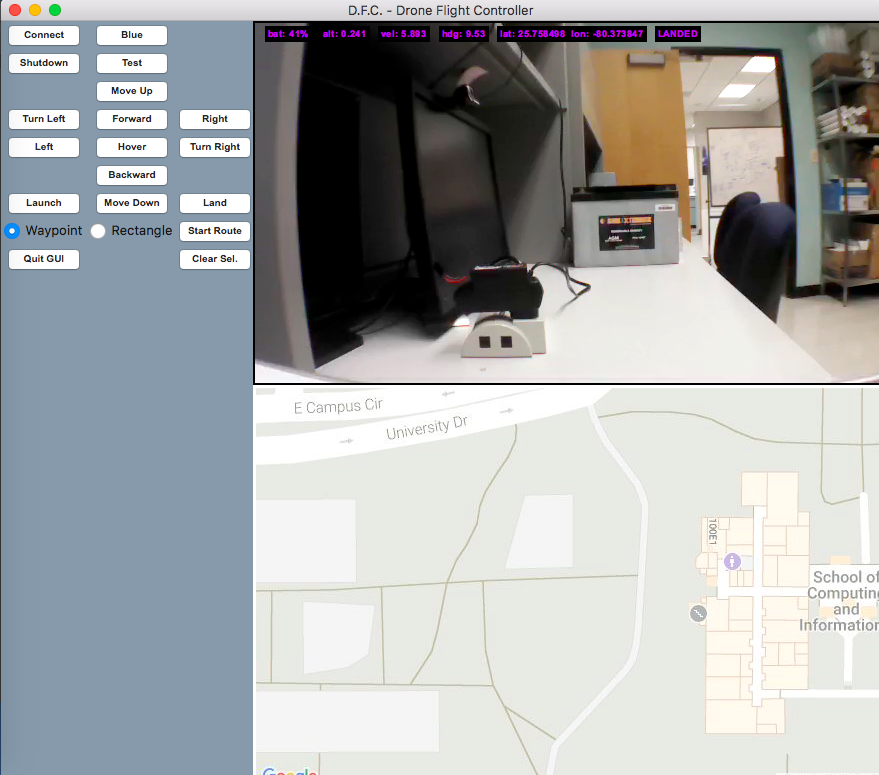
**Sequence Diagram Use Case #290**

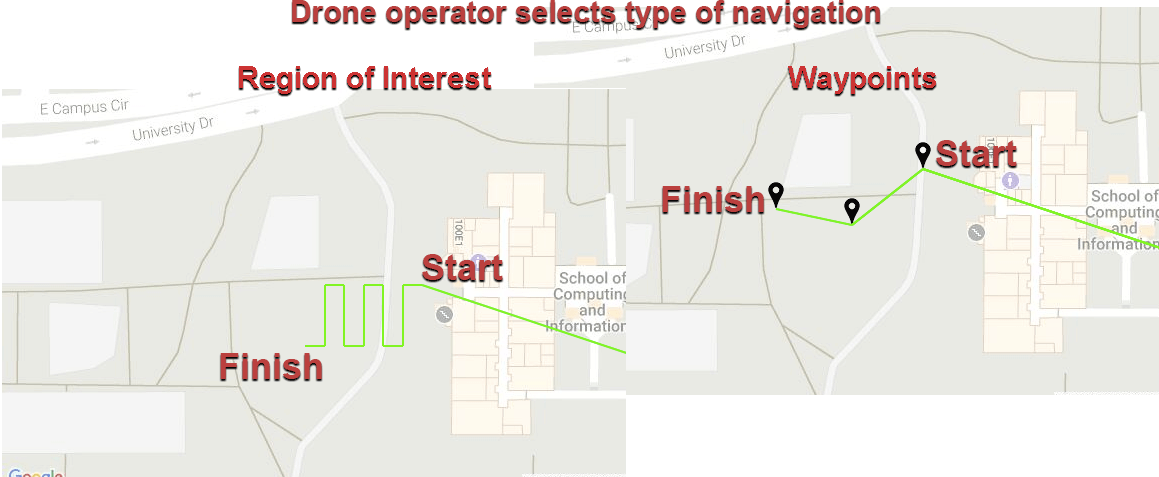


**Sequence Diagram Use Case #292**



**Appendix B - User Interface Design**





**Appendix C - Sprint Review Reports**

**Sprint 1**

Attendees: Sean Monroe, Christian Silva, Tauhidul Alam, Leonardo Bobadilla

Start time: 05/30/2017 2:30:00 PM

End time: 05/30/2017 2:40:00 PM

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

* #262 - GPS Coordinate and Sensor Data Acquisition
* #272 - Verify Equipment
* #273 - Research Python GUI

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.

