

March 2016

Autonomous Quadrotor Navigation and Guidance

Alyssa Nicole Hollander
Worcester Polytechnic Institute

Jonathan David Blythe
Worcester Polytechnic Institute

Krzysztof Adam Borowicz
Worcester Polytechnic Institute

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

Repository Citation

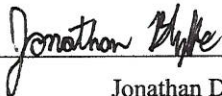
Hollander, A. N., Blythe, J. D., & Borowicz, K. A. (2016). *Autonomous Quadrotor Navigation and Guidance*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/3584>

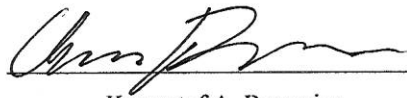
This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.

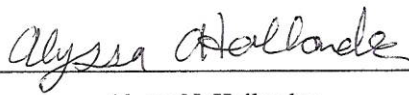
Autonomous Quadrotor Navigation and Guidance


A Major Qualifying Project Report
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
In Aerospace Engineering

By


Jonathan D. Blythe


Krzysztof A. Borowicz


Alyssa N. Hollander

Approved by: 
Prof. Raghvendra V. Cowlagi, Advisor
Aerospace Engineering Program
Mechanical Engineering Department, WPI

Abstract

This project involves the design of a vision-based navigation and guidance system for a quadrotor unmanned aerial vehicle (UAV), to enable the UAV to follow a planned route specified by navigational markers, such as brightly colored squares, on the ground. A commercially available UAV is modified by attaching a camera and an embedded computer called Raspberry Pi. An image processing algorithm is designed using the open-source software library OpenCV to capture streaming video data from the camera and recognize the navigational markers. A guidance algorithm, also executed by the Raspberry Pi, is designed to command with the UAV autopilot to move from the currently recognized marker to the next marker. Laboratory bench tests and flight tests are performed to validate the designs.

Fair Use Disclaimer: This document may contain copyrighted material, such as photographs and diagrams, the use of which may not always have been specifically authorized by the copyright owner. The use of copyrighted material in this document is in accordance with the “fair use doctrine”, as incorporated in Title 17 USC S107 of the United States Copyright Act of 1976.

Acknowledgments

We would like to thank the following individuals for their help and support throughout the entirety of this project.

Project Advisor: Professor Raghvendra Cowlagi

Table of Authorship

Section	Author	Project Work
Background		
Real World Applications	AH	N/A
Research	AH	N/A
Navigation, Guidance, and Control	JB	N/A
GPS	AH	N/A
Python and OpenCV	JB, KB	N/A
System Design		
Image Processing and Design Specifications	JB, KB	JB, KB, AH
Quadrotor UAV and Flight Control	JB	JB, KB
On-Board Computer	JB	JB, KB
Camera	JB	JB, KB
Power Systems and Flight Time	AH	AH
Mounting Hardware	KB	JB, KB
Quadrotor FAA Registration	JB	KB
System Development and Testing		
Preliminary Flight Tests	JB	JB, KB, AH
Shape Detection Code on PC	JB	JB, KB
Source Code	JB	JB, KB
Integration onto the Raspberry Pi	JB	JB, KB
Marker Selection	JB	AH
Connection to Pixhawk	JB	JB, KB
Mavlink and Dronekit API	JB	JB, KB
Cable Management	KB	KB
Results		
Image Processing	JB, KB	N/A
Hardware System Integration	JB, KB	N/A
Conclusion	JB, KB	N/A

Contents	
Abstract	I
Acknowledgments	II
Table of Authorship	III
1 Background	- 1 -
1.1 Project Objective	- 1 -
1.2 UAV Research	- 1 -
1.3 Navigation, Guidance, and Control	- 1 -
1.4 GPS	- 2 -
1.5 Python and OpenCV	- 2 -
2 System Design	- 4 -
2.1 Image Processing Design Specifications	- 4 -
2.2 Quadrotor UAV and Flight Control	- 5 -
2.3 On-Board Computer	- 5 -
2.4 Camera	- 6 -
2.5 Power Systems and Flight Time	- 6 -
2.6 Mounting Hardware	- 10 -
2.7 Quadrotor FAA Registration	- 14 -
3 System Development & Testing	- 15 -
3.1 Preliminary Flight Tests	- 15 -
3.2 Shape Detection Code on PC	- 17 -
3.3 Source Code	- 20 -
3.4 Integration onto the Raspberry Pi	- 20 -
3.5 Marker Selection	- 21 -
3.6 Connection to Pixhawk	- 21 -
3.7 Mavlink and Dronekit API	- 22 -
3.8 Cable Management	- 23 -
4 Results	- 24 -
4.1 Image Processing	- 24 -
4.2 Hardware System Integration	- 26 -
5 Conclusions	- 28 -
Appendix A: Raspberry Pi Setup	- 29 -
Appendix B: Connecting to Mavlink	- 31 -
Appendix C: Source Code and Testing Video Links	- 32 -
Appendix D: References	- 33 -

1 Background

Unmanned aerial vehicles (UAVs) are quickly becoming a staple of modern business and recreation; their versatility and inherent decreased risk to human life, makes UAVs particularly useful for aerial photography, surveillance, et cetera. With the future of UAVs quickly growing to everyday use, advanced guidance systems are becoming critical to the seamless navigation of UAVs. This project focuses on using a vision-based guidance system to achieve real world execution of a planned route. GPS is frequently used as a primary source of navigation data for UAVs however, in areas where GPS signal availability may be limited such as in buildings or underground, the need for an alternative navigation schema is clear.

1.1 Project Objective

This project explores the use of a vision-based guidance algorithm using an on board camera to control and guide the UAV when human intervention may be impossible or impractical. Some of the real world applications of this project include package delivery, crop dusting, and firefighting. Large corporations, such as Google and Amazon, are actively researching the possibility of using UAVs for the delivery of packages. This project provides guidance for UAVs which would allow the aircraft to avoid no fly zones, such as airports, and military installations. Additionally, the vision-based guidance of UAVs could help agriculturists though crop dusting by using rows in fields as markers. This removes the necessity for agriculturists to hire pilots to dust their fields and allows for simple automation of the entire process. A local fire department is actively pursuing using UAVs to enter buildings for search and rescue missions. [1] Using vision-based guidance; the UAV could provide firefighters with imaging of the inside of the burning building and could allow for an assessment of the condition of the fire as well as if there are people still inside the build without placing firemen at risk. In this case, the UAV would be fitted with a thermal camera in order to supply the UAV with data on the hotspots in the fire. This would provide guidance and navigation to keep the UAV from flying into the hotspot.

1.2 UAV Research

Autonomous quadrotor UAVs are growing more popular and are the subject of much research. A notable contribution is by GRASP Laboratory at the University of Pennsylvania. This work consists of autonomous quadrotor UAVs communicating with one another and avoiding obstacles [2, 3]. Similarly, the Flying Machine Arena team report on autonomous quadrotor UAVs that use vision-based guidance that communicate with one another and interact with a human operator [4, 5]. In order for this system to work, the quadrotor UAV needs a computer, on-board or off-board, that processes the image captured by the camera and provides commands to the quadrotor UAV.

1.3 Navigation, Guidance, and Control

To accurately traverse a planned route, a UAV must use various sensors and external monitors. In the case of the IRIS+, GPS and accelerometers are the primary devices used for navigation and guidance. These sensors provide telemetry which the onboard flight controller uses in order to fully understand its position in space. In order to travel from point A to point B, a guidance system needs to be implemented in order to compare the desired destination and current location. This comparison tells the UAV what it needs to do in order to accurately execute movements to reach the final destination. The output of the comparison sends commands to the control system for the motors which then execute the movements. All three parts need to work perfectly together in order to reach the final destination in a timely and accurate manner.

The navigation system and guidance system both have a multitude of sensors. Each sensor continually sends data to the main flight controller which then compares all of the readings. Once the readings are processed and compared, all systems receive a translated data input that are then put into a

feedback loop that compares old data to the new data that the sensors are constantly receiving. The continuous data processing creates an optimal route for travel.

GNC systems rely heavily on spatial awareness. Location information is determined using multiple sensors onboard the aircraft. As more sensors are implemented, more data is received by the flight controller which increases the system's awareness of its position of its environment. The disadvantage of this is that with more data, the flight controller and individual navigation and control systems require more time to fully process all parts. Therefore, a compromise needs to be made between the amount of sensors needed to accomplish the mission and the capabilities of the individual systems.

This project utilizes vision-based navigation and guidance to augment onboard navigation equipment allowing for marker traversal. Vision-based navigation and guidance describes any system that relies on image data to make determinations for path planning and motion. The image collected by the optical sensor is used for relative location data that is used to determine the vehicle's position. This information is then processed for guidance of the vehicle. A previous Worcester Polytechnic Institute Major Qualifying Project explored the use of stereovision to detect and avoid obstacles. [6]

1.4 GPS

Global Positioning System (GPS) provides position estimation. GPS is a passive system, meaning that the communication between the satellites and the receivers is unidirectional. There are three segments associated with GPS, which are outlined as follows: the space segment, user segment, and control segment. The space segment is composed of approximately 32 satellites, which are in approximately circular orbits at a radius of 26,560 km. There are 6 paths that are oriented 55 degrees with respect to the Earth's equatorial plane. The satellites continually transmit signals. The user segment is comprised of an unlimited number of receivers, which receive and process the signals that the satellites transmit. There is no restriction on the number of active receivers because the signal is unidirectional. This allows millions of people to use GPS at the same time. The control segment is comprised of a Master Control Station and 17 monitoring stations. Each satellite is visible to at least 2 monitoring stations at any one instant in time. The monitoring stations update the satellite's time and navigation messages in order to correct the satellite's position and time. [7]

1.5 Python and OpenCV

OpenCV is an open-source vision and communication library that allows for image processing and manipulation through the use of C, C++, Python, MATLAB and Java on various operating systems. OpenCV is the backbone of the python script and allows for the required image processing so that the quadrotor UAV can follow the ground route. [8, 9]

Python is an open-source programming language that is very well supported. C++ was considered for the project, but it was found that Python provides an easier environment for coding and testing. Python's strict adherence to 'good coding standards,' requiring strict indentation, aids in production of easily readable code as well as forces good programming habits. In addition, Python comes preinstalled on Raspbian which makes it the prime choice for using a Raspberry Pi 2 for communication with the quadrotor UAV. Python is both efficient and powerful; the syntax makes scripts easy to read for those working on them as well as someone simply trying to figure out someone else's script. Python is considered a high level language meaning it is closer to the way people communicate as opposed to assembly language. High level languages are known for intuitive functions and ease of programming, making python the perfect choice for this project. While high level languages are extremely useful for creating complex programs, they frequently make low-level operations more difficult such as Inter-device communication (the specialty of low level systems programming languages) however; with the vast array of libraries that are supported, Python is capable of accomplishing many tasks.

We use the OpenCV libraries to process live video frame-by-frame from the Pi Cam attached to the Raspberry Pi through a series of image manipulations and transformations.

1. `cv2.VideoCapture(0)`

We first need to declare the video capture objects from the Pi Cam using the `VideoCapture` function. This creates an object that we can later reference and pull frames from which we will manipulate. This essentially acts as a port through which the program has access to data captured by the Pi Cam.

