Unmanned Aircraft System Operating Autonomously in GPS Denied Environments

MAE 435: Project Design and Management II

Old Dominion University

December 2nd, 2019

Advisors: Dr. Thomas Alberts, Dr. Ahmed Mekky

Team: Nicholas Darden, Vivian Fakharizadeh, Zavia Harris, Brendan Regan, Richard Jones, Trent Jones, Joshua Lee, Ryan McLaughlin, Rushal Patel, Quinton Vehon

Table of Contents

[Abstract 4](#_Toc26218920)

[Introduction 5](#_Toc26218921)

[Methods 6](#_Toc26218922)

[Completed Methods 6](#_Toc26218923)

[LiDAR: 6](#_Toc26218924)

[Dead Reckoning: 7](#_Toc26218925)

[Visual Odometry: 10](#_Toc26218926)

[GPS: 13](#_Toc26218927)

[DroneKit: 15](#_Toc26218928)

[Results 16](#_Toc26218929)

[Results Summary: 16](#_Toc26218930)

[Analysis: 19](#_Toc26218931)

[Discussion 22](#_Toc26218932)

[Purpose Summary: 22](#_Toc26218933)

[Limitations: 22](#_Toc26218934)

[Proposed Methods: 22](#_Toc26218935)

[LiDAR: 22](#_Toc26218936)

[Dead Reckoning: 23](#_Toc26218937)

[Visual Odometry: 24](#_Toc26218938)

[GPS: 24](#_Toc26218939)

[DroneKit: 25](#_Toc26218940)

[Conclusion 25](#_Toc26218941)

[References 27](#_Toc26218942)

[Appendices 30](#_Toc26218943)

[Appendix A 30](#_Toc26218944)

[Appendix B 31](#_Toc26218945)

[Appendix C: 38](#_Toc26218946)

[Appendix D: 41](#_Toc26218947)

[Appendix E 53](#_Toc26218948)

Table of Figures

[Figure 1:The two-step process of the Kalman Filter 9](#_Toc26215650)

[Figure 2: ArUco Markers 11](#_Toc26215651)

[Figure 3:Raspberry Pi 3B+ 13](#_Toc26215652)

[Figure 4: GPS mounted on Raspberry Pi 14](#_Toc26215653)

[Figure 5: LiDAR Sensor Experimental Data 17](#_Toc26215654)

[Figure 6:Raw IMU acceleration data for 1-D test in y-direction 18](#_Toc26215655)

[Figure 7: Example of testing the IMU at rest 18](#_Toc26215656)

[Figure 8: Closer look at accelerometer bias in X and Y direction. 18](#_Toc26215657)

[Figure 9: With flipud function 19](#_Toc26215658)

[Figure 10: Without flipud function 19](#_Toc26215659)

[Figure 11:ArUco Marker Identification. 20](#_Toc26215660)

[Figure 12: ArUco Marker showing three axes using calibration code. 21](#_Toc26215661)

# Abstract

Unmanned Aircraft Systems (UAS) are currently used in a wide scope of applications, ranging from recreation and hobbyists, the filming industry, military, and even geographical mapping and surveying in construction and exploration. Most systems usually rely on Global Positioning Systems (GPS) and Inertial Measurement Units (IMU) for navigation and altitude controls, which in certain applications, such as military search and rescue missions and cargo transportation, are extremely vulnerable to being hacked and traced. To remedy and minimize these operational and security threats, the United States Navy has contracted Old Dominion University to develop a UAS that is capable of navigating to a predetermined location without the use of GPS in order to improve aircraft stealth in hostile environments. This project will provide alternative navigation systems, such as Light Detection and Ranging (LiDAR) and altimeters for altitude/obstacle detection, inertial measurement units for position estimation, visual odometry for landing zone detection, and DroneKit to allow the various systems to communicate. These sensors, which have all been tested, have been found to be accurate within desired parameters and upon integration, will exclude the need for GPS navigation during flight.

# Introduction

The current and primary methods the Navy utilizes in transporting equipment to and from ships are Vertical Replenishment (VERTREP) (using helicopters to deliver munitions to ships on the open ocean), small vessel transport, or the return of the ship itself to shore in a time consuming, expensive, and dangerous process. Therefore, a replenishment method that possesses high system security [1], low operational cost [2], and low system complexity [3] is needed for the transportation of payloads across large distances to vessels at sea.

VERTREP UAS capabilities require an aircraft to possess range determination, onboard guidance without GPS, and landing zone recognition. These can be accounted for using LiDAR, barometric altimeters, and visual optics, which are capable of providing obstacle avoidance, altitude adjustment, and landing execution by utilizing a Raspberry Pi micro-computer running multiple integrated software. Additionally, IMUs linked with magnetometers have allowed UASs to maintain their correct course without manual controls and visual optics. DroneKit can be used to tie these systems together and allow them to communicate.Therefore, UASs are capable of autonomous navigation without GPS using these on-board sensors [4, 5]. Past missions have accumulated navigational errors when navigating over the ocean for 200 miles or more, resulting in lack of success [1, 4]. To provide a solution to this problem, the aforementioned array of sensors will be integrated into a Vertical Take-Off and Landing (VTOL) platform that will allow the UAS to accurately fly across a predetermined path without remote control or GPS guidance. Therefore, the primary objective of this project is to produce an inexpensive UAS design that will ensure the safety of offshore resupply missions for the U.S. Navy without the aid of GPS.

# Methods

The project team was initially divided into four subgroups to handle the four main components making up the UAS: LiDAR, Dead Reckoning, Visual Odometry, and GPS. This was done so that each system could be tested, optimized, and tuned to operate correctly as an individual unit before being integrated to work as a cohesive system. After each subsystem has been successfully optimized, the subgroups will be rejoined into one large integration team to combine the subcomponents into one system and troubleshoot any errors that arise. In the latter half of this project’s research phase, a small team was tasked with finding an appropriate Application Programming Interface (API) to allow for effective communication between the various systems for future iterations of this project. DroneKit was determined to be the most promising open source API available.

## Completed Methods

#### LiDAR:

The LiDAR group began by testing the SF11/C (LightWare Optoelectronics, Gauteng, South Africa) LiDAR sensor over still, turbulent, murky, and ocean water surface profiles to test the sensor’s accuracy over the various water conditions in which the UAS will be required to operate. Since its accuracy was within parameters, the SF11/C’s wires were then soldered to the connector wires in order of 5V, GND, RXD, TXD in connector holes 1,2,3, and 6 respectively in GPS Port 2 of the main PixHawk 2.1 (Hex, Sha Tin, Hong Kong) flight controller.