* #264 - Flight Controller
* #265 - Drone Path Navigation

**Sprint 2**

Attendees: Sean Monroe, Christian Silva, Tauhidul Alam, Leonardo Bobadilla

Start time: 06/12/2017 8:20:00 PM

End time: 06/12/2017 8:30:00 PM

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

* #263 - Drone Commands
* #264 - Flight Controller
* #265 - Drone Path Navigation

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.

* None

**Sprint 3**

Attendees: Christian Silva, Sean Monroe, Leonardo Bobadilla, Tauhidul Alam

Start time: 06/26/2017 2:30:00 PM

End time: 06/26/2017 2:50:00 PM

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

* #289 - GPS Map
* #290 - Interactive Drone Targeting
* #292 - Path Determination

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.

* None rejected for future sprint, but the following are unfinished and require more work:
  + #290 - Interactive Drone Targeting
  + #292 - Path Determination

**Sprint 4**

Attendees: Christian Silva, Sean Monroe, Leonardo Bobadilla, Tauhidul Alam

Start time: 07/17/2017 6::00 PM

End time: 07/17/2017 6::00 PM

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

* User stories were incomplete, and have one task each remaining.

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.

* #290 - Integrate Sensor Data into Controls
* #292 - Multiple-Waypoint Navigation

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

* **USER MANUALS**
  + Python Drone Flight Controller use requires, at minimum, the software detailed in the installation/maintenance section of Appendix D.
  + Remove drone from box and connect drone battery.
  + Allow drone to complete startup (all drone indicator led’s green).

* + After creating a local repository, cloned from Github, cd to .../Agricultural-Robotics-3.0/Code/Production.
  + Run python AgriDrone.py (GUI window will display).
  + In the GUI press connect button (currently the connection startup messages output to the command line interface). Connection takes a few seconds. If connection freezes, in command line interface, type **CTRL + C** and then press quit in the GUI.
  + After connection is made GUI displays live camera with drone information (i.e. battery status, altitude, latitude and longitude, velocity and drone state (“flying,” “hovering” or “landed”)) and live drone map positioning.

**\*NOTE\* Testing of flight control buttons should be performed outdoors.**

* + Test drones computer vision, shape recognition, by toggling the “*blue*” button (e.g. currently computer vision detects bluish hues, enclosing objects within a convex rectangle).
  + Test drones live map positioning by simply walking the drone within the physical location of the map image; system using drone GUI interface must be within drone wifi-signals’ reception. **Recommended pre-flight to insure adequate GPS reception.**
  + Test drones manual flight controller (**Only Perform Outdoors)**. Place drone on surface where propellers are unobstructed. Press “*launch*” button. The drone will begin in stable hover; initial hover is approximately 1-meter from surface.
    - ***Manual Flight Control Button Functionality***
      * “Hover” - As mentioned this is the drones post launch state. However, hover command, at altitudes greater than 1-meter from surface, result in the same characteristic of 1-meter hover. This similarity is present at the desired altitude.
      * “Forward” - This command moves the drone positively on the y-axis.
      * “Backward” - This command moves the drone negatively on the y-axis.
      * “MoveUp” - This command moves the drone positively on the z-axis.
      * “MoveDown” - This command moves the drone negatively on the z-axis.
      * “Right” - This command moves the drone positively on the x-axis.
      * “TurnRight” - this command rotates the drone to the right using the z-axis frame of reference.
      * “Left” - This command moves the drone negatively on the x-axis.
      * “TurnLeft” - this command rotates the drone to the left using the z-axis frame of reference.
      * “Land” - This command lands the drone.
      * “Shutdown” - This command performs two consecutive actions, “land” followed by disconnection from drone wifi.
    - ***Autonomous Flight Control***
      * GUI flight controller has two options, “**Waypoint**” or “**Region of Interest**”. Select one of the navigation options.
      * Waypoint navigation:
        + Select point on the map (multiple points may be selected). Selecting point/points renders the drone flight path with green edges from each vertex.
        + Select “clear route” button if chosen route is unacceptable. Otherwise selecting “**Region of Interest**” will automatically clear the current route.
        + Toggle “start route” button to execute the current flight route.
      * Region of Interest navigation:
        + Select two points on the map. After selection, a path will be rendered displaying the drone flight route within containment region. Location of selected points on the map may result in North/South or East/West row pathing (vice-versa applies to both row pathing).
        + Select “clear route” button if chosen route is unacceptable. Otherwise selecting “**Waypoints**” will automatically clear the current route.
        + Toggle “start route” button to execute the current flight route.
    - ***Exit GUI***
      * Press “Quit GUI” button to exit application.
* **INSTALLATION/MAINTENANCE**
  + **Python 2.7 (Mac OS X)**
    - Check for Python 2.7. Open terminal window and run python --version