2. `bcapWebcam.isopened()` and `capWebcam.read()`

Using the `capwebcam.isopened` function we can ensure that the program only runs while receiving frames from the Pi Cam. Each time the program iterates through the frame collection loop it first checks to make sure it received a frame to avoid crashing. This allows for the program to exit the program cleanly in the event of a camera disconnect rather than throwing an error as well as allowing us to send a “Land” command when this happens. The first step of the loop uses the `capWebcam.read()` function which takes a frame from the video stream and saves it as a local variable for the rest of the program to process.

3. `cv2.cvtColor(...)`

The first step in processing the image is to convert it to grayscale. This is done using the `cvtColor` command which takes a color image and converts it to black and white. This process causes an unavoidable loss of resolution. This is not detrimental to the program because we will be blurring the image in the next phase.

4. `cv2.GaussianBlur(...)`

To prepare the image for the next phase the image must first be blurred using the `GaussianBlur` function which uses a Gaussian based filter to blur the image. This is critical to the success of the Canny function in the next phase.

5. `cv2.canny(...)`

The `canny` function in the OpenCV library is the last step in the preparation of frames for processing by the code. This function outlines everything in view of the camera.

6. `cv2.findContours(...)`

This step eliminates the picture from the processing by storing all of the contours produced by the `canny` function in an array for processing. This allows us to quickly process the image by only considering the information that is relevant to us.

2 System Design

2.1 Image Processing Design Specifications

This project requires the use of a mounted camera and a predetermined path that is laid out with ground markers. The purpose of the camera is to gather information about the vehicle's local environment. In this case, the information gathered is the location of a ground marker with respect to the position of the quadrotor. The implemented script should determine the location of the center of the marker in the frame. This location is most easily determined in pixels and therefore will need to be converted to real world distances as a function of altitude. This can be completed using the standard OpenCV Python library. This process must be completed rather quickly; the minimum frame processing rate required is less than the time it takes the quadrotor UAV to traverse two adjacent markers.

To facilitate fast processing time, the group must maximize the ease of object detection. Using the OpenCV library, the image must be prepared for marker recognition initially through image blurring which eliminates the need for a high resolution camera; however, the camera must have a sufficient field of view for reasonably spaced markers and flight altitude. Similarly, the image is converted to high contrast greyscale which requires a high contrast marker of distinct shape that is not commonly found in nature. This prevents erroneous marker identification. Flight altitude must be determined by camera selection and marker size and spacing so that there are always at least two markers in the frame at one time but no more than three.

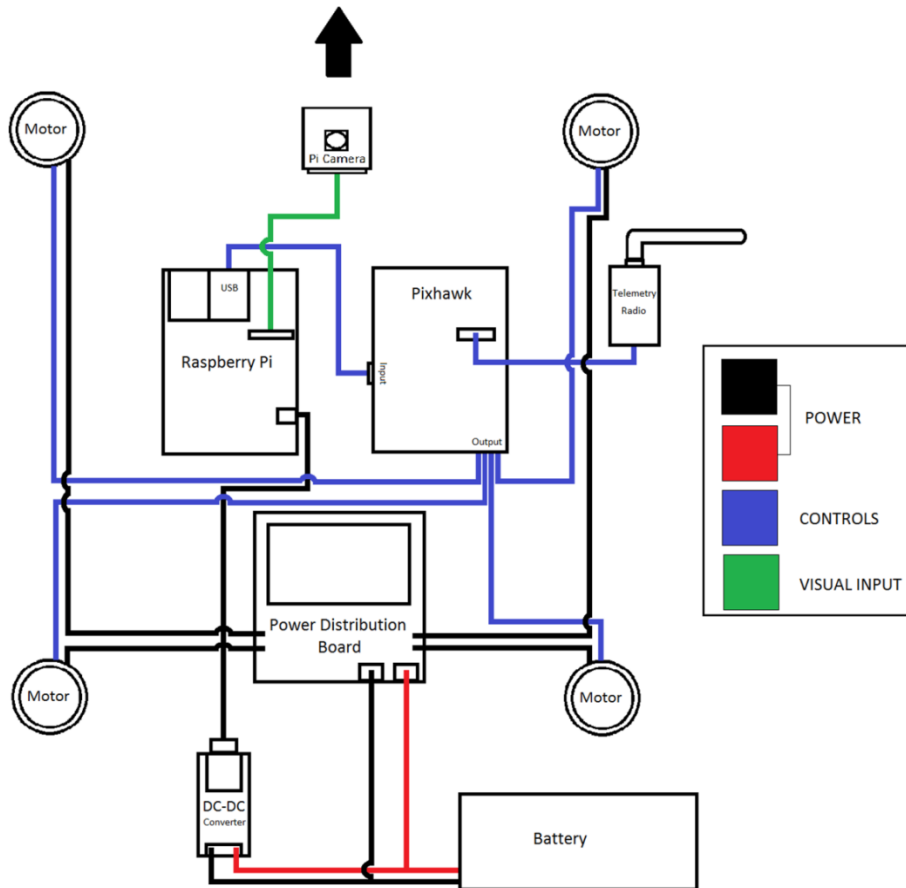


Figure 1: System Block Diagram

2.2 Quadrotor UAV and Flight Control

This project required a quadrotor UAV that was user friendly and had the ability to communicate with an on-board computer. The Vision-based Obstacle Avoidance MQP from last year selected the 3D Robotics IRIS for its inclusion of the Pixhawk flight controller. The Pixhawk is essentially the brain of the quadrotor UAV and is based on the Ardupilot Firmware [10, 11]. The purpose of the flight controller is to make any quadrotor UAV autonomous. Ardupilot comes with a program called Mission Planner for any computer or tablet that allows for communication between the quadrotor UAV and pilot. The software provides telemetry and waypoint creation abilities. There are many ports on the Pixhawk that allows for sensors such as GPS, radio, sonar, and serial ports for external computers. With all of the abilities of the Pixhawk and the ease of flight for the IRIS, the team decided to purchase the IRIS+ from 3D Robotics. The IRIS+ has the same design and components as the IRIS but has newer versions of the Ardupilot Firmware.

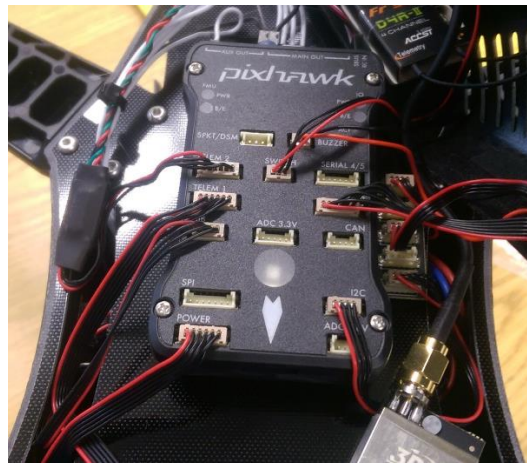


Figure 2: Pixhawk Flight Controller



Figure 3: 3DR IRIS+ Quadrotor UAV

2.3 On-Board Computer

In order to process the code and follow ground markers, the IRIS+ needs movement commands from an on-board computer. Arduino, Intel Galileo, and Raspberry Pi 2 were all considered. These computers each have their merits; however, the Raspberry Pi 2 was chosen. The Raspberry Pi 2 interface

is user friendly and the processing power as well as speed of the Raspberry Pi 2 also fit the requirements of the project.

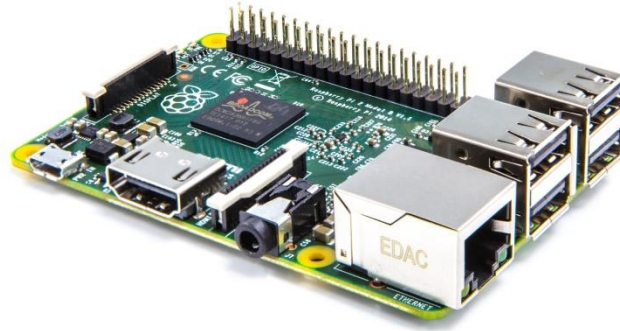


Figure 4: Raspberry Pi 2 [12]

Python comes natively on the Raspberry Pi 2 which is especially helpful since the image processing code is written in Python. Also, there are specific libraries such as Dronekit which enables communication with quadrotor UAV flight controllers. The computer can be powered directly from the Pixhawk so a secondary power supply is not necessary.

The Raspberry Pi runs with a fully interactive GUI which is wonderful for development but too CPU heavy for running on the Pixhawk. The Raspbian operating system allows for running the Raspberry Pi directly from the terminal as soon as power is plugged in. Running from the terminal on boot allows for less CPU intensive operating system and allows for more thread allocation to the image processing script.

2.4 Camera

The Python script for image processing blurs and greyscales the images that it captures, so a black and white camera was not necessary. The team needed a camera that provided enough resolution to accurately see ground markers at various heights and velocities. To meet these requirements, we considered a Webcam or a Raspberry Pi Camera, but determined that the Pi Cam's low weight and ease of integration was preferable and therefore the team continued the project using a Raspberry Pi Camera.

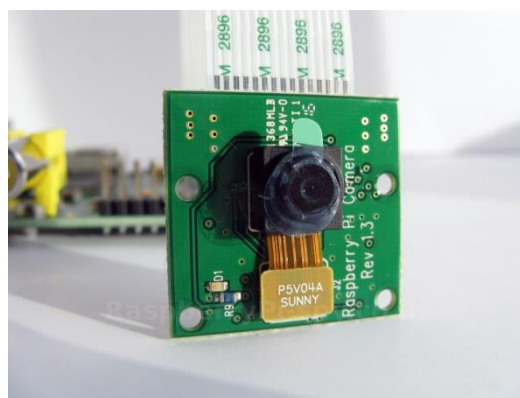


Figure 5: Raspberry Pi Camera [13]

2.5 Power Systems and Flight Time

In order to power the quadrotor and Raspberry Pi without adding another battery pack or external power supply, a power bus was constructed. The bus takes the power from the 3 cell, 11.1 volt, 5.6 Amp-hr battery and creates a positive and negative rail. These rails power the DC-DC converter and the

quadrotor UAV. The DC-DC converter is used to drop the voltage from 11.1 volts to 5 volts per the Universal Serial Bus Specification Revision 2.0 Standards [14]. From the DC-DC converter, a USB cable powers the Raspberry Pi, the on-board computer. The other branch of the power system directly feeds 11.1 volts to the quadrotor which is the required voltage to run the on-board flight controller, systems, and motors.

For this project, a power distribution system was developed that would successfully power all of the onboard electronics. Two different designs were considered to meet the power needs of the Quadrotor and the Raspberry Pi microcontroller. The first design developed a power distribution system and the second design add an independent circuit to power the Raspberry Pi microcontroller.

The first design was to create a power distribution cable that would distribute the power from the quadrotor's lithium polymer battery to the Raspberry Pi microcontroller and the quadrotor (see Figure 6). This design allowed the quadrotor and the Raspberry Pi microcontroller to be powered from the same source. This design also avoided the need to mount an additional source to the quadrotor. The total weight of this design is 1327 grams. Since the battery operates between 10.5 and 12.6 Volts, a dc to dc converter is necessary to decrease the voltage to 5 Volts for the Raspberry Pi Microcontroller.