Next, Mission Planner (ArduPilot Dev Team) was used to set up flight parameters and record the data from the LiDAR that was interpreted by the PX4 Companion Software (Dronecode Project, San Francisco CA). The functionality of the LiDAR was then tested by lifting the VTOL UAS and comparing the recorded data with a predetermined measured height.

### Dead Reckoning:

The Dead Reckoning team was tasked with developing the position detection component of the UAS using dead reckoning methods (predicting current state from past measurements), an IMU, and Extended Kalman Filtering. This was done using an ADIS16475 IMU (Analog Devices, Norwood, MA), which was used to collect acceleration and angular velocity data about the x, y, and z axes, and a Raspberry Pi 3 B+ (Raspberry Pi, UK) micro-computer, which was linked to the IMU to read the output data through the 32-bit pinout.

***IMU Testing:***It was necessary to conduct testing of the IMU to determine the nature of the data it was collecting (data quality/accuracy, noise/interference). To collect the data, the IMU was attached to a file cabinet door with a rolling track that provided a stable mount, reduced noise, and a single axis of motion when pulled out 0.5 meters from its initial position. The recorded data was then run through a custom IMU position code on MatLab (The Mathworks, Natick, MA) (Appendix B.2-5), which uses Euler integration to find velocity and position values from the sets of acceleration and angular velocity data points. In order to convert the body frame velocities to inertial frame velocities, a rotation matrix, R, was created (Appendix A; Eq.1), which computed the Euler angles (yaw ψ, pitch θ, and roll φ) by equating each element in R with the corresponding element in the matrix product Rz(ψ)Ry(θ)Rx(φ). This value was then used to find the inertial frame velocities (Eq. 2).

(2)

A similar process was used for the angular rates, where transformation matrices (Appendix A; Eq. 3) were constructed to convert body frame angular rates to inertial frame angular rates (Eq.3).

(3)

***Extended Kalman Filter:*** During testing, the IMU produced significant bias and noise measurements that needed to be corrected prior to integration so that the position estimates could be calculated accurately. Therefore, extensive research was done on the Kalman Filter optimal linear estimation algorithm to eliminate this noise. This algorithm consists of a two-step process that iteratively predicts, optimizes, and updates the current state data in real time. In the prediction step, the Kalman Filter produces estimates of the current state as well as the uncertainty measurements in that state estimate. The next time step measurement is then observed along with the current state estimate and the estimates are updated, with more weight being given to the estimate with less uncertainty.

Due to the fact that the UAS operates in nonlinear paths, it was necessary to research an Extended Kalman filter (EKF) which is the nonlinear version of the Kalman Filter. The EKF linearizes the non-linear function around the mean of the current state estimate and the local linearization of each time step (Figure 1). Multiple EKF MATLAB codes were then compiled so that they could be modified to fit the quadrotor model.

A screenshot of a cell phone

Description automatically generated

Figure 1:The two-step process of the Kalman Filter

An EKF code that had been used by a CubeSat team was chosen as the best candidate to modify for the UAS. Project advisor, Dr. Ahmed Mekky, worked closely with the Dead Reckoning team to modify the code and develop state and covariance matrices modeling quadrotor flight that could be implemented into the modified code (Appendix B.1). This resulted in a functioning, though imprecise, Extended Kalman Filter that needs to be tuned by adjusting the covariance parameters.

***Bias Correction:*** In addition to noise, the IMU was also found to be reading acceleration in the x and y axis when at rest, indicating bias in these directions (Figure 10). This was initially reduced by subtracting the average of the biases across a 10 second static test of the IMU (IMU at rest on flat surface) from the original biased readings.

Another proposed solution was conducting an IMU recalibration that consisted of running the IMU in 12 different orientations in accordance with (Zhang, Reindl, & Höflinger, 2014). This calibration produces scaling factors that are to be implemented into the IMU EKF and position code to correct the bias.

### Visual Odometry:

The visual odometry team was tasked with developing the landing procedures and landing zone identification system of the UAS. This system utilizes a Raspberry Pi Camera Module V2-8 Megapixel 1080p (Raspberry Pi, UK) linked with a Pixhawk flight controller and Raspberry PI to identify ArUco markers located at the landing zone.

***OpenCV, ArUco and Microsoft Visual Studio C++ Setup:***The first step in enabling the UAS to land on a specific symbol (Figure 2) was to attempt the process in a computer simulation. Therefore, an object detection script was obtained from an open-source database of object tracking and camera-based algorithms: OpenCV version 4.1.0 (Open Source Computer Vision Library). This script was run through Microsoft Visual Studio C++ 2017 (Microsoft, Redmond, WA) on a laptop [1]. In order to allow the algorithms and commands from OpenCV to work, the directories were connected by entering them into the properties of the Visual Studio project. This connection allowed Visual Studio to pull specific commands from the OpenCV database and use them within the script [1].

***ArUco Setup and Camera Calibration:***In order for the UAS to detect the predetermined landing symbols, another open-source database called ArUco (GitHub Inc., San Francisco, CA) was needed. This resource was used in parallel with OpenCV [2,3] to create unique black and white markers containing identifying functions recognizable by the camera. These markers allow the UAS to estimate its distance and orientation with respect to the marker (Figure 2).



Figure 2: ArUco Markers

While testing the system, it was discovered that calibrations and relative distance testing had to be conducted using a checkerboard marker to accurately read the markers using the camera. The calibration involved holding up a checkerboard marker, tilting it, and recording the data points (described in more detail below).

***Integration of Open Source Software:***Once simulated testing of the ArUco system had been completed, the OpenCV and ArUco scripts were downloaded onto a Raspberry Pi to begin testing the camera’s ability to detect the ArUco markers and estimate their positions in real life. After this, the Raspberry Pi was connected to the Pixhawk, allowing the Raspberry Pi to communicate with the Pixhawk, determine the position of the UAS, and allow the UAS to autonomously land on the markers. It was found that the installation process would be different on the laptop and the code would need to be written in Python (Python Software Foundation, Wilmington, DE) instead of C++.