The terminal should display Python 2.7. If not installed proceed with

installation.

* + - Install python. First install C-compiler xcode, if already installed move onto homebrew installation. Open terminal window and type:

xcode-select --install

* + - Install homebrew package manager, if already installed move onto python 2 installation. In terminal type:

/usr/bin/ruby -e "$(curl -fsSL

https://raw.githubusercontent.com/Homebrew/install/master/install)"

* + - Install python. In terminal type: brew install python
  + **Python Libs and Modules**
    - Check if pip is installed. Open a terminal window and run pip --version
    - If pip is not installed proceed with installation, otherwise move onto required module installation.
    - Install pip. In terminal run:

$ curl -O http://python-distribute.org/distribute\_setup.py  
 $ python distribute\_setup.py  
 $ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py  
 $ python get-pip.py

* + - Ensure pip is up to date. Run pip install -U pip setuptools
    - Install required modules. Run pip install Tkinter Pillow numpy scipy
  + **OpenCV3**
    - Download opencv 3 IOS pack from http://opencv.org/releases.html (3.2.0) as zip file and unpack.
    - Open terminal and cd to opencv build folder cd ../opencv-3.2.0/build/
    - Run: cmake \

-D CMAKE\_BUILD\_TYPE=Release \

-D PYTHON2\_EXECUTABLE=/usr/bin/python2.7 \

-D PYTHON\_INCLUDE\_DIR=/usr/include/python2.7 \

-D PYTHON\_LIBRARY=/usr/lib/python2.7 \

-D PYTHON2\_NUMPY\_INCLUDE\_DIRS=/usr/local/Cellar/

numpy/1.13.0/lib/python2.7/site-packages/numpy/core/include \

-D WITH\_EIGEN=OFF \

-D WITH\_1394=OFF \

-D CMAKE\_INSTALL\_PREFIX=/usr/local \

-D INSTALL\_C\_EXAMPLES=OFF ..

**\*Note PYTHON2\_NUMPY\_INCLUDE\_DIRS** this is the pip install numpy

location. Check this directory to make sure it is correct.

* + - Now run: make
    - Next run: make install
* **SHORTCOMINGS/WISHLIST**
  + Time restrictions hampered fleshing out a Pilot object to fully decompose the PS\_Drone module, separating movement activities from it.
  + Weather severely restricted flight tests; May and June were particularly stormy, and mosquitos made most tests quickly done and lightly documented.
  + Reliable documentation of hobbyists was restricted to opaque Node interfaces and the ArduDrone family of programming; an active Python community in Parrot drones would have been welcome help.

References

Parrot AR.Drone 2.0 API:

Philipp de Graaff, J., “PS-Drone Programming a Parrot AR.Drone 2.0 with Python- The Easy Way,” http://www.playsheep.de/drone, 2014

Pip Installation:

PyLadies, “Getting your mac ready for Python programming,”

http://www.pyladies.com/blog/Get-Your-Mac-Ready-for-Python-Programming/

Python Installation Guide:

Kenneth Reitz, “Installing Python 2 on Mac OS X”, http://python-guide-pt-br.readthedocs.io/en/latest/starting/install/osx/, 2016