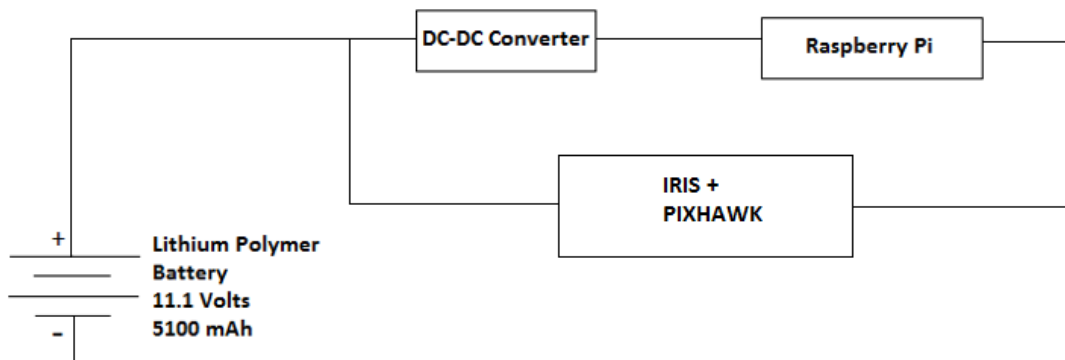


Figure 6: Power Distribution Diagram

The second design powered the quadrotor and the Raspberry Pi microcontroller from separate power sources (see Figure 7). Since this design uses two sources, the additional weight of the battery pack added to the quadrotor would cause the quadrotor battery to drain faster than the unloaded quadrotor. The total weight of this design is 1419 grams. This design like the previous would require the dc-dc converter to decrease the voltage for the Raspberry Pi microcontroller.

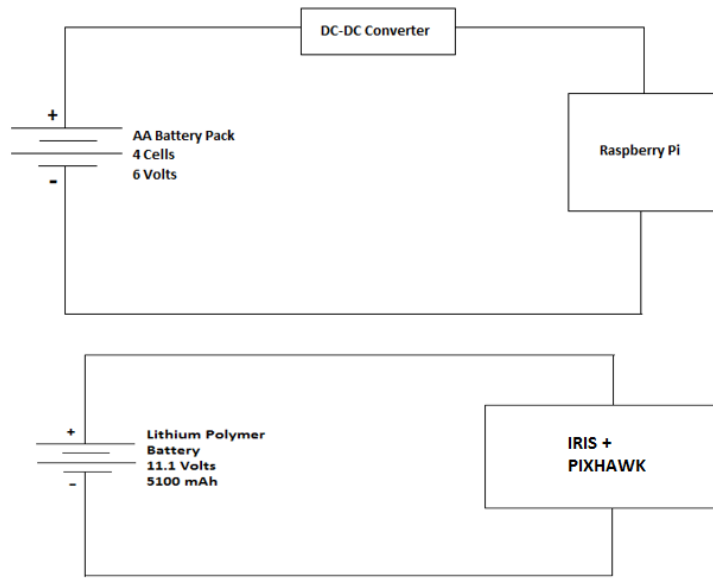


Figure 7: Independent Power Supplies Circuit Diagram

Calculations to determine how much the flight time decreased were used to determine which design to use for the power system. From the documentation, the quadrotor without any payload would have between 16 and 22 minutes of flight time. After several flight test with the quadrotor, the quadrotor was found to have approximately 20 minutes of flight time. The Lithium Polymer Battery was a 5100 milliamp-hour supply and had a weight of 320 grams, and the 4 cell AA battery pack was 5600 milliamp-hour supply and had weight of 92 grams. Additionally, the Raspberry Pi microcontroller could draw between 700 and 1000 milliamps. From this information, the flight time for each design was calculated (see Table 1).

Design	Weight (g)	Flight Time (Minutes)	
IRIS+ With Battery	1282	20	
IRIS+ with Raspberry Pi B (Design 1)	1327	18.42	18.07
IRIS+ with Raspberry Pi (Design 2)	1419	17.88	

Table 1: Flight Time Summary

Below are the calculations for the different designs:

Calculations for design 1:

Finding the current drawn by the quadrotor:

$$I = \frac{5100mAh * 60 \frac{min}{h}}{20 min} = 15300 mA$$

Minimum current drawn by the Raspberry Pi microcontroller and the quadrotor:

$$I_{min} = 15300 mA + 700 mA = 16000 mA$$

Maximum current drawn by the Raspberry Pi microcontroller and the quadrotor:

$$I_{max} = 15300 \text{ mA} + 1000 \text{ mA} = 16300 \text{ mA}$$

Maximum flight time (t) without weight factor:

$$t_{max} = \frac{5100 \text{ mAh} * 60 \frac{\text{min}}{\text{h}}}{16000 \text{ mA}} = 19.125 \text{ min}$$

Minimum flight time (t) without weight factor:

$$t_{min} = \frac{5100 \text{ mAh} * 60 \frac{\text{min}}{\text{h}}}{16300 \text{ mA}} = 18.773 \text{ min}$$

Maximum flight time (T) with weight factor:

$$T_{max} = 19.125 \text{ min} - \left(\frac{20 \text{ min}}{1282 \text{ g}} * 1327 \text{ g} - 20 \text{ min} \right) = 18.423 \text{ min}$$

Minimum flight time (T) with weight factor:

$$T_{min} = 18.773 \text{ min} - \left(\frac{20 \text{ min}}{1282 \text{ g}} * 1327 \text{ g} - 20 \text{ min} \right) = 18.071 \text{ min}$$

Calculations for Design 2:

Flight time (T) with weight factor:

$$T = 20 \text{ min} - \left(\frac{20 \text{ min}}{1282 \text{ g}} * 1419 \text{ g} - 20 \text{ min} \right) = 17.88 \text{ min}$$

The first design was selected because this design reduced the flight time less than the second design. Additionally, the first design added less weight to the quadrotor. Weight caused a greater decrease in flight time than adding an additional load to the lithium polymer battery. The first design also used a simpler construction as no additional power source needed to be mounted to the quadrotor. The power distribution system was constructed from XT60 connectors, flexible wire, and a dc-dc converter that outputs the voltage via a USB port (See Figure 8). The dc-dc converter output is ideal since the Raspberry Pi microcontroller is powered by USB to micro USB.



Figure 8: Power Distribution Cable

During the flight test after the integration of the power distribution system, the flight time decrease to about the flight time calculated value that corresponds with the minimum current draw for the

Raspberry Pi microcontroller. By using a singular source, the quadrotor and Raspberry Pi microcontroller boot at the same time which is necessary to make to connection between the Pixhawk and the Raspberry Pi microcontroller. Additionally, the battery maintains the ability to power the Raspberry Pi microcontroller for a short period of time after the quadrotor can no longer maintain flight.

2.6 Mounting Hardware

Aerial vehicles such as the IRIS+ quadrotor UAV have limited available space and payload capabilities. In order to streamline the system, we mounted all external guidance systems onboard the aircraft and avoided off board computation as much as possible.

The flexibility and small lead time of additive manufacturing persuaded us to pursue 3D printing for all externally mounted hardware including the camera and Raspberry Pi. Relatively low cost and lack of waste material made 3D printing perfect for this project's very limited budget.

Available printing materials include ABS, PLA, HIPS, Nylon, PVA and LayWOOD. Each option provides a lightweight medium for designing the mounting hardware. We quickly narrowed down the selection to Acrylonitrile butadiene styrene (ABS) and Polylactic acid (PLA) due to the plethora of information regarding printing with each material. Both ABS and PLA have their own desirable attributes; however, after trying both, we settled on using PLA.

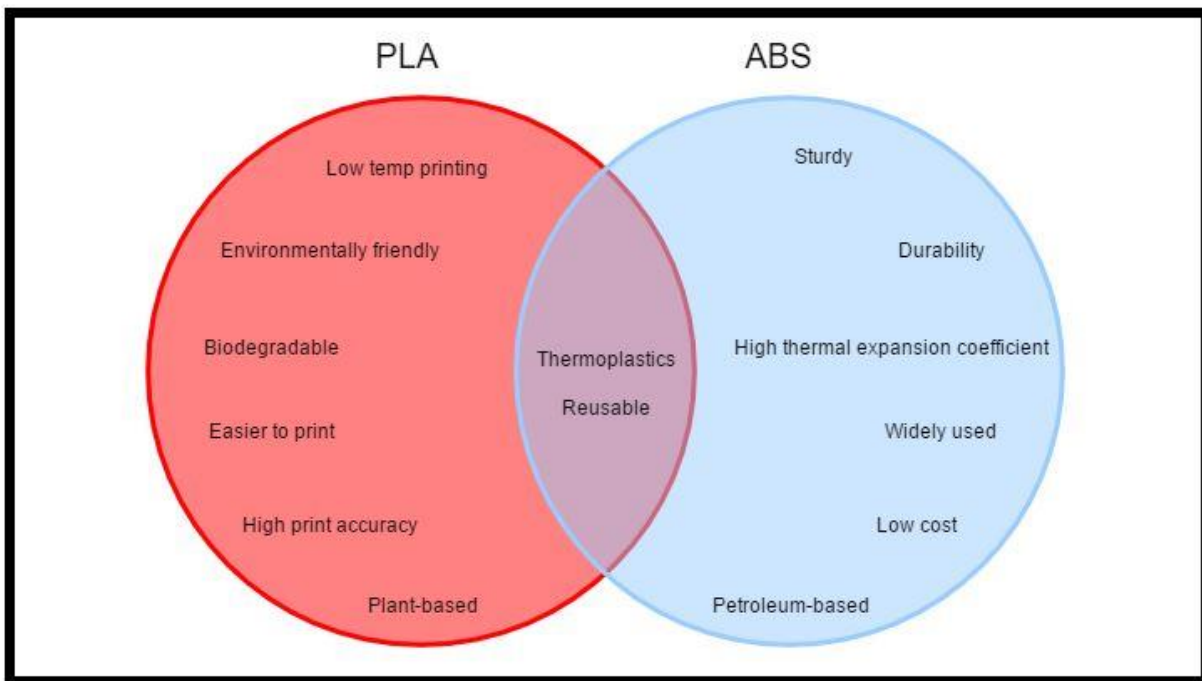


Figure 9: PLA and ABS Comparison

PLA's high print accuracy and ease of printing were what ultimately persuaded us towards selecting it over ABS, whose high thermal expansion coefficient makes part deformation result in a high print failure rate which we cannot afford with time and budget constraints. Additionally, PLA's ecofriendly nature was an added bonus.

Due to the low cost and quick turnaround of 3D printed parts, we were able to rapidly try multiple designs for a single part without having to wait for a custom part to be manufactured and shipped. By allowing for custom designed parts, we were able to manufacture parts in order to fit the group's specific needs rather than alter expectations to fit off the shelf components.

While 3D printing is advantageous, it also presents a series of challenges. Due to expansion and flow before solidification, part tolerancing can be extraordinarily difficult. 3D printing works on the principle of using layers of plastic laminated on top of one another in order to create a 3D component.

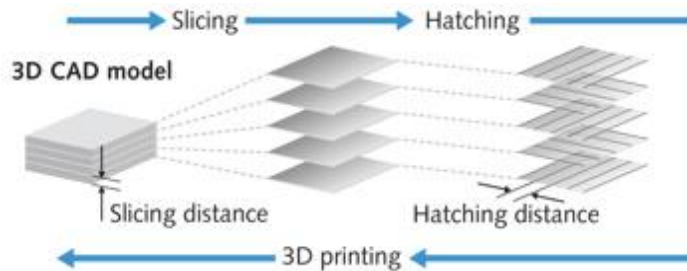


Figure 10: 3D Printing [15]

This method restricts us from designing components with large overhangs or suspended surfaces without the use of support materials, adding a new layer of difficulty to part design; many printed parts had to either be split or entirely redesigned to ensure that they were fully printable.