First, a Raspberry Pi (Figure 3) was set up with Raspbian, which is an operating system for the Raspberry Pi. In order to allow convenient use of the Raspberry Pi, two pieces of open source software were used without the need for a separate computer station. The first was PuTTY (Cambridge, England), a server software that allows the user to access the terminal of the Raspberry Pi from a laptop screen [9]. The second was VNC Viewer (Cambridge, UK), a virtual desktop software that connects to the Raspberry Pi via the internet and allows the user to use the laptop screen as the display for the Raspberry Pi itself [11]. Next, the Raspberry Pi was set up to download tools, dependencies, libraries, and packages to accurately setup ArUco and OpenCV [4]. A virtual environment was then created so that different versions of Python could be run on the Raspberry Pi. The OpenCV and ArUco libraries were downloaded and compiled in this environment. To make sure OpenCV was installed on the Raspberry Pi, the terminal was asked to detect which version of OpenCV had been downloaded. Once the version was confirmed, the Raspberry Pi Camera Module V2-8 Megapixel 1080p was enabled and a test code using Python was compiled to make sure the camera and OpenCV were running correctly [5].

Once this was complete, the camera on the Raspberry Pi had to be calibrated to accurately detect and estimate the three-dimensional coordinates of the markers relative to the camera [6]. This was done by taking several pictures of a chessboard marker at various angles. After the calibration was finished, code was created that allowed the camera to identify the markers and then estimate the position of those markers in x, y, and z coordinates relative to the UAS [7].

After this, the Raspberry Pi was connected to the Pixhawk, allowing the Raspberry Pi to communicate with the Pixhawk, determine the position of the UAS, and allow the UAS to autonomously land on the markers. When setting up the connection from the PixHawk to the Raspberry Pi, it was realized that Dronekit, which is a library used for communicating with drones, needed to be installed on the Raspberry Pi. Because Dronekit only works with Python 2.7, OpenCV was reinstalled with Python 2.7 instead of Python 3. The PixHawk and Raspberry Pi were then connected via the UART pins which was a faster method to receive and output data between the PixHawk and Raspberry Pi. Then, a code was developed that allows the distance and position of the marker is estimated when the camera on the Raspberry Pi locates and locks onto the marker (Appendix D.3). When the camera sees the marker at the set distance, the PixHawk will initiate landing mode.



Figure 3:Raspberry Pi 3B+

### GPS:

The purpose of the GPS team was to determine the output data format of the GPS and convert the IMU data into the same format. This was necessary because the PixHawk flight controller is designed to operate with a specific GPS data format. Therefore, it was necessary to determine what this format was and how to convert the IMU data to serve as a “spoof” GPS signal.

The Berry GPS-IMU V3 (Berry GPS-IMU V3, Ozzmaker, Australia) was chosen for the final GPS integration stage of the project. This model of GPS was chosen due to its faster connection rate and better mounting position on Raspberry Pi compared to other modules. This GPS mounts on top of the Raspberry Pi (Figure 4) using female headers soldered onto the PCB (print circuit board) of the GPS. This setup removes the need of using wires to connect to the GPIO pins of the Raspberry Pi.

A circuit board

Description automatically generated

Figure 4: GPS mounted on Raspberry Pi

Once the GPS module was connected to the Raspberry Pi, GPS libraries were required to read data on the Raspberry Pi. Minicom, gpsd, and gpsmon were the Python functions chosen to help test GPS data, communicate with the GPS module and convert location data into a readable format. The minicom library was also used to test the raw GPS data.

After it was determined that the test-GPS could be connected and transmit data accurately, the Berry GPS V3 (Berry GPS-IMU, Ozzmaker, Australia) was used during the final integration stage of the project. Data from the GPS was transmitted in NMEA (National Marine Electronics Association) sentences consisting of latitude and longitudinal coordinates. Gpsd client was run as a background process to receive data from the GPS and transmit it to gpsmon in a readable format. Gpsmon or cgps are two commands that reformat readable NMEA sentences into a table consisting of latitude, longitude, altitude, and speed data. To install these libraries, a command was written into the terminal window of the Raspberry Pi that downloaded and installed the files from the internet. As such, an internet connection was required for Raspberry Pi to install any kind of library onto the system. For example, typing “sudo apt-get install minicom,” in the terminal window prompted the installation of the minicom library onto the system. To test the accuracy of the GPS, latitude and longitude coordinates were checked in a web browser to compare the measured location. An open source code (Appendix C.1) was used to extract only necessary information from the NMEA strings into a readable text file.

### DroneKit:

DroneKit allows for flight control commands to be sent simply, accurately, and effectively from the Raspberry Pi 3 B+ (Raspberry Pi, UK) companion computer to the PixHawk onboard flight controller. It accomplishes this by accessing MAVLink code, the API developed for commercial UAS flight control, and organizes it into callbacks that are defined in the syntax of DroneKit’s library. With DroneKit, a script can be developed that manipulates the data being sent to the PixHawk Flight controller, allowing the UAS’s GPS information to be captured, manipulated, and sent to the UAS.

***Flight Control Scripting:***The Flight Control team was tasked with researching the viability of using an API to allow the Raspberry Pi to inter-operate and manipulate the flight data sent from the sensory array attached to it. DroneKit was determined to be the most promising open source API available and establishing a infrastructure for implementing a flight control script would be necessary for future iterations of this multi-year project. DroneKit allows for flight control commands to be sent simply, accurately, and effectively from the Raspberry Pi companion computer to the PixHawk onboard flight controller. DoneKit accomplishes this by accessing MAVLink code, the API developed for commercial UAS flight control, and organizes it into callbacks that are defined in the syntax of DroneKit’s library. Using this, a script can be developed that manipulates the data being sent to the PixHawk Flight controller, allowing the UAS’s GPS information to be captured, manipulated, and sent to the UAS.

***Planning/Setup/Testing:***With Dronekit as the flight control API, a programing block diagram was developed to visualize how the flight control script would be written (Appendix F.1). In order to use DroneKit on the Raspberry Pi, both DroneKit and MAVLink’s open source libraries had to be installed. Additionally, Python 2.7, DroneKit, Mavlink, and DroneKit-SITL were installed on the team’s computers to allow for simultaneous testing in a closed loop environment.

The first test was a systems verification test. The Camera and GPS sensors were attached to the UAS and tested independently to see if they could communicate with the Raspberry Pi using MAVLink. The second test was a system verifications test in which the Raspberry Pi was attached to the PixHawk flight controller and a system diagnostic was requested using MavLink. The third test was a simulated run of DroneKit preforming a system check on DroneKit-SITL, and the fourth test was a scripting verification test where the DroneKit Script used in test 3 was uploaded to the Raspberry Pi. The Pi then ran the systems check script.

# Results

## Results Summary:

Altitude data readings from the LiDAR over varying water surface profiles showed a maximum of 3.1% error (Figure 5) when over turbulent water. With the readings being within the desired parameters of accuracy of 6 centimeters, the SF11/C sensor was made the primary altitude measurement device required for the final product. For the Dead Reckoning team, the IMU data still showed acceleration in the x-direction (Figure 7-8) even though it was restricted and only moved in the y-direction. The inherent bias in the device was confirmed by running the IMU while stationary, producing the results in Figures 7-8. Additionally, the EKF was producing drastically different plots based on the inclusion/exclusion of the ‘flipud’ function (Figures 9-10). In Visual Odometry, the optic odometry software successfully identified the pre-generated markers, measuring the rotation and translation vector after running the ArUco and camera calibration code. The GPS team successfully determined the data input type needed to “spoof” the GPS signal, and the result of the multiple levels of DroneKit tests confirmed that a script written using the DroneKit API would be able to provide the capabilities required to perform the primary mission objective. These results are discussed in detail in the analysis below.

|  |  |  |  |
| --- | --- | --- | --- |
| Measured height (m) | Actual height (m) | Water | Error (%) |
| 3.3 | 3.20 | Turbulent | 3.1 |
| 3.0 | 3.02 | Still | 0.7 |
| 3 | 3.05 | Murky | 1.6 |
| 3.8 | 3.79 | Ocean | 0.3 |

Figure 5: LiDAR Sensor Experimental Data

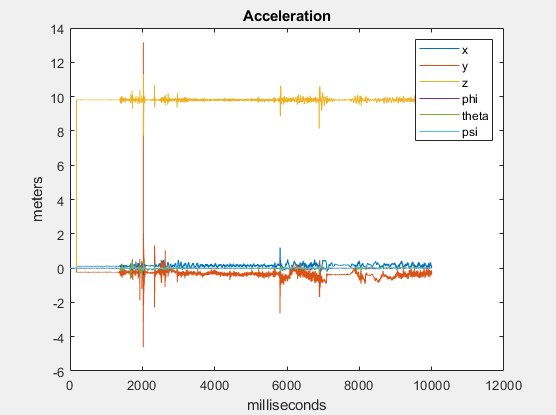


Figure 6:Raw IMU acceleration data for 1-D test in y-direction

A screenshot of a social media post

Description automatically generated

Figure 7: Example of testing the IMU at rest

A screenshot of a cell phone

Description automatically generated

Figure 8: Closer look at accelerometer bias in X and Y direction.



Figure 9: With flipud function

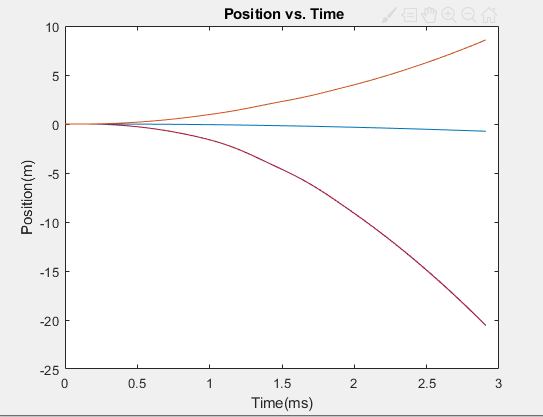


Figure 10: Without flipud function

## Analysis:

***LiDAR:***The results indicate that the PX4 software requires further post-processing due to the current inaccuracies of the sensors. A higher powered SF11/C LIDAR unit showed major improvement in the accuracy of altitude estimation with under 4% error, which is accurate enough to be the primary altitude measurement device for the UAS.

***Dead Reckoning:***After running the static test data through the position code, detected movement was reduced to zero the first three seconds of the testing period, but error accumulation from subtracting averages still resulted in detected movement. From 1-3 seconds, the position reading was zero for all three axes, which was correct given the IMU was at rest, but by the end of the test, the code indicated a significant movement of 0.05m. Additionally, the “flipud” function drastically changed the output plot of the EKF, stressing the importance of determining the reason for its implementation into the original code.

***Visual Odometry:***The calibration performed on the laptop using the C++ code (Appendix D.1) and laptop camera identified a pre-generated marker within its library database (Figure 11). The green, red, and blue lines indicate the x, y, and z axes, respectively and the resulting data shows the rotation and translation vectors relative to the camera.

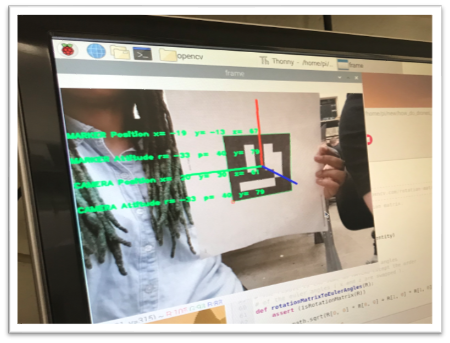


Figure 11:ArUco Marker Identification.

The Raspberry Pi Camera was also capable of identifying a pre-generated marker (Figure 12). The green, red, and blue lines indicate the x, y, and z axes respectively. The data shows the rotation and translation vectors relative to the camera [Table 2, Appendix A].

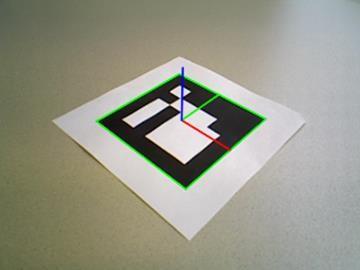


Figure 12: ArUco Marker showing three axes using calibration code.

***GPS:***Comparing latitude and longitude coordinates from gpsmon and Google Maps proved that the GPS module was outputting precise data. Similar types of test were also conducted when the GPS module was being moved from Kaufman Hall to Perry Library. In this case, the continuous stream of GPS data when moving was monitored to check its accuracy. These tests concluded that the Berry GPS-IMU V3 was capable of producing precise data while stationary and moving (Appendix C.5).

Multiple tests were conducted to determine the accuracy of GPS data. The gpsmon command was used to screen GPS data on the Raspberry Pi terminal, and Google Maps was used to check its accuracy. The module was placed near the back entrance of Kaufman Hall to let it acquire satellite connection and produce GPS data. Once the data was acquired from gpsmon, its was compared to latitude and longitude data for Kaufman Hall from Google Maps (Appendix C.4).

***DroneKit:***Test 1: All systems were able to function appropriately using the Pi. Test 2: The UAS successfully conducted a systems diagnostic routine. Test 3: DroneKit-SITL proved effective and displayed realistic results. Test 4: The UAS successfully conducted a systems diagnostic routine using drone kit (Appendix F.2).

# Discussion

## Purpose Summary:

The overall objective of this project is to develop an autonomous, GPS denied UAS for VERTREP Naval operations. The current shorter-term objective is to enable the UAS to fly across 300m of Kaufman Mall without the assistance of a GPS signal. Once that objective is reached, the final test for this iteration of the project is to fly the Fixed-Wing VTOL UAS 5 miles over water and return autonomously to land on a pre-generated ArUco marker.