A case design was found on GrabCad (see Figure 11) which was made specifically for the Raspberry Pi 2 [16]. Due to the case having flat surfaces, an adapter was needed to properly affix the downloaded model to the curved shape of the IRIS+ belly. The initial concept was developed by 3D scanning (see Figure 12) the belly of the quadrotor and was refined by using a model of an IRIS+ from GrabCAD [17]. The quadrotor is designed by 3DR to be an aerial videography UAV so it has an existing Go-Pro™ mount attached to it. We utilized this mount by designing the Pi Cam mount to interface with the features of the Go-Pro™ mount making it a plug-and-play connection. Once parts were printed we began mounting them to the quadrotor. In addition, new feet were created by the team in order to widen the surface area that comes in contact with the ground. In snow and mud, this allows for the quadrotor to land with sinking. Wider feet also prevent the quadcopter from tipping on one leg.



Figure 11: Raspberry Pi Case [16]



Figure 12: 3D Scanning of IRIS+ Belly

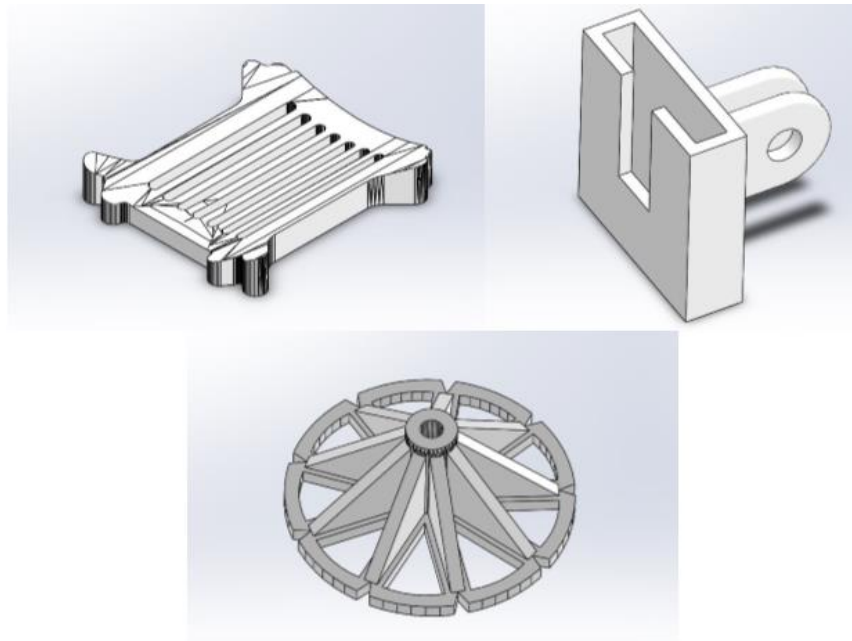


Figure 13: Developed Hardware CAD Models



Figure 14: 3D Printed Parts Mounted on IRIS+

2.7 Quadrotor FAA Registration

In order to fly the quadrotor in a public area, there are certain rules and regulations that any quadrotor pilot must follow. The rules regulations and polices can be found on the Federal Aviation Administration (FAA) website regarding Unmanned Aircraft Systems (UAS) [18]. One of the biggest requirements is that the aircraft must be within line of sight of the operator at all times and UAVs are limited to a 400 feet ceiling. Additionally, when flown within 5 miles of an airport, the operator of the aircraft must let the air traffic control of the airport know that flying will occur. All quadcopters must be registered to the pilot. During the registration process, the pilot will be assigned a registration number which must be placed on the quadrotor in a location that is easily visible or accessible without the use of tools. For ease of placement, the team attached the registration number on both sides of the quadrotor (see Figure 15

).



Figure 15: Quadrotor with Registration Number

3 System Development & Testing

3.1 Preliminary Flight Tests

In order to best understand the flight characteristics of the quadrotor UAV, initial non-autonomous flights were necessary. The IRIS+ comes with a USB antenna that can be connected to a computer in order to get telemetry data in real time. This ability pairs with software called Mission Planner which allows for waypoint planning, telemetry data, controller sensitivity, et cetera [19]. The controller has multiple modes for the quadrotor UAV to follow. The modes include standard, loiter, and auto. Standard flight is completely controlled by the pilot with the controller. Any inputs that the pilot puts into the controller are then fed to the quadrotor UAV. Loiter, as described before, allows for pilot input while maintaining a level and relatively stationary flight. Auto mode is the mode that is used when running the script on the Raspberry Pi 2. The quadrotor UAV was calibrated per the setup instructions. Flying the quadrotor UAV allowed for better understanding of the way it handles in various conditions. The quadrotor UAV has a loiter features where it hovers in place and tries to maintain its position. It was discovered that the quadrotor UAV stays within a three-foot radius.



Figure 16: Initial Flight Test in Net

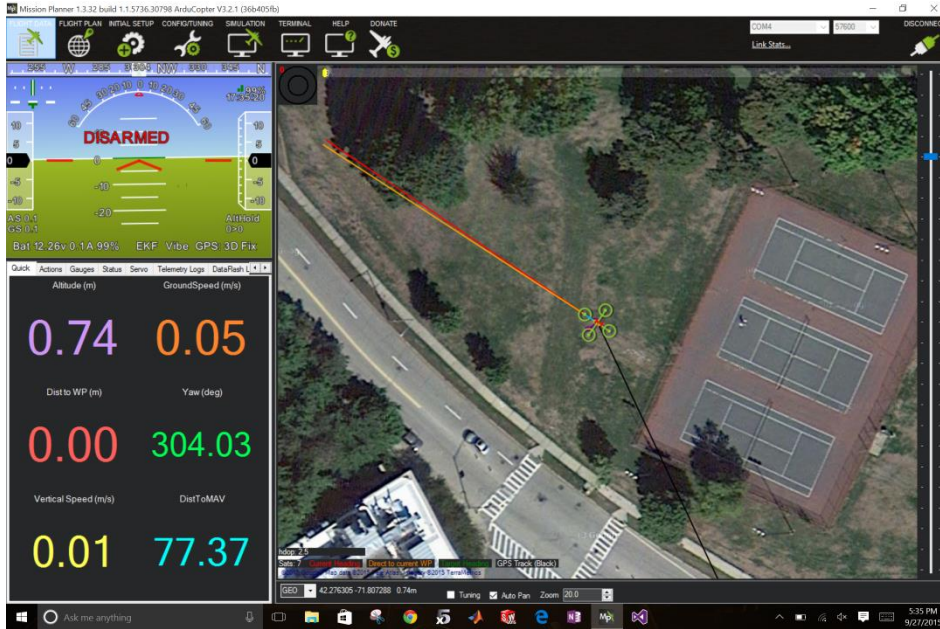


Figure 17: Mission Planner Screenshot

Once initial flights occurred, autonomous flights were tested using the Mission Planner waypoint software. First, waypoints were set to see how the quadrotor UAV naturally wants to get from one waypoint to the next. It was discovered that the quadrotor UAV wants to get to the next waypoint as fast as possible without worrying about its own orientation. When waypoints were set at various altitudes, the quadrotor UAV would slowly increase or decrease altitude along the flight path until it reached the proper height at the next waypoint. This feature can be hazardous if there are obstacles in the way. A preferred method of altitude gain or loss would be for the quadrotor UAV to fly vertically to the proper altitude before moving to the next waypoint.

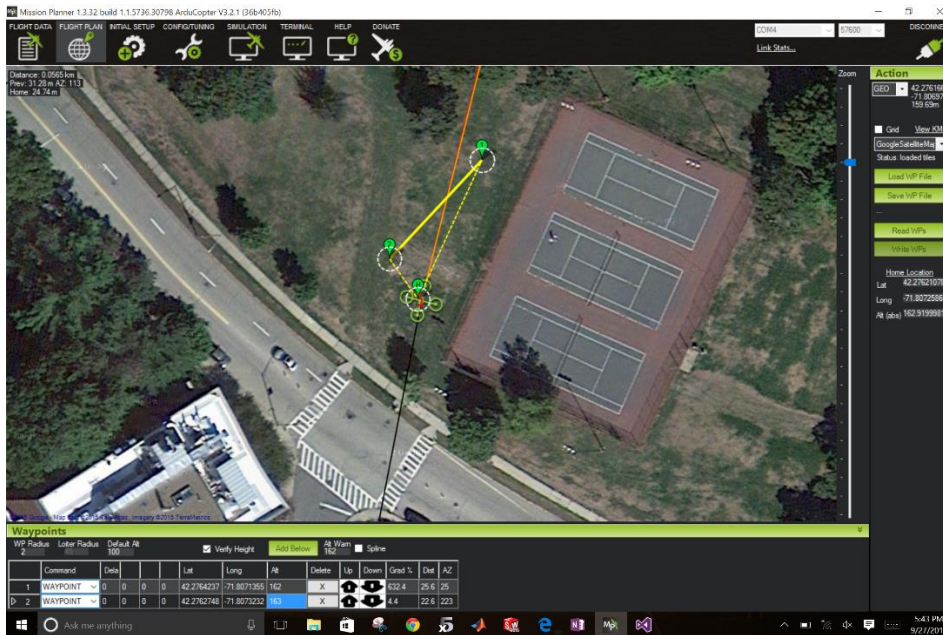


Figure 18: Mission Planner Waypoints Screenshot

Another important autonomous feature is the return to home mode. Wherever the quadrotor UAV is or whether the quadrotor UAV is flying between waypoints, it will return to the location where the flight began. This feature allows for the ability to return to the start location in case of emergency or error. When returning home, the quadrotor UAV will land in a three-foot radius of the take-off location. The same goes for the auto-land feature where the quadrotor UAV will stop in place and slowly descend to the ground. Both commands are able to be implemented regardless of flight mode by the operator of the controller. The Raspberry Pi also has the ability to give such a command.

During testing, the original IRIS+ that was purchased malfunctioned and shorted prior to full integration of the Raspberry Pi onboard. This created a delay in the full integration schedule along with full flight testing. The quadrotor UAV used for the MQP from the previous year was initially used but connection issues ensued due to the older firmware on the Pixhawk system. The broken quadrotor UAV was sent back to the manufacturer for repair. During this time, a second quadrotor UAV was purchased and was used for all final flight and integration tests.

3.2 Shape Detection Code on PC

In order to follow the marked path, a code needed to be developed which utilizes OpenCV and an external camera. Initial occurred on the PC due to the higher processing power of a PC. Once bugs were fixed and the code ran smoothly, the team moved the code to the Raspberry Pi. Python 2.7, OpenCV and a compiler were installed on the computer. The group used example image processing and shape detection Python scripts in order to develop the one specifically for the application of this project. One of the example scripts took an image and looked for contours within the image. The contours were then outlined and the image was blurred in order to get the outline to stand out. Another example script looked for various shapes within an image and did a similar process of outlining the desired shape. These two scripts were combined in order to create a video capture version of image processing. The group elected to use square markers for path following because squares are distinct enough from the environment to be picked up by the script regularly.

The overall object is for the quadrotor UAV to see square markers and then center the marker within the image frame. The first step to script development was for squares to be recognized. Next coordinates of the center of the square with respect to the center of the image frame was necessary. The location of the square in the frame was then translated to desired movement of the quadrotor UAV. The location output was given in pixels rather than meters so a conversion equation was created.

Two objects were placed on the ground in the lab and set a meter apart. Using the camera in the computer, pictures were taken at various heights starting from 0.5 meters to 2 meters. Since the distance of the objects was known, the distance in the image was measured in pixels and converted. Once all of the images were taken and analyzed, a curve fit was applied to the data points resulting in a linear relationship to marker height versus marker distance in pixels. The curve fit slope and intercept were applied to the Python script for pixel conversion. The altitude portion of the equation was set to be manually input for test purposes due to the lack of GPS signal inside the lab. GPS in conjunction with the accelerometers built into the quadrotor UAV provide altitude telemetry measurements. During the debugging process, the altitude was manually set to 1 meter in order to have a consistent altitude reading.

$$\frac{\text{Meters}}{\text{Pixels}} = (0.0016 * \text{Altitude}) + (3 \times 10^{-5})$$

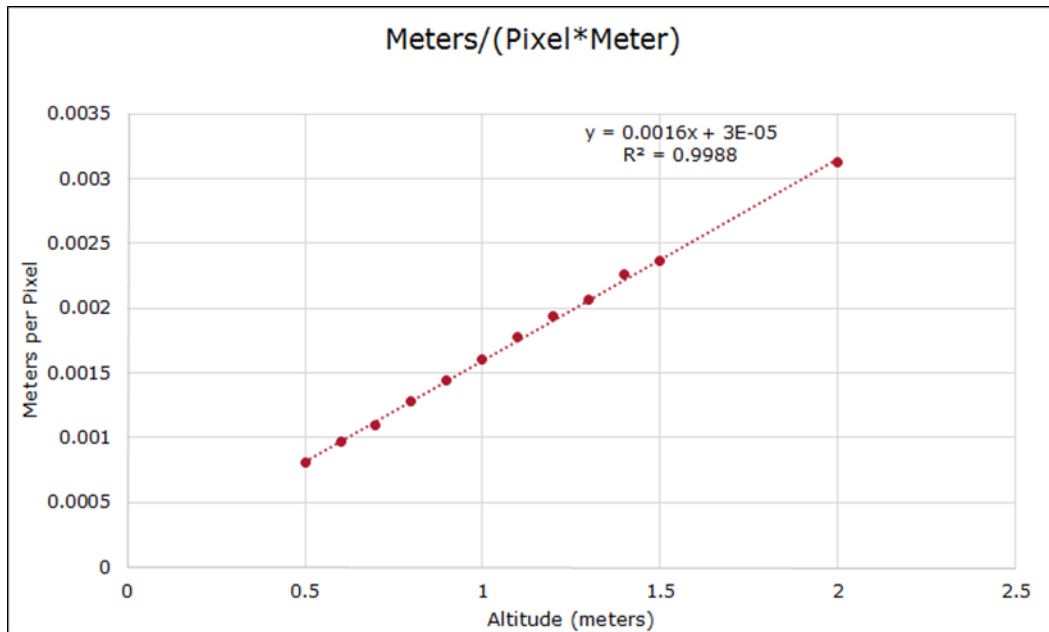


Figure 19: Altitude versus Meters per Pixel

Markers must be distinct from the environment in order for the camera to differentiate it from the environment that it is in. When testing the distance and direction output, a square was made in Microsoft paint and projected onto a screen. The computer was held at a constant distance and angle pointed at the screen. Making the computer and camera stationary allowed for the movement of the square on the projected image to see if the output of movement commands were accurate and constant as well.

After ensuring the distance output functioned, different modes need to be created so that once the next square in the path was found, there would not be confusion between the first and second squares. When multiple squares were in the image originally, the script would jump between the various squares and the distance output would change respectively. Creating various modes remedied this issue. The two developed modes are Search and Fly-To. Fly-To mode splits the image into 4 sections. The center of the image has a square section that is slightly larger than the marker in the frame which acts as a null zone where the previous marker is ignored to allow the finding of other markers. Around this null zone is a hover zone where the if the marker exits the null zone due to drift, the quadrotor UAV will center the marker in the frame. Around the hover region is a mode switch and search region. While the first marker is ignored, the script looks for the next marker in the search zone. The quadrotor UAV then proceeds to center that square in the center of the image frame. Once this process begins, the marker passes through a mode switch mode.

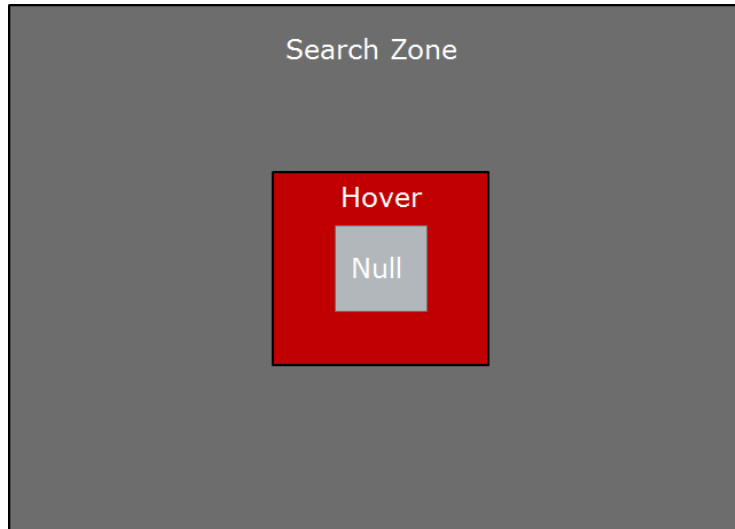


Figure 20: Search Mode Frame Zones

The second mode is the Fly-To mode. This mode splits the image into 3 sections. The bottom half of the frame is a null zone. As the quadrotor UAV moves forward towards the next marker, the original marker will leave the search mode null zone, therefore in Fly-To mode, a null zone is needed behind the quadrotor UAV so that the first marker is ignored. At the very top of the frame is a search zone where the script looks for the marker toward which the quadrotor UAV is flying. As the markers get closer to the center, it passes through a mode switch zone. During the switch, the marker enters the hover zone of the search mode and is then centered. Meanwhile the script looks for the next marker. The process of Search, switch mode, Fly-To, and switch mode repeats until the quadrotor UAV flies to the final marker. Once the quadrotor UAV flies to the last marker, it will hover for 30 seconds and then land once the algorithm processes that there are no more markers.

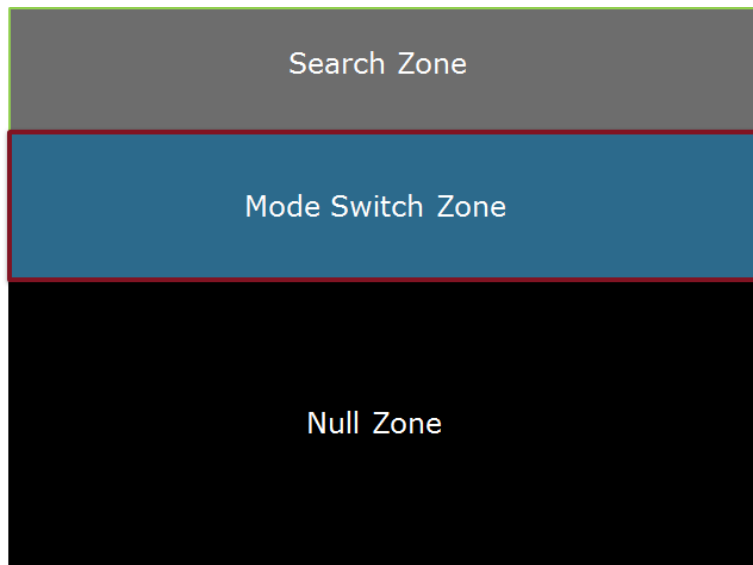


Figure 21: Fly-To Mode Frame Zones

A similar test was conducted with the movable square marker on a projector screen. Again, the camera was held stationary and the square was moved in the frame. The output of the script worked as desired with mode switching and movement commands.

3.3 Source Code

In addition to the OpenCV library, other open source Python scripts were adapted and combined in order to get Quadrotor UAVCode.py to work as desired. In order to follow the marked out path, the quadrotor needs to process the image frames captured by the Pi Cam. Better understanding the OpenCV commands required looking for example scripts that did exactly what the team needed. One script was found that processes still images and outlines all of the edges [20]. The issue with these this script was that it only focused on individual images rather than continuous video feed. Therefore, another script was found that uses shape detection through continuous video [21]. Both examples were decomposed and then converted to a form that allows for the groups system to find squares in a continuous video capture environment and outline the square to calculate distances. See [Appendix C](#) for links to the source code.

3.4 Integration onto the Raspberry Pi

After all of the initial PC testing, the code was implemented onto the Raspberry Pi. The Raspberry Pi was loaded with the proper libraries and the shape detection code was implemented. Before using the Pi Cam with the Raspberry Pi, a USB webcam was connected in order to verify that the script work in the same manner as on the computer. At first, the framerate of the output video was too low for consistent marker recognition. This issue was fixed by overclocking the CPU on the Pi as well as using the Pi Cam.

The next step was to test the program outside of the GUI. Running from the terminal and out of GUI would increase frame rate and run the overall code faster since the CPU does not need to run the visual desktop. Once the GUI was disabled, an attempt was made to run the code from the boot terminal. The problem with this method was that the terminal did not boot into the main root user as needed so the script did not run due to lack of permissions. When the team tried to log back into the GUI environment for further development, it was discovered that the team was locked out. The MQP members from the previous year had changed the password on the Pi which was unknown to the team.

This issue resulted in a clean reinstall of the Raspbian operating system. At first, the newest version of Raspbian, Jessie, was installed and the OpenCV libraries were installed once more. When running the script, the libraries were not recognized, thus causing a failure to run the script. The solution to this problem was to install an older version of Raspbian called Wheezy. The complete refresh of the Raspberry Pi helped remove any extra libraries that were not needed as well as reset the password for easy transition between GUI and terminal boot options. The install process as well as need libraries are outlined in [Appendix A](#).

3.5 Marker Selection

For the purposes of initial flight tests, the altitude at which the quadrotor UAV would fly was selected to be at head height or lower for safety purposes. Floppy disks were selected as the ground markers due to their square shape as well as relatively small size. The team had multiple floppy disks of various colors in order to have different options of contrasting color depending on the flight environment. With the integration of the Raspberry Pi Camera, altitude tests were conducted to see how high the quadrotor UAV needed to fly in order to consistently see the floppy disk marker. It was discovered that at 1.5 meters, the floppy disk was seen and its location processed by the shape recognition algorithm consistently.



Figure 22: Ground Marker

3.6 Connection to Pixhawk

In order to connect the Pixhawk and Raspberry Pi, a shortened USB cable was used to connect the Type A USB port on the Raspberry Pi to the external micro-USB port on the IRIS+. The USB cable provides telemetry readings from the Pi to the Pixhawk. Initially, the connection allowed power to go from the battery through the Pixhawk and into the Raspberry Pi but was not enough to fully boot the Raspbian operating system. This limitation resulted in a power bus outlined in Section 2.5. Specific libraries must be installed on the Raspberry Pi in order to allow communication between the two devices.

3.7 Mavlink and Dronekit API

Mavlink is the communication protocol used to send commands to the Pixhawk from the Raspberry Pi. Once all of the aforementioned libraries are installed on the Raspberry Pi, one can go into the terminal and run the mavproxy python script [22]. The protocol then looks for external connections to the Pixhawk and sends a ‘heartbeat’ that ensures a solid connection is made. In order to have a solid connection, a 2 Hertz transmission between both systems is required. After a connection is made, the user is able to send commands to the Pixhawk by typing directly into the terminal. The original test, seen below, armed and disarmed the motors. See [Appendix B](#) for connection directions.

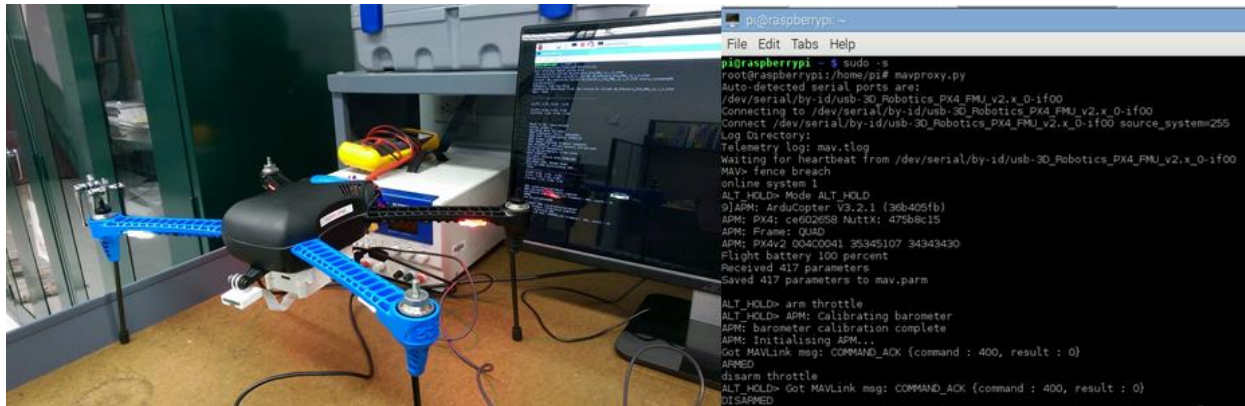


Figure 23: Initial Connection Bench Test

Once the initial connection and command was tested and verified, further testing ensued. The Raspberry Pi was hooked up to a wireless keyboard and a wireless mouse. While plugged into a monitor via a 50 foot HDMI cable, the quadrotor UAV was placed in the net in the lab and flown. As before, the arm and disarm throttle commands were used but in order to fly, the flight mode needed to be changed in order to accept full flight commands. The mode change command is a simple statement (mode GUIDED) which switches the mode as if the done on the controller.

Testing the connection was done through terminal commands. Once a connection is established, the Raspberry Pi looks for a signal or ‘heartbeat’ from the quadrotor UAV. After establishing a solid connection, commands can be sent from the Raspberry Pi to arm the quadrotor UAV and initialize flight. When finding markers, the image processing script provides flight commands and controls the movement from one marker to the next.

DroneKit is an API that allows for communication of Python scripts and the Pixhawk flight controller [23]. The API has its own flight commands for arming the motors, taking off, switching the flight modes, et cetera. Droneapi is an imported library within the Quadrotor UAVCode.py that utilizes these flight commands. In order to fly autonomously, Mavlink does not need to be started. Mavlink is only needed to test that a connection can be made. Once that connection is proven, the Python script can be run and it will send the desired commands to the Pixhawk. For testing purposes, the group created a simple Python script called armingtest.py in order to make sure that DroneKit worked as desired. The script armed the motors, waited 6 seconds, disarmed the motors, waited 6 seconds, and repeated the process for a total of 5 arms and 5 disarms.

Originally, the script was run through Mavlink but the commands were not communicated properly. On the monitor, random letters, numbers, and symbols streamed across the screen. It was later discovered that the reason for this was due to the Raspberry Pi trying to communicate with two signals over the same connection. The dual signal over the one USB connection created noise that would not

allow for the flight commands to be sent properly. Eliminating the Mavlink signal instantly fixed the issue; therefore, the group decided to only use Mavlink for initial connection testing.

3.8 Cable Management

Cable management is a critical part of any system that integrates both mechanical and electrical systems. This is particularly important for systems with moving parts or, in this case, the fast moving rotors. We dedicated a few days to plan and route cables throughout the quadrotor to ensure that the cables would remain clear of the rotors despite potential for high wind conditions. After carefully planning where the cables were going to go, we began implementing the wiring scheme, this involved drilling several small holes in the belly and top of the fuselage of the quadrotor UAV as well as shortening several cables to ensure that they could easily and cleanly fit. Once this was complete, we began testing out setup to make sure the solution would stand up to various different flight conditions.

4 Results

There are two main components for the vision-based guidance system to function properly. The first being an efficient, light, image processing program that takes in live video and converts it to path instructions. The second component is successful integration between hardware components, namely the Raspberry Pi microcontroller and Pixhawk flight controller. These systems can function independently but necessarily work in harmony to execute the intended goal.

4.1 Image Processing

Initial image processing code was developed to process one image at a time from a webcam in the C programming language. This version of the code did not function properly, likely due to an incorrect implementation of OpenCV for C. After substantial research, we found that OpenCV versions for Python were more stable and began redeveloping the code in Python syntax. The group saw immediate results with a successful frame capture from the webcam (see Figure 24). After further research, we found examples of video capture and began implementing it into the code as well as a basic image processing algorithm. This integration was seamless and we were quickly able to process a crude image via webcam. With this algorithm in place we began working on perfecting the image processing system and were able to improve the frequency with which the algorithm successfully detected markers. This was done through blurring and contouring filters in OpenCV (see Figure 25). Despite various efforts, marker detection remained inconsistent. However, the group was satisfied with its efforts and continued the development of the processing algorithm.

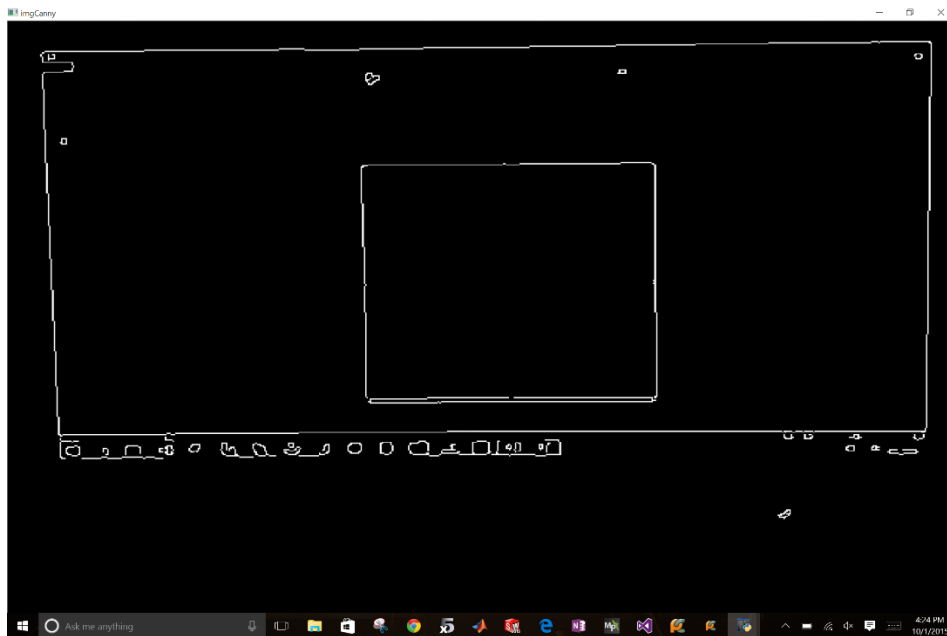


Figure 24: Canny Edge Using Python

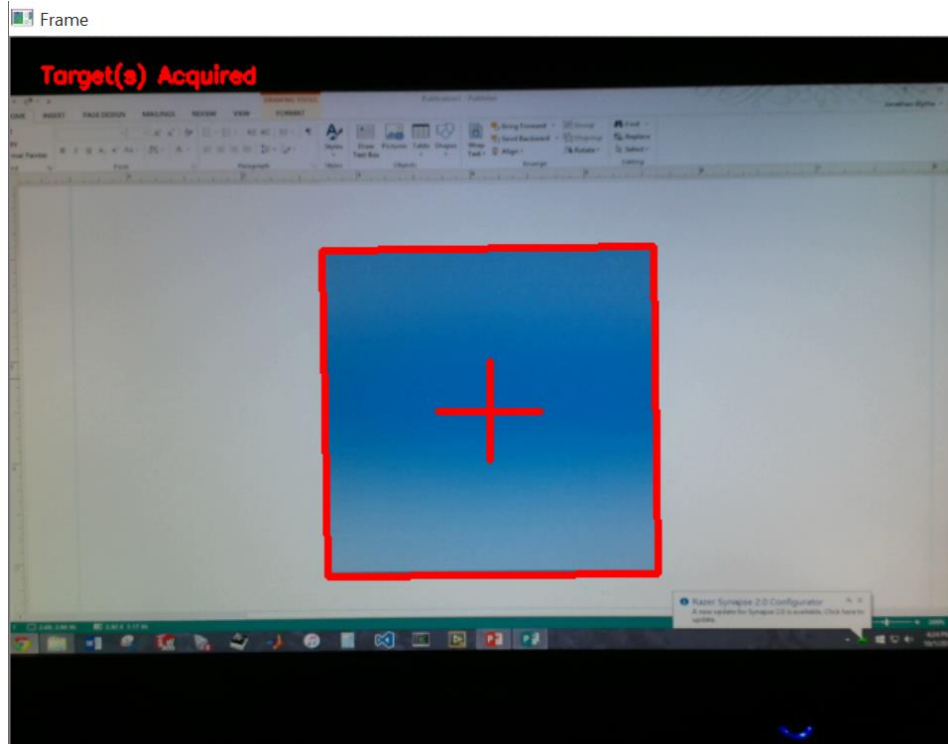


Figure 25: Successful Shape Detection

With satisfactory marker recognition, the group delved into actual path instruction generation. We were successful in making the code generate the pixel location of the marker in the frame, which we were then able to use to create relative location in pixels from the center of the image (see Figure 26). From there we were able to create a function that takes in pixel distances and converts them into real world distances as a function of camera altitude. To prevent the quadrotor UAV from returning to markers that are in its rear field of view, the group developed a series of flight modes and null zones in the image processing algorithm. We were able to successfully get the algorithm to cycle through the searching and fly-to modes as the camera traversed between markers [24]. At this point, the group was successful in optimizing the code to run on a Raspberry Pi and began tackling increased efficiency and use of the Raspberry Pi Camera. After slight changes, we were able to achieve roughly 15 frames per second processing speed, which is functionally equivalent to the image output rate to the Raspberry Pi Camera. This means that the group's code does not substantially hinder the Raspberry Pi's image acquisition.

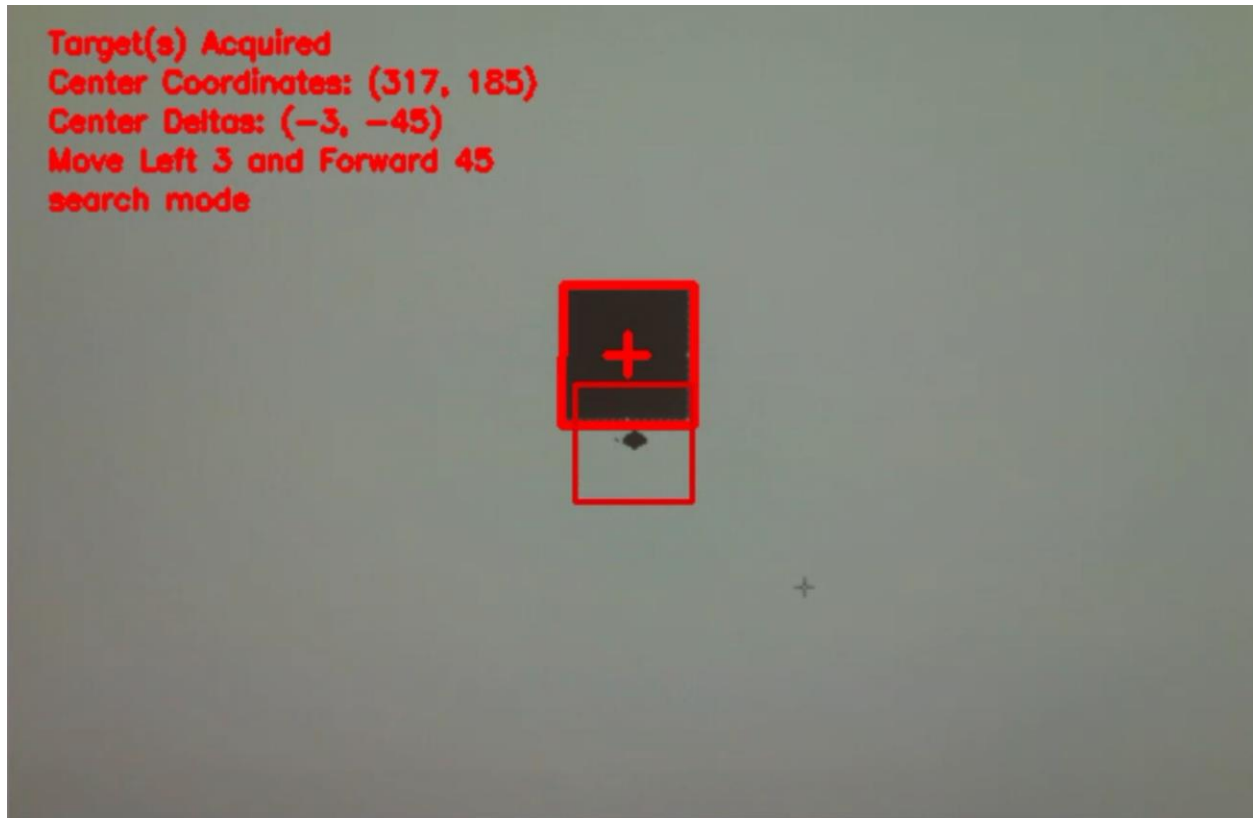


Figure 26: Shape Detection with Pixel Location

4.2 Hardware System Integration

One of the first things the group considered was how to distribute power to onboard hardware. After determining that flight time would be least impacted by a distribution system with a single power source, we proceeded to manufacture a power distribution device. Once adequate power and distribution was confirmed, we began to try to communicate with the onboard flight controller. Based upon the recommendation of online sources, we attempted to connect the serial pinouts from the Raspberry Pi to the Telemetry 2 port on the Pixhawk (see Figure 27) [22, 25]. When attempting testing the connection through Mavlink, the connection failed to completely link. The solution found was to connect the two devices via the USB out on the Raspberry Pi and the micro-USB in on the Pixhawk. Running the same test as with the serial connection, we were able to successfully communicate with the Pixhawk through the Raspberry Pi.

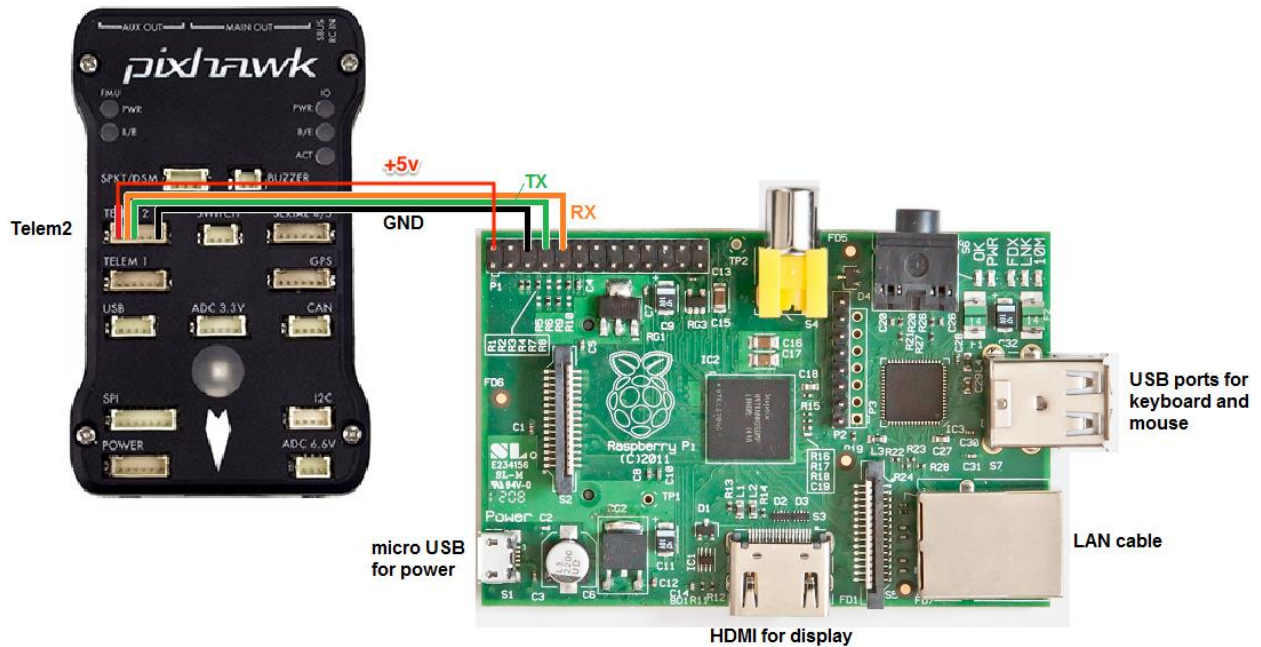


Figure 27: Pixhawk to Raspberry Pi Serial Connection [22]

After ensuring connection success, the team designed and 3D printed custom mounting hardware in order to attach the Raspberry Pi and Pi Camera to the IRIS+. Manual flight test proved that the added weight to the quadrotor UAV did not hinder flight performance nor flight time due to the light weight nature of both the 3D printing material as well as the Raspberry Pi. Once the Raspberry Pi was officially attached to the quadrotor UAV, the team successfully armed the quadrotor UAV motors and had the IRIS+ takeoff through commands in the Raspberry Pi terminal.

The marker detection script was then altered in order to convert marker coordinates into actual flight commands that the Pixhawk flight controller could understand. The specific Python commands were done through the use of Dronekit Application Programming Interface (Dronekit API). At first, communication through Python was unsuccessful. It was discovered that the initial Mavlink connection sent one signal through the USB connection while the Python script sent a second signal through the same connection. The dual signal caused an issue with executing the Python script properly. It was then discovered that the need of Mavlink was not necessary in order to run the Python script successfully. The team was able to arm the motors and initialize the search for markers. [26] The final test resulted in failure to takeoff due to issues with the takeoff command provided by the developers of Dronekit API.

5 Conclusions

During this project, the team was able to develop a guidance and navigation algorithm. Through the use of Python, OpenCV, and a Raspberry Pi microcontroller, the team successfully created an onboard computer that will provide commands and corrections to the quadrotor UAV flight path in real time. The bench and flight tests conducted showed that the shape detection algorithm worked as desired but more tuning and flight tests can increase the accuracy and marker recognition consistency of the ground markers. Due to the built in Python capabilities of the chosen microcontroller, the Raspberry Pi proved to be the ideal candidate for onboard flight control and command. The same holds true for the use of OpenCV due to its abundant support in Python.

Future recommendations include the use of a wireless HDMI connection to a monitor in order to fully see the telemetry and frame captures during flight. Additionally, Mavlink and Dronekit API have wireless capabilities which would eliminate the necessary mounting of the Raspberry Pi as well as the USB connection directly into the quadrotor UAV. Although a wireless system is not required as demonstrated by this project, a comparison of command execution time between the two methods would be notable. Comparing the two would ensure that the most efficient method is utilized. Further system development would enhance the marker recognition capabilities and autonomous flight of the quadrotor UAV.

Appendix A: Raspberry Pi Setup

A Raspberry Pi 2 was used for this project while using the Wheezy version of Raspbian. In order to run the program, Debian Wheezy must be used instead of the newer version Raspbian Jessie. The Raspberry Pi runs off of a micro-SD card which stores all of the files as well as operating system. A webcam may be used but the use of a Raspberry Pi Camera is recommended due to the integrated support on the Raspberry Pi. A minimum of 64GB is required for this project. The following steps layout how to properly setup the Raspberry Pi to run the program and interact properly with the IRIS+. [27]

1. Plug the Micro-SD card into a computer and format it to be FAT32
2. Download and Install Win32 DiskImager from <http://sourceforge.net/projects/win32diskimager/>
Note: This application writes the Raspbian Disk Image to the Micro-SD
3. Download the Raspbian Wheezy zip file from <https://www.raspberrypi.org/downloads/raspbian/>
4. Extract the contents of the downloaded zip file to a file location of your choice
5. Using Win32 DiskImager, write the extracted Raspbian Disk Image to the Micro-SD
6. Eject the Micro-SD from the computer and insert it into the Raspberry Pi
7. Power up the Raspberry Pi
Note: Be sure to have the Raspberry Pi connected to a display with and HDMI cable. In addition, internet is required through Ethernet or WIFI dongle. A keyboard and mouse are also required.
8. Once the Raspberry Pi fully boots, a configuration menu will be seen on the display (if the Raspberry Pi does not display this but goes right to the desktop, go to the terminal and type `raspi-config` and hit enter)
 - a. In the configuration menu, while the first option (Expand File System) is highlighted, hit enter.
Note: This allows for the Raspberry Pi to use the entirety of the Micro-SD. Also, while in the configuration menu you must use the arrow keys on the keyboard to navigate.
 - b. Navigate down the configuration menu to the third option (Enable Boot to Desktop/Scratch) and hit enter
 - i. Select “Desktop Login as user Pi” and hit enter
 - c. Navigate to the fifth option (Enable Camera) and hit enter
 - i. Enable the Raspberry Pi Camera
 - d. Hit the right arrow key twice until the <Finish> button is highlighted and hit enter
Note: This will reboot the Raspberry Pi
9. After reboot, the Raspbian Desktop will appear on the screen. Open the File Manager
10. Put the `QuadrotorUAVCode.py` in the `/home/pi` file location
Note: This file can be put anywhere but the specified location above allows for easier access. Also, the same can be done for `ShapeDetection.py` which can be used for testing purposes.
 - a. Put the `QuadrotorUAVCode.py` file on a USB drive
 - i. Plug the USB drive into the Raspberry Pi
 - ii. Open files and copy `QuadrotorUAVCode.py` to the specified file location above

Now download all of the required libraries and packages through Terminal Commands as root user. In the terminal, type `sudo -i` in order to become root user.

11. Update and upgrade all existing packages

```
$sudo apt-get update
$sudo apt-get upgrade
```

Note: The \$ symbol is not typed by the user. This just denotes a new line in the terminal.
12. Install OpenCV libraries and Packages

```
$sudo apt-get install opencv-dev python-opencv
```
13. Install package that allows for the use of Webcams

```
$sudo apt-get install guvcview
```

Note: This step is not necessary if using a Raspberry Pi Camera

14. Install all packages for Mavlink

```
$sudo apt-get install screen python-wxgtk2.8 python-matplotlib  
python-pip python-numpy
```

15. Enable the use of the serial ports on the Raspberry Pi for communication with the Pixhawk

```
$sudo nano /etc/inittab
```

Note: This is not necessary if connected between devices via USB

- a. Navigate to the very last line of the text and add a # at the beginning of the line, this comments out the last line which should look like this

```
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Note: In order to type #, you cannot use SHIFT-3 but instead must use the \ key

- b. Press CTRL-X and then Y in order to save the change made

16. Install all the required libraries for Mavlink

```
$sudo apt-get install screen python-wxgtk2.8 python-matplotlib  
python-opencv python-pip python-numpy python-dev libxml2-dev  
libxslt-dev  
$sudo pip install pymavlink  
$sudo pip install mavproxy
```

17. Disable the OS control of the serial port

```
$sudo raspi-config
```

- a. Navigate to Advanced Options and hit enter
- b. Navigate to A8 Serial and hit enter
- c. Navigate to Disable and hit enter

18. Install Dronekit API

```
$sudo pip install dronekit droneapi
```

19. Reboot the Raspberry Pi to finalize the installation of all packages and libraries

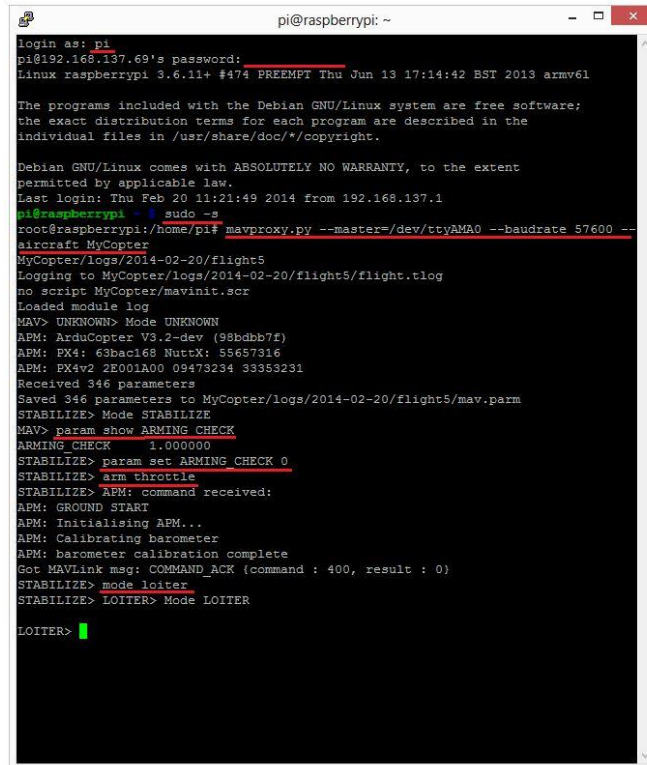
```
$sudo reboot
```

When the Raspberry Pi fully reboots, everything is ready to run. Now the code can be tested by opening the File Manager and navigating to the /home/pi file location (or in which ever location the code was saved). Right click on the QuadrotorUAVCode.py and select Open With: Python 2 IDLE. Note that the code will not run and OpenCV will not be recognized if Python 3 IDLE is used. Once Python 2 IDLE and Python Shell are open, click Run in the IDLE.

Appendix B: Connecting to Mavlink

Once all the required libraries are installed from Appendix B, test to verify that the connection is made between the Raspberry Pi and Pixhawk. Make sure to remove the rotor blades before testing the connection. During initial connection, be sure to have the Raspberry Pi plugged directly into the Pixhawk via USB without keyboard or mouse plugged in. If the USB cable is not plugged into the Pixhawk within 30 seconds, the port on the Pixhawk is disabled. [22, 28, 25] Once the system is booted fully, plug in the keyboard and mouse and navigate to the terminal. In terminal start Mavlink connection by entering the following:

```
$sudo -s mavproxy.py
```



```
pi@raspberrypi: ~
login as: pi
pi@192.168.137.69's password:
Linux raspberrypi 3.6.11+ #474 PREEMPT Thu Jun 13 17:14:42 BST 2013 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Feb 20 11:21:49 2014 from 192.168.137.1
pi@raspberrypi ~$ sudo -s
root@raspberrypi:/home/pi# mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --
aircraft MyCopter
MyCopter/logs/2014-02-20/flight5
Logging to MyCopter/logs/2014-02-20/flight5/flight.tlog
no script MyCopter/mavinit.scr
Loaded module log
MAV> UNKNOWN Mode UNKNOWN
APM: ArduCopter V3.2-dev (98babb7f)
APM: PX4: 69bae168 NuttX: 55657316
APM: PX4v2 2E001A00 09473234 33353231
Received 346 parameters
Saved 346 parameters to MyCopter/logs/2014-02-20/flight5/mav.parm
STABILIZE> Mode STABILIZE
MAV> param show ARMING CHECK
ARMING_CHECK 1.000000
STABILIZE> param set ARMING_CHECK 0
STABILIZE> arm throttle
STABILIZE> APM: command received:
APM: GROUND START
APM: Initialising APM...
APM: Calibrating barometer
APM: barometer calibration complete
Got MAVLink msg: COMMAND ACK (command : 400, result : 0)
STABILIZE> mode loiter
STABILIZE> LOITER> Mode LOITER
LOITER> █
```

Figure 28: Mavlink Screenshot [22]

When the connection is made, the terminal should look like Figure 10 up until “STABILIZE> Mode STABILIZE”. To arm the throttle, type “arm throttle” and hit enter. The motors should start spinning. To disarm, type “disarm throttle” and hit enter. The motors should turn off. If the disarm command is not entered within 15 seconds, the motors will automatically turn off due to the safety protocols built into the Pixhawk. If this test is successful, then the Pixhawk and Raspberry Pi connection is solid and the code can be run fully.

Appendix C: Source Code and Testing Video Links

Source Code on BitBucket:

https://bitbucket.org/rvcowlagi/src1_mqp/src/41c23302270e002a4e46b1c43ec291646437fe71/MQP2016/Autonomous_Quadrotor/?at=master

YouTube Video Testing Library:

<https://www.youtube.com/channel/UCQRa7QE0rmnZ8fNfBzzHfJA>

Appendix D: References

- [1] C. G. Daugherty, Interviewee, *UAV use for Search and Rescue*. [Interview]. November 2015.
- [2] V. Dr. Kumar, *Robots that fly ... and cooperate*, TED, 2012.
- [3] V. Dr. Kumar, "Vijay Kumar Lab," 20 March 2016. [Online]. Available: <http://www.kumarrobotics.org/research/>.
- [4] R. Dr. D'Andrea, *The astounding athletic power of quadcopters*, TED, 2013.
- [5] R. Dr. D'Andrea, "Flying Machine Arena Research," 2015. [Online]. Available: <http://flyingmachinearena.org/research/>.
- [6] S. Friedman, K. Hancock and C. Ketchum, "Vision-Based Obstacle Avoidance for Small UAVs," Worcester Polytechnic Institute, Worcester, 2015.
- [7] R. V. Dr. Cowlagi, *Supplementary Lecture Notes (Aircraft Dynamics and Control)*, Worcester Polytechnic Institute, 2015.
- [8] MicrocontrollersAndMore, "OpenCV Tutorial," 20 September 2015. [Online]. Available: https://github.com/MicrocontrollersAndMore/OpenCV_3_Windows_10_Installation_Tutorial.
- [9] Itseez, "OpenCV (Open Source Computer Vision)," 2016. [Online]. Available: <http://opencv.org/>.
- [10] 3DR, "Pixhawk Overview," September 2015. [Online]. Available: <http://copter.ardupilot.com/wiki/common-pixhawk-overview/>.
- [11] PX4 Autopilot, "Pixhawk Flight Controller," 5 November 2015. [Online]. Available: <https://pixhawk.org/choice>.
- [12] Raspberry Pi, "Raspberrypi 2," 7 December 2015. [Online]. Available: https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB_-comp.jpeg.
- [13] Raspberrypi, "Raspberrypi Camera," 7 December 2015. [Online]. Available: <https://www.raspberrypi.org/wp-content/uploads/2014/03/raspberrypi-camera-module.jpg>.
- [14] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, "Universal Serial Bus," 27 April 2000. [Online]. Available: http://sdphca.ucsd.edu/lab_equip_manuals/usb_20.pdf.
- [15] Laser Focus World, "3d Printing," 12 January 2016. [Online]. Available: <http://www.laserfocusworld.com/content/dam/lfw/printarticles/2014/08/1408LFW03f1.jpg>.
- [16] M.-P. Spierer, Artist, *Raspberrypi 2 (and B+) case with VESA mounting*. [Art]. GrabCAD.
- [17] E. Yam, Artist, *IRIS+ Quadcopter Drone*. [Art]. GrabCAD.
- [18] Federal Aviation Administration, *Unmanned Aircraft Systems (UAS) Regulations & Policies*, Federal

Aviation Administration, 2015.

- [19] Ardupilot, "Mission Planner Home," 2015. [Online]. Available: <http://planner.ardupilot.com/>.
- [20] A. Rosebrock, "Finding Shapes in Images using Python and OpenCV," 20 October 2014. [Online]. Available: <http://www.pyimagesearch.com/2014/10/20/finding-shapes-images-using-python-opencv/>.
- [21] A. Rosebrock, "Target acquired: Finding targets in drone and quadcopter video streams using Python and OpenCV," 4 May 2015. [Online]. Available: <http://www.pyimagesearch.com/2015/05/04/target-acquired-finding-targets-in-drone-and-quadcopter-video-streams-using-python-and-opencv/>.
- [22] Ardupilot, "Communicating with Raspberry Pi via MAVLink," Ardupilot, [Online]. Available: <http://dev.ardupilot.com/wiki/raspberry-pi-via-mavlink/>. [Accessed December 2015].
- [23] Dronekit, "Welcome to DroneKit-Python's documentation!," Dronekit, [Online]. Available: <http://python.dronekit.io/>. [Accessed 8 December 2015].
- [24] Systems & Robot Control Laboratory @WPI, "MQP 1601 - Demonstration of Marker Recognition On board IRIS+," Worcester Polytechnic Institute, March 2016. [Online]. Available: https://www.youtube.com/watch?v=pm8--_zf4LI.
- [25] Ardupilot, "PX4 Development Guide," Ardupilot, [Online]. Available: <http://dev.ardupilot.com/wiki/raspberry-pi-via-mavlink>. [Accessed 2015].
- [26] Systems & Robot Control Laboratory @WPI, "MQP 1601 - Execution of Simple Python Script using Dronekit API," Worcester Polytechnic Institute, March 2016. [Online]. Available: <https://www.youtube.com/watch?v=XIAoLKoBLZk>.
- [27] Raspberry Pi Foundation, "Getting Started with NOOBS," Raspberry Pi Foundation, [Online]. Available: <https://www.raspberrypi.org/help/noobs-setup/>. [Accessed 2015].
- [28] QGroundControl GCS, "MAVLink Micro Air Vehicle Communication Protocol," MAVLink, 8 December 2015. [Online]. Available: <http://qgroundcontrol.org/mavlink/start>.
- [29] Ardupilot, "Initial Configuration," PX4 Development Guide, [Online]. Available: <http://dev.px4.io/starting-initial-config.html>. [Accessed 2015].