## Limitations:

The primary limitation of this project has been the team members’ lack of knowledge with components and software with respect to UAS technology, such as programming and control systems. This resulted in increased research time for each of the teams. Additional limitations were found in the malfunctioning IMU, which severely slowed the progress of the Dead Reckoning team. The GPS and Visual Odometry teams experienced challenges in software compatibility between the Raspberry Pi, the GPS module, and the PixHawk. Many hardware components, such as modified connectors and wires, had to be constructed by hand which further limited the teams progress. These setbacks resulted in scope creep in which the semesters goals had to be reset and the scope shortened to fit within the semester timeline.

## Proposed Methods:

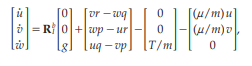
### LiDAR:

***LiDAR Mounting and Flight Testing:*** A lightweight 3D printed frame will be designed to fit the LiDAR to the UAS arm. After successfully integrating LiDAR SF11/C Sensor into the PixHawk 2.1 platform, additional testing will be performed over various water conditions. Since LiDAR has not been used to calculate altitude over a water surface before, this test can document the future use of LiDAR in long distance travel over the ocean.

### Dead Reckoning:

Future tasks on the Dead Reckoning team consist in further attempts to identify the cause of the bad IMU readings (equipment malfunction/hardware malfunction, or some other cause), completing the IMU calibration by running the IMU through the calibration orientations in accordance with (Zhang, Reindl, & Höflinger, 2014). The effectiveness/accuracy of the bias correction would then need to be determined, and the EKF will need to be tuned to optimize the position estimation through adjustment of the covariance matrix parameters “P” and “Q” (Appendix B.1) via trial and error. While or after resolving the IMU data collection issue, it is recommended that the team investigate the use of MATLAB function “flipud’ in the EKF code. This function, which was copied over from the custom position code that was passed on, flips the data top to bottom before it is processed through the program. It is currently unknown why this function was implemented, and it was not able to be resolved due to the inability to collect reliable data from the IMU that could be used to test how the data gets manipulated. One possible theory is that the IMU inputs data bottom to top, creating the necessity of flipping the data before processing it through code.

Additionally, if time allows, an augmented EKF (Eq. 6), which accounts for drag force as well as gravity and rotor thrust, should also be researched and implemented. The added drag components (the last term in Eq.6) is proportional to the body-fixed-frame velocity of the quadrotor.



(6)

### Visual Odometry:

***Raspberry Pi and Code Implementation:***In the future, the goal is to connect the Raspberry Pi to the PixHawk, which will allow the position of the UAS to be determined during flight. The Raspberry Pi will then be able to estimate the distance of the ArUco markers relative to the UAS, allowing the UAS to land on the marker [8]. Additionally, the Raspberry Pi Camera will need to undergo multiple phases of testing. The first will be a stationary test in which the Raspberry Pi Camera will be mounted in place and the ArUco marker moved along the 3D axis. This will record the position coordinates of the sensor with respect to the marker. The Raspberry Pi Camera will then need to be mounted to the custom 3D printed frames, one of which will have the camera positioned with the lens facing the ground, the other having the camera facing the ground at an angle of 35 degrees from the horizontal. This dual-angle setup will test whether the Raspberry Pi Camera is able to lock onto the ArUco markers from various positions.

### GPS:

***GPS to PixHawk:*** To allow communication between Raspberry Pi and PixHawk, a USB connector will need to be fashioned to send data from the “Telemetry 2” port on PixHawk to the USB port on Raspberry Pi. Mavproxy and Mavlink will also need to be installed on the Raspberry Pi, allowing the Raspberry Pi to send and receive data. Once the connection is established between the two modules, a Python script can then be used to command the Raspberry Pi to convert and transmit IMU and GPS data in place of the GPS signals to provide accurate data readings for the PixHawk.

***Creating Spoof GPS Signals:***Use of codes such as ‘pi@raspberrypi ~ $ screen /dev/serial0 9600’ and ‘pi@raspberrypi ~ $ gpspipe –r’ will need to be used in the future to extract the GPS data and store it into a log file on the Raspberry Pi. Additional future tasks include creating and utilizing a control system script that disables the GPS and appends the EKF code from the IMU and altitude data from LiDAR and send it to the GPS log over short time intervals during flight. Simultaneously, the Raspberry Pi will have to send these compiled ‘spoof’ GPS signals to the PixHawk Flight controller to adjust path movement.

### DroneKit:

***Flight Control Scripting:***Future iterations at of this project should continue by permanently integrating the sensory array to the UAS, determining what open source scripts are available to preform the objectives outlined in the Programming Block Diagram (Appendix F.1), develop tests to verify the capabilities of the open source codes independently, and develop a flight script that fulfills all the objectives outlined in the Programming Block Diagram, while integrating verified opensource code when possible. Additionally, verification of the flight control scripts functionality while on the ground should be conducted, as well as verification of functionality while on a mission in flight.

# Conclusion

The purpose of this project was to produce a Vertical Take-off and Landing Unmanned Aerial System capable of completing unguided autonomous VERTREP Naval operations. Therefore, a UAS flight system is being developed using a Light Detection and Ranging sensor, inertial measurement unit, and visual odometry system, with the end goal of producing a more efficient, safe, stealthy, and effective means of overseas cargo transfer to and from Navy vessels. Based on the research and testing conducted, LiDAR sensors are capable of serving as the altitude detection system of UAS. A rudimentary EKF has been developed to estimate the position of the UAS while GPS denied and, with some fine tuning and modification, will allow the vehicle to navigate accurately. Visual odometry successfully identified an ArUco marker, supporting the likelihood of the systems ability to serve as landing zone recognition for the vehicle. Lastly, the GPS team successfully determined the input data format required for the PixHawk flight controller and, by utilizing DroneKit, assured that all of the individual systems can be integrated and communicate data with each other. This system, when completed, will significantly reduce the possibility of hacking and tracking by eliminating the use of GPS navigation, allowing the warfighters to carry out their crucial and secret missions undetected, more efficiently, and more safely.

# References

[1] Braga, Jose R. G., et al. “Estimation of UAV Position Using LiDAR Images for Autonomous Navigation over the Ocean.” 2015 9th International Conference on Sensing Technology (ICST), 2015, doi:10.1109/icsenst.2015.7438508

[2] I. Sa, M. Kamel, M. Burri, M. Bloesch, R. Khanna, M. Popovic, J. Nieto, and R. Siegwart, “Build Your Own Visual-Inertial Drone: A Cost-Effective and Open-Source Autonomous Drone,” IEEE Robotics & Automation Magazine, vol. 25, no. 1, pp. 89–103, Dec. 2017.

[3] E. Fresk, G. Nikolakopoulos, and T. Gustafsson, “A Generalized Reduced-Complexity Inertial Navigation System for Unmanned Aerial Vehicles,” IEEE Transactions on Control Systems Technology, vol. 25, no. 1, pp. 192–207, Apr. 2016.

[4] J. Pestana, I. Mellado-Bataller, J. L. Sanchez-Lopez, C. Fu, I. F. Mondragón, and P. Campoy, “A General Purpose Configurable Controller for Indoors and Outdoors GPS-Denied Navigation for Multirotor Unmanned Aerial Vehicles,” Journal of Intelligent & Robotic Systems, vol. 73, no. 1-4, pp. 387–400, Oct. 2013.

[5] T. Wang, C. Wang, J. Liang, Y. Chen, and Y. Zhang, “Vision-Aided Inertial Navigation for Small Unmanned Aerial Vehicles in GPS-Denied Environments,” International Journal of Advanced Robotic Systems, vol. 10, no. 6, pp. 1–12, May 2013.

[6] OpenCV team, “About,” OpenCV. [Online]. Available: https://opencv.org/about/. [Accessed: 09-Jul-2019].

[7] ArUco, “Detection of ArUco Markers,” OpenCV. [Online]. Available: https://docs.opencv.org/3.1.0/d5/dae/tutorial\_aruco\_detection.html. [Accessed: 27-Jul-2019].

[8] racer993, racer993, T. J. 10, C. J. 8, roma M. 30, racer993 P. authorM. 31, eric O. 29, K. A. 17, and racer993 P. authorA. 17, “Connect to the Raspberry Pi via SSH / Putty,” *raspberrypi4dummies*, 24-Mar-2013. [Online]. Available: <https://raspberrypi4dummies.wordpress.com/2013/03/17/connect-to-the-raspberry-pi-via-ssh-putty/>. [Accessed: 14-Oct-2019].

[9] racer993 and racer993, “First time configuration of my Raspberry Pi,” *raspberrypi4dummies*, 24-Mar-2013. [Online]. Available: <https://raspberrypi4dummies.wordpress.com/2013/03/17/first-time-configuration-of-my-raspberry-pi/>. [Accessed: 14-Oct-2019].

[10] “Raspberry Pi VNC Server - Setup Remote Desktop for your Pi,” *YouTube*, 14-Feb-2016. [Online]. Available: <https://youtu.be/JYKZjUgtOYw>. [Accessed: 14-Oct-2019].

[11] G. Lecakes, “OpenCV Basics - 11 - Building OpenCV Contribute with CMake,”  YouTube, 08-Apr-2016. [Online]. Available: https://www.youtube.com/watch?v=fIpTks0G2m0&list=PLAp0ZhYvW6XbEveYeefGSuLhaPlFML9gP&index=11. [Accessed: 09-Jul-2019].

[12] A. Rosebrock, “Install OpenCV 4 on your Raspberry Pi,” *PyImageSearch*, 28-Jun-2019. [Online]. Available: https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/. [Accessed: 10-Oct-2019].

[13] A. Rosebrock, “Accessing the Raspberry Pi Camera with OpenCV and Python,” *PyImageSearch*, 05-Feb-2019. [Online]. Available: https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/. [Accessed: 10-Oct-2019].

[14] T. Fiorenzani, “OpenCv and Camera Calibration on a Raspberry Pi 3,” *YouTube*, 20-Jan-2018. [Online]. Available: https://youtu.be/QV1a1G4lL3U. [Accessed: 10-Oct-2019].

[15] T. Fiorenzani, “PRECISION LANDING | with OPENCV and ARUCO Markers | Part 1,” *YouTube*, 19-Aug-2018. [Online]. Available: https://youtu.be/wlT\_0fhGrGg. [Accessed: 10-Oct-2019].

[16] MAVLink, “Python (mavgen),” *Python (mavgen) · MAVLink Developer Guide*. [Online]. Available: https://mavlink.io/en/mavgen\_python/. [Accessed: 02-Dec-2019].

[17] “MAVProxy¶,” *MAVProxy - MAVProxy 1.6.4 documentation*. [Online]. Available: http://ardupilot.github.io/MAVProxy/html/index.html. [Accessed: 02-Dec-2019].

# Appendices

## Appendix A

(1)

` (3)

Table 2: *Rotation and Translation Vectors*

A close up of text on a white background

Description automatically generated

## Appendix B

B.1: *MATLAB Dead Reckoning Code*

clear;clc;close all  
% Daniel Montgomery  
% Inertial Dead Reckoning  
% Last Modified 3/28/2019  
% Modified 4/18/2019 By: Ahmed Mekky  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% Initial Conditions  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% COORDINATE SYSTEM  
% Z: Up  
% X: Forward (North)  
% Y: Left (West)  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% Inertial Frame  
 % Rotation  
 wI=[0,0,0]; % Inertial Angular Rate [phidot thetadot psidot]  
 thetaI=[0,0,0]; % Euler Angle [phi theta psi]  
 % Translation  
 aI=[0,0,0]; % Inertial Acceleration [xddot yddot zddot]  
 vI=[0,0,0]; % Inertial Velocity [xdot ydot zdot]  
 rI=[0,0,0]; % Inertial Position [x y z]  
% Body Frame  
 % Rotation  
 wB=[0,0,0]; % Body Angular Rate [p q r]  
 thetaB=[0,0,0]; % Body Angle [pInt qInt rInt]  
 % Translation  
 aB=[0,0,0]; % Body Acceleration [udot vdot wdot]  
 vB=[0,0,0]; % Body Velocity [u v w]  
  
A=xlsread('testF6D2c.xlsx');  
A=flipud(A);  
 [n,~]=size(A);  
 rawimu=A(:,1);  
% Shift matrix down to skip random initial data  
 j=1;  
 while j < 100  
  
 count=j;  
  
 if A(j,3) <= 9.81\*cos(pi/6)  
 j=j+1;  
  
 else  
 j=100;  
 end  
 end  
 [n,~]=size(A);  
 A=A(count:n,:);  
  
  
% Timestep  
 Hz=1000; % Hertz  
 dt=1/Hz; % Timestep  
 tstart=(count-1)\*dt; % Starting time of calc  
 tstop=(n-1)\*dt; % Final time at n  
 t=tstart:dt:tstop; % Time vector  
  
  
% Gravity calibration  
 G=A(count:50+count,1:3); % collect averages of component accelerations  
 gx=mean(G(1:50,1)); % assuming drone is somewhat level and still  
 gy=mean(G(1:50,2)); % for first .05 seconds  
 gz=mean(G(1:50,3));  
 g=[gx,gy,gz];  
% gg=[0, 0, -9.81]; %Mekky  
  
% State Vector  
 X=zeros(n-count,21+3+3+3); % [wB,aB,wI,thetaI,aI,vI,rI];  
  
% Euler Integration Method  
vI2=[0,0,0]; rI2=[0,0,0]; thetaI2=[0,0,0]; v1b=[0,0,0];  
rI3=[0,0,0]; rIB=rI3;  
  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% Kalman Filter  
P\_bar = 1\*eye(12,12); %priori covariance  
x\_caret\_i=0;  
y\_i = 0;  
  
Q = 0.001\*eye(12); %Covariance of noise for State (assumed to be zero)  
R = 0.001\*eye(12); %Covariance of noise for for Measurements (assumed to be zero)  
P = eye(12);  
  
t=0;  
for i=count:n-count+1  
  
% Read in data  
 wB=A(i,4:6).\*(pi/180); % wB @t(i)  
 aB=A(i,1:3); % aB @t(i)  
 g\_b = 9.8\*[-sin(thetaI(1)), sin(thetaI(1))\*cos(thetaI(2)), cos(thetaI(1))\*cos(thetaI(2))];  
 a\_bb =aB-g\_b; % Acce. in Body frame without gravity  
% Angular Rate Transformation  
 T=Transform(thetaI); % thetaI @t(i)  
 wI=T\*wB'; % wI @t(i)  
 wI=wI';  
% Linear Acceleration Transformation  
 R1=Rotate(thetaI); % thetaI @t(i)  
 aI=R1\*aB'; % aI @t(i)  
 V\_BI=R1\*v1b';  
 aI=aI'-g;  
 aII=R1\*a\_bb';  
 aII=aII';  
  
% State Defined @ t(i)  
 X(i,:)=[wB,a\_bb,wI,thetaI,aII,vI,rI,v1b,V\_BI',rI3];  
  
% Euler Integration  
 thetaI2 = EulerInt(thetaI,wI,dt);  
 v1b = EulerInt(vB,a\_bb,dt);  
 rI3 = EulerInt(rIB,V\_BI',dt);  
 vI2 = EulerInt(vI,aI,dt);  
 rI2 = EulerInt(rI,vI,dt);  
  
 % Update vectors  
 thetaI=thetaI2;  
 vI=vI2;  
 vB=v1b;  
 rIB=rI3;  
 rI=rI2;  
  
 x\_bar = [rI vI thetaI wI];  
  
 %"A"Matrix  
 F=[1 0 0 (dt/2) 0 0 0 0 0 0 0 0;  
 0 1 0 0 (dt/2) 0 0 0 0 0 0 0;  
 0 0 1 0 0 (dt/2) 0 0 0 0 0 0;  
 0 0 0 1 0 0 0 0 0 0 0 0;  
 0 0 0 0 1 0 0 0 0 0 0 0;  
 0 0 0 0 0 1 0 0 0 0 0 0;  
 0 0 0 0 0 0 1 0 0 (dt/2) 0 0;  
 0 0 0 0 0 0 0 1 0 0 (dt/2) 0;  
 0 0 0 0 0 0 0 0 1 0 0 0;  
 0 0 0 0 0 0 0 0 0 1 0 0;  
 0 0 0 0 0 0 0 0 0 0 1 0;  
 0 0 0 0 0 0 0 0 0 0 0 1];  
  
%B Matrix  
B=[0 0 0;  
 0 0 0;  
 0 0 0;  
 (dt/2) 0 0;  
 0 (dt/2) 0;  
 0 0 (dt/2);  
 0 0 0;  
 0 0 0;  
 0 0 0;  
 0 0 0;  
 0 0 0;  
 0 0 0];  
  
H=[0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 0 0 0;  
 0 0 0 0 0 0 0 0 0 1 0 0;  
 0 0 0 0 0 0 0 0 0 0 1 0;  
 0 0 0 0 0 0 0 0 0 0 0 1];  
  
  
Y=[wI];  
u=aI;  
P12=P\_bar\*H'; %cross covariance  
K=P12\*inv(H\*P12+R); %Kalman filter gain  
x\_caret = x\_bar+(K\*([zeros(1,9) Y]-(H\*x\_bar')')')'; %state estimate  
P\_caret=P\_bar-K\*H\*P\_bar;  
  
P\_dot = F.\*P\_caret.\*F' + Q;  
x\_dot = (F\*x\_caret')'+(B\*u')';  
  
P\_bar = P\_caret+ (P\_dot) \* dt;  
x\_bar = x\_caret+ (x\_dot) \* dt;  
x\_caret\_i = x\_caret\_i+ x\_caret \*dt;  
y\_i = y\_i + (Y).\*dt;  
Yi (:,i) =y\_i;  
Y\_b (:,i) =Y;  
X\_BAR(:,i)= x\_bar;  
X\_CARET(:,i)= x\_caret;  
X\_CARET\_I(:,i)=x\_caret\_i;  
TT(i) = t;  
  
t=dt+t;  
  
  
  
end  
X(:,1:3)=X(:,1:3).\*180/pi;  
X(:,7:9)=X(:,7:9).\*180/pi;  
X(:,10:12)=X(:,10:12).\*180/pi;  
plot(TT,X\_CARET(1:3,:),TT,X\_BAR(1:3,:),TT,X(:,19:21))  
xlabel('Time(ms)');  
ylabel('Position(m)');  
title(' Position vs. Time');  
  
  
% Functions  
% Euler State Integrator  
function x2 = EulerInt(x,xdot,dt)  
  
 x2=x+xdot.\*dt;  
  
end  
% Rotation Matrix  
function R = Rotate(a)  
  
 Rx=[1 0 0; 0 cos(a(1)) -sin(a(1)); 0 sin(a(1)) cos(a(1))];  
 Ry=[cos(a(2)) 0 sin(a(2)); 0 1 0; -sin(a(2)) 0 cos(a(2))];  
 Rz=[cos(a(3)) -sin(a(3)) 0; sin(a(3)) cos(a(3)) 0; 0 0 1];  
 R=Rz\*Ry\*Rx;  
  
end  
  
% Transformation Matrix  
function T = Transform(a)  
  
T=[1 sin(a(1))\*tan(a(2)) cos(a(1))\*tan(a(2));  
 0 cos(a(1)) -sin(a(2))  
 0 sin(a(1))/cos(a(2)) cos(a(1))/cos(a(2))];  
  
end

B.2: *IMU Test Results (x-direction motion)*

A screenshot of a video game

Description automatically generated

B.3: *IMU Test Results (y-direction motion)*

A screenshot of a cell phone

Description automatically generated

B.4: *IMU Test Results (z-direction motion)*

A screenshot of a video game

Description automatically generated

B.5: *Raw IMU acceleration data for 1-D test in x-direction (1) and z-direction (2)*A close up of a map

Description automatically generated

Appendix C:

GPS Python Coding for Raspberry Pi

C.1: *Python code to extract specific lines that include keywords from NMEA strings in GPS data.*

infile = r"D:\Documents and Settings\xxxx\Desktop\test\_log.txt"

important = []

keep\_phrases = ["test",

"important",

"keep me"]

with open(infile) as f:

f = f.readlines()

for line in f:

for phrase in keep\_phrases:

if phrase in line:

important.append(line)

break

print(important)

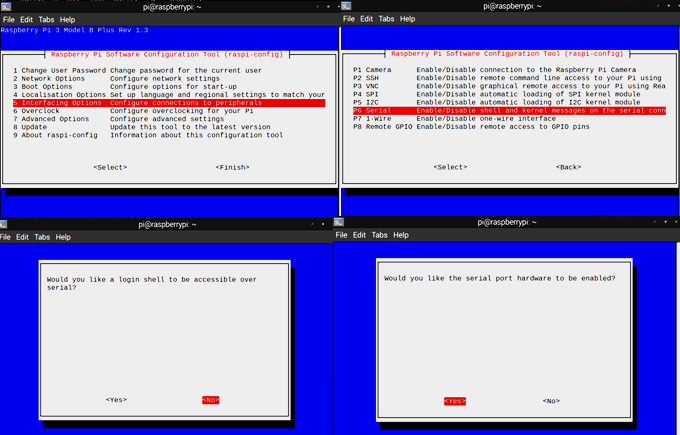
C.2: *Primary Configuration*

Using the sudo commands in the device terminal:

pi@raspberrypi ~ $ sudo apt-get update  
pi@raspberrypi ~ $ sudo apt-get upgrade  
pi@raspberrypi ~ $ sudo reboot

Enable the serial port on our Pi by assessing the configuration file and enabling the serial port while ensuring that the serial console remains disabled. This can be accomplished in the device’s configuration file by typing ‘pi@raspberrypi ~ $ sudo raspi-config’, then Select in order: Interfacing options, Serial, No, Yes, and then Yes to reboot (Figure A-D.2). This should allow the GPS to function on the Raspberry Pi.

C.3: *Enabling Raspberry Pi Serial Port*

C.4*: Gpsmon data of Kaufman Hall*A screenshot of a cell phone

Description automatically generated

C.5 : Google Map data of Kaufman Hall

18:35 
49th St 
Kaufman Hall 
Chick-fil-A 
LTE • 
Hair Of The Dog 
Eatery, Norfolk 
Del Vecchios 
Kaufman Hall 
W 47th S 
W 46th St 
Chartwav Arena 
4635 Hampton Blvd, Norfolk, VA 23529 
1 min 
+ FollOW 
DIRECTIONS 
O 
START 
SAVE 
9 
Are you here now? v 
Measure distance 
8785VMPW+82 
(36.8857957, -76.3049635) 
Add a missing place 
LABEL 
O 

## Appendix D:

D.1: *ArUco Marker and Camera Calibration Code*

A screenshot of a social media post

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

D.2: *Building OpenCV Video*

G. Lecakes, “OpenCV Basics - 11 - Building OpenCV Contribute with CMake,”  YouTube, 08-Apr-2016. [Online]. Available: https://www.youtube.com/watch?v=fIpTks0G2m0&list=PLAp0ZhYvW6XbEveYeefGSuLh

aPlFML9gP&index=11. [Accessed: 09-Jul-2019].

D.3: *ArUco Marker Landing Code*

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

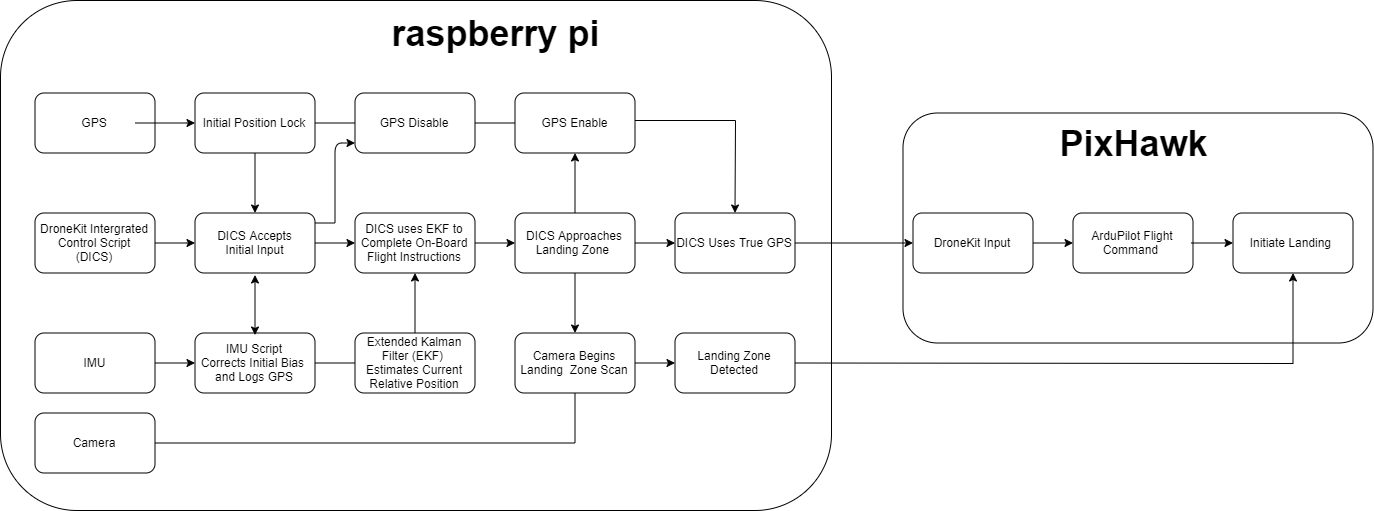
Description automatically generated

A close up of a logo

Description automatically generated

## Appendix E

E.1: *Program Block Diagram.*



E.2:

