Fiducial Pattern Tracking Drone

By

Alexander Amiel (aamiel2)

Angelos Guan (tguan2)

Umer Belagam (belagam2)

Final Report for ECE 445, Senior Design, Spring 2021

TA: Chaitanya Sindagi

May 5, 2021

Project No. 59

Abstract

For our project, we designed a hands free controllable drone with the use of an RC receiver and transmitter for take off and landing purposes. The Drone's flight controller is based on BetaFlight code which can take commands from an arduino, through PPM signals, to change the drone's speed and orientation while keeping the drone oriented upright even when met with various weather conditions. Through the use of detection and distance algorithms run on the Raspberry Pi, and video capture done by the camera module, we are able to detect a fiducial pattern, which is similar to a QR code, light up an LED to signal the user that it has been detected, and keep the drone a certain distance from it. The Raspberry Pi sends distance measurements to an arduino through two wires to then communicate with the flight controller. Further improvements could be done with the tuning of various inputs in the BetaFlight software and the use of a different inertial measurement unit (IMU).

Contents

1. Introduction

1.1 Background and Overview

The goal of our project is to develop an equipment-free drone guided by a fiducial pattern. A fiducial pattern is similar to a QR code except it is not as complex and can be detected less computationally costly [1]. The drone is able to detect and track the pattern, maintaining a set distance from it at all times.

The motivation for our project is to provide an alternative solution to the problem of controlling the drone without a physical controller. Currently, the most popular controlling method is to use mobile phones to keep track of location via Bluetooth and monitor the drone's perspective vision via camera. Such an approach is limited by the phone and events such as the phone dying, which may trigger unwanted behaviors for the drone. Another alternative way to pilot a drone without physical controllers are through the use of hand gestures, DJI's spark [2] and Samsung [3], and through the use of a muscular-control system, MIT's artificial intelligence lab [4]. The problem with hand-gestured controlled drones is that it may detect someone else's hand and may result in undesired behaviors. In MIT's muscular-based control approach, problems arise when the person piloting the drone has a muscular breakdown or accidentally squeezes too hard or too soft.

Our solution provides a new way to pilot the drone by detecting a given fiducial pattern and maintaining a distance from it. It is more cost efficient hardware-wise than app-controlled drones because it does not require the user to have a specific device that can run the software, and more reliable than gesture-controlled drones. The user holds the fiducial pattern and guides the drone by moving the fiducial pattern to the desired position. If no pattern is detected, the drone will signal the user through a buzzer and LED. Our method provides a controller-free approach to reliably pilot a drone while solving the problem of confusing other people's hands as a user input.

1.2 High Level Requirements

The high level requirements of our project:

1. Our project must be able to use a camera mounted on the drone to detect and track a fixed sized fiducial pattern from 1-10 feet away, light up an LED, and estimate the real-world position coordinate of the detected pattern within 3% mean squared error.
2. The drone must be able to fly with an upright orientation and maintain 4 to 5 m to the detected fiducial pattern within 15% steady-state error and less than 5 s response latency. If a fiducial pattern is not detected, the drone should maintain its current position and ring a buzzer to notify the user.
3. The drone must be able to halt in place if there are any obstacles detected in 0.2 m range. Unfortunately, we did not integrate an ultrasonic sensor subsystem as it was hard

to adjust the drone's flight controller software. If given more time, this will be integrated into our project.

1.3 Block Diagram

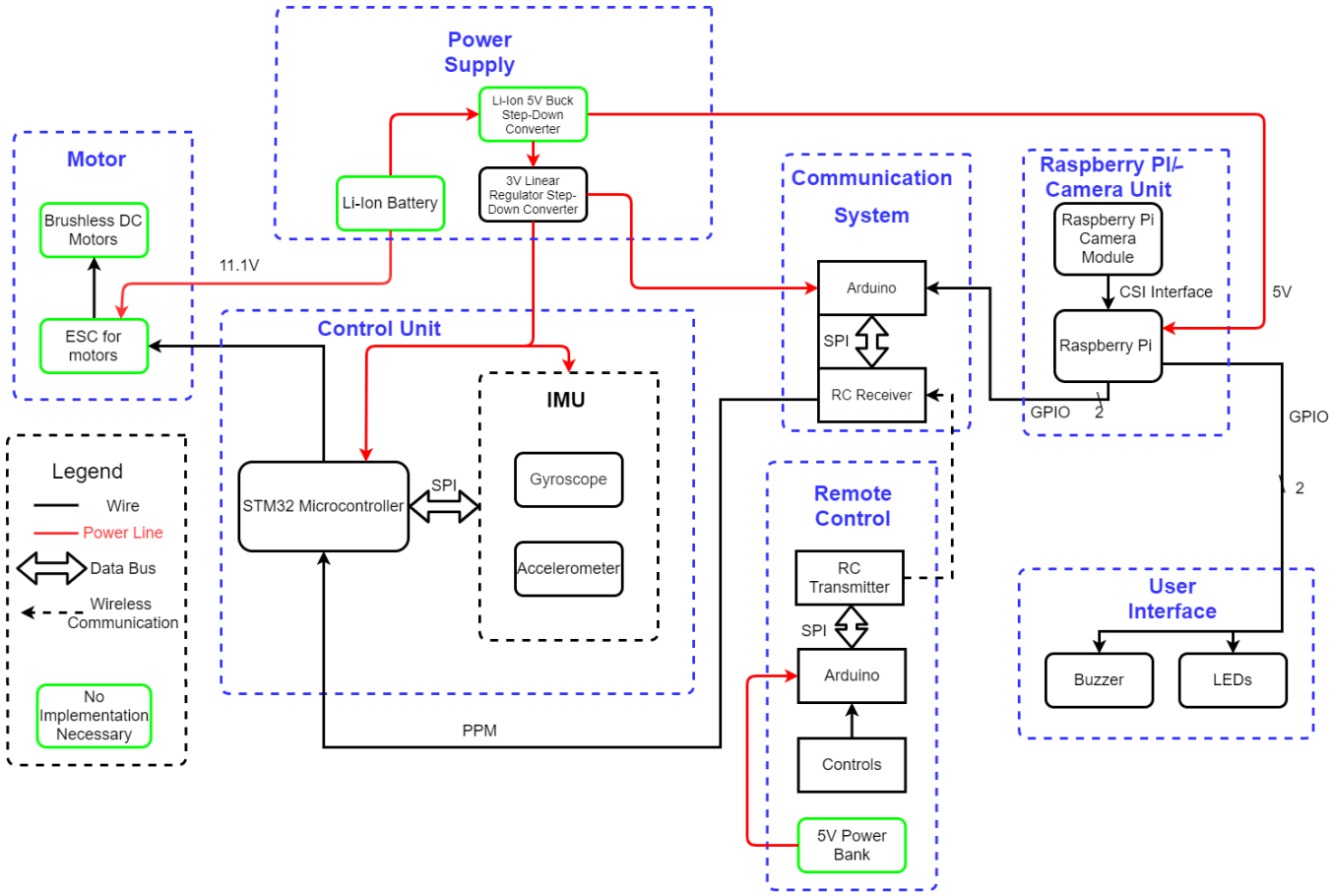

Figure 1: Block Diagram

Our design contains 7 subsystems: motor unit, power supply, control unit, communication system, Raspberry Pi/camera unit, user interface unit, and remote control unit. The motor unit is made up of 4 brushless DC motors and their corresponding ESCs. The power supply contains a 3S Li-ion battery, a 5 V buck step-down converter, and a 3 V linear regulator step-down converter. The control unit contains a STM32 that is running betaflight software, and sensors (IMU) that supports the flight controller. It takes input from the communication unit and sends output to the ESCs in order to control the motors according to commands it receives from the arduino. The communication unit contains an arduino board and a RC receiver. It is listening to RC signals coming from user input via remote control as well as output from the Raspberry Pi/camera unit. It will process control signals from the user and camera detection and send a combined final command to the control unit. The Raspberry Pi/camera unit is responsible for running the detection algorithm to determine which direction to move the drone according to the relative position of the fiducial pattern from the camera. The user interface unit contains a

buzzer and LEDs, which both serve as ways to indicate the status of the drone to the user. We added a remote control unit with a custom-designed controller and RC transmitter in order to test the drone safely in case other subsystems fail. It is also responsible for manually taking off and landing. It sends command signals to the communication unit and lets the communication unit decide what final commands to send to the flight controller.

We have made several changes to our block diagram from the design document. We added a communication unit and remote control unit for testing and for sending command inputs to BetaFlight running on the control unit. In addition, we deleted the pressure sensor and ultrasonic sensors from the control unit.

2. Design

2.1 Power Supply

In Figure 1, the power supply subsystem is mainly based on the 3S Li-Ion battery pack with a nominal voltage of 11.1 V and 5200 milliamp hour (mAh) energy capacity, a 5 V buck step-down converter, and a 3 V linear regulator step-down converter. We chose the 3S Li-ion battery pack because we based our drone on similar size drones, where most of the time the drone is spent hovering, and has a current draw of 25 A. The estimated flight time is

$$5.2\ \text{Ah} * 60\ \text{min} * 80\%\ \text{battery drain} * 1/25\text{A} = 10\ \text{min} \quad (2.1)$$

10 minutes of flight time. The ESCs are directly powered by the battery pack, a supplied 11.1 V.

2.1.1 Use of 5 V Buck Step-Down Converter

The Raspberry Pi is powered by 5 V at 2.5 A. The Li-ion battery pack has a wire connected to the 5 V buck step down converter which then has a wire that goes to the Raspberry Pi's 5 V power pin [See Appendix B and Figure 18]. We chose to use the PDB-XT60 distribution board because it can step down 12 V, from the battery pack, to 5 V to power the Raspberry Pi. The Raspberry Pi we use is the Raspberry Pi 4 which has a recommended current capacity of 2.5 A, and this breakout board can output a continuous max current of 2.5 A. We initially used the ADP2303ARDZ-5.0-R7 chip, but when verifying the voltage, we could only output 1 V after 5 seconds of stabilization [See Appendix A for Voltage Verification].

2.1.2 Use of 3 V Step-Down Converter

The STM32 microcontroller, arduino, and IMU are powered by 3 V from the 3 V linear regulator step-down converter. The microcontroller we use is the STM32F405RGT6W chip which has an operating voltage from 1.8 V to 3.6 V [See Appendix A for Voltage Verification].

2.2 Flight Controller

The flight controller is the most important component of our drone and it is what enables the drone to fly. It has a microcontroller that reads and processes the data from the IMU and takes commands from the communication system and uses those to control the ESCs.

2.2.1 Flight Controller Software

During the design phase we had considered using an open source flight control software for our flight controller (FC) and after looking at a few different ones, we selected BetaFlight [5] as it has support for the microcontroller we chose and a variety of different sensors. BetaFlight is responsible for running the PID loop with the desired values for yaw, pitch roll and throttle as

input, and updating the corresponding output voltage to ESC according to the IMU sensor values. We did the PID tuning via BetaFlight's GUI by adjusting the gains shown in Figure 2.

Figure 2: PID tuning in BetaFlight

Betaflight has a list of commercially available flight controller boards that you can select from to flash the software on it but since our flight controller was custom we had to modify the source code and compile it. Since the microcontroller we used is common among flight controllers, there was not much modification needed to the code. We had to modify the pin mappings and select what features and drivers to include in the compiled binary. The biggest change we made was add the driver for the IMU we are using as it is not supported by Betaflight. For this we modified the driver for another IMU, the ICM20649 which is from the same manufacturer so most of the SPI registers and addresses were the same.

2.2.2 Microcontroller

We use STM32F405RGT6W microcontroller as it is compatible with Betaflight. It has 1MB flash memory which is enough for Betaflight with the drivers and features we required. It has a FPU (floating point unit) which is required to make fast calculations on the IMU data. It has DMA (direct memory access) which is necessary for faster communication with the ESCs as they can directly read the controls for speed without the STM32 CPU in between. It runs at 168MHz which is more than enough to run the PID loop at 9KHz which is the maximum rate the IMU can output data.

2.2.3 IMU

The IMU, which stands for inertial measurement unit, contains the gyroscope and accelerometer sensors. The gyroscope measures angular velocity and the accelerometer measures acceleration which includes the acceleration from gravity. Both these sensors give a reading in all 3 axes and with this information, we can calculate how fast the drone is rotating and its orientation. This data is fed into a PID control loop to make sure the drone is moving as intended by correcting any inconsistencies between how it is supposed to move from the commands received and how it has actually moved as determined by the sensors. This correction is also performed if the drone moves from any external forces like wind.

The IMU we chose is the ICM20948 which also includes a magnetometer which is basically a 3D compass that can determine the north direction. Initially, we intended on using the magnetometer in our design so we could maintain the direction the drone points towards, but we were not able to get it working with the software, so now the drone is subject to slight drifts in its direction. We chose this IMU because it has a fast refresh rate and a high sensitivity which results in smoother flying according to our initial research. The gyroscope operates at a 9 KHz refresh rate and its range can be set at ± 250, 500, 1000 or 2000 dps (degrees per seconds). We set it at ±1000 dps as per our requirements. The accelerometer operates at a 4.5 KHz refresh rate and its range can be set at ± 2, 4, 8 or 16 G (gravity in m/s). We set it at ±8 G as per our requirements. The IMU was initially going to communicate with the STM32 using I2C and we designed our PCB around that but we could not get the I2C driver working in Betaflight so we used SPI instead with jumper wires connected to other unused pins on the PCB

The data from the IMU is noisy and after trying out different filter types and parameters we have our final settings in Figure 3. Based on the model of our IMU and the size of our drone we knew that a low pass filter with low cutoff frequencies would work better. We got to the exact values by monitoring the IMU data and reducing the noise when the drone was stationary, and by seeing how much the drone wobbles during the flight and trying to minimize that.

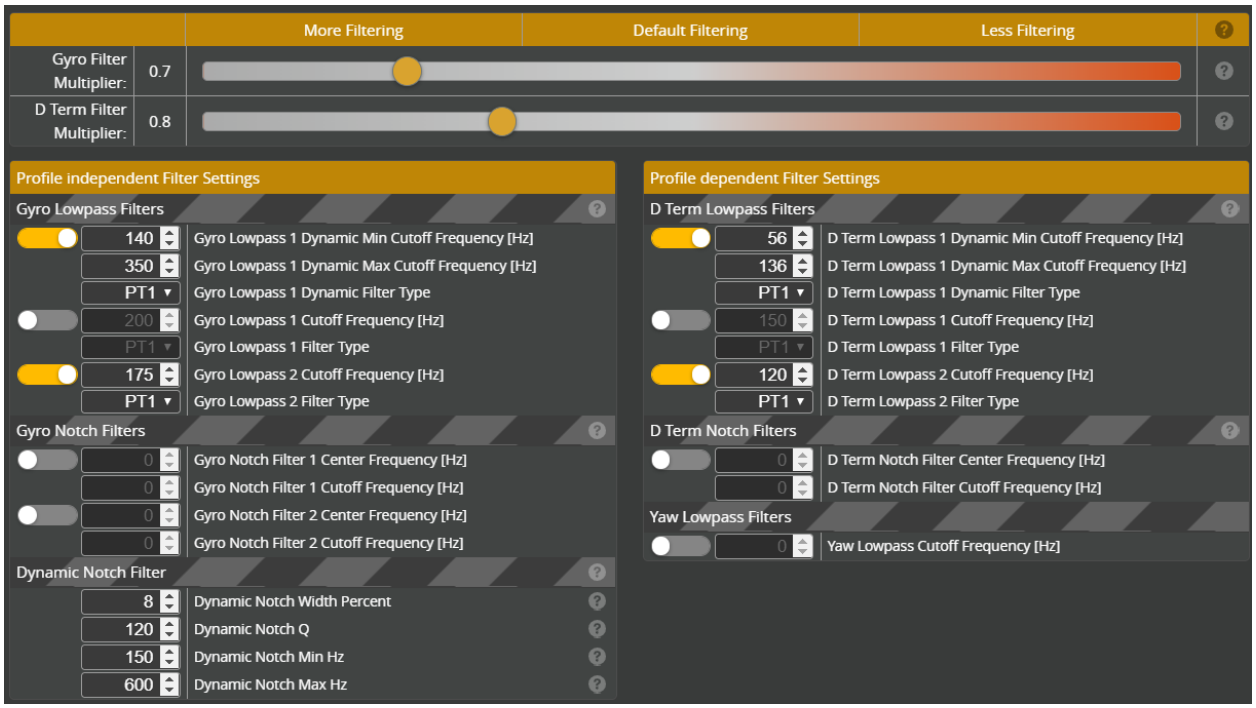

Figure 3: Filter settings in Betaflight

2.3 Raspberry Pi

The purpose of the Raspberry Pi is to detect an AprilTag, which is a specific set of fiducial patterns, calculate the distance from it, and signal an arduino how far the drone is away from the AprilTag. We use python's OpenCV library to capture the AprilTag in real time, and measure the distance to it with each frame of the captured video. We use another python package called "apriltag" which can specify a specific family of AprilTags to detect.

We use a Raspberry Pi because we can run our detection algorithm without having to type in a command to run it, and the communication procedure to signal the arduino is easy using the GPIO pins of the Raspberry Pi. Our group is also familiar with the functionality of the Raspberry Pi, so it seemed as the most logical choice to use for object detection.

Another consideration to use the Raspberry Pi and its camera module is that setting it up on the drone is easy using zip ties. Figure 4 shows how the Raspberry Pi is oriented on the drone, and where the camera is placed.

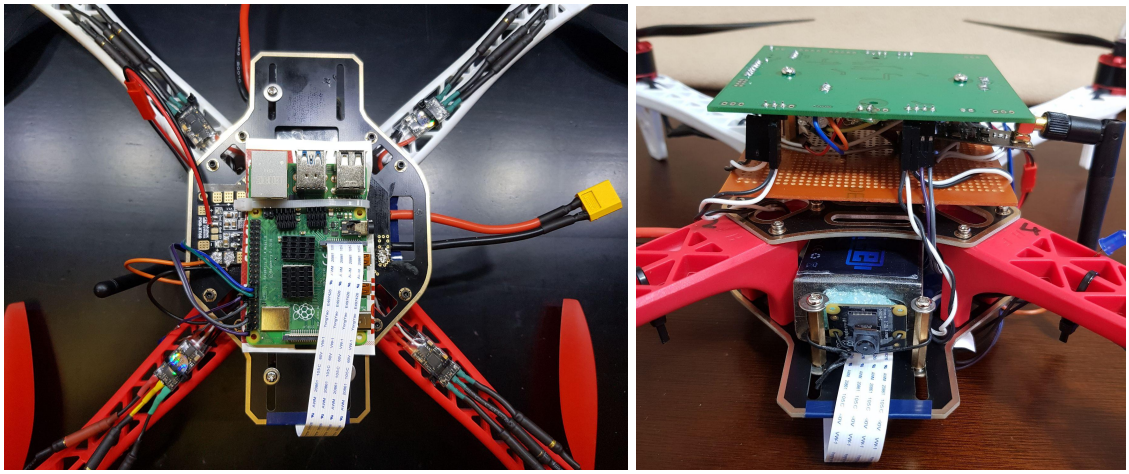

Figure 4: Raspberry Pi and camera placement on the drone

Zip tie the Raspberry Pi to the bottom of the base of the drone, orient the camera to face forward, and zip tie it to two mounts that are connected to the base of the drone.

2.3.1 AprilTag Detection

We detect the AprilTag in real time using the Raspberry Pi's camera module as our video capture system. The camera module connects to the Raspberry Pi through the CSI interface. The AprilTag we use is shown in Figure 5.

Figure 5: AprilTag family Tag36h11

This is the most generic AprilTag family. Since generating an AprilTag is sufficiently harder, we use an already generated AprilTag. The flowchart in Figure 6 shows how the AprilTag is detected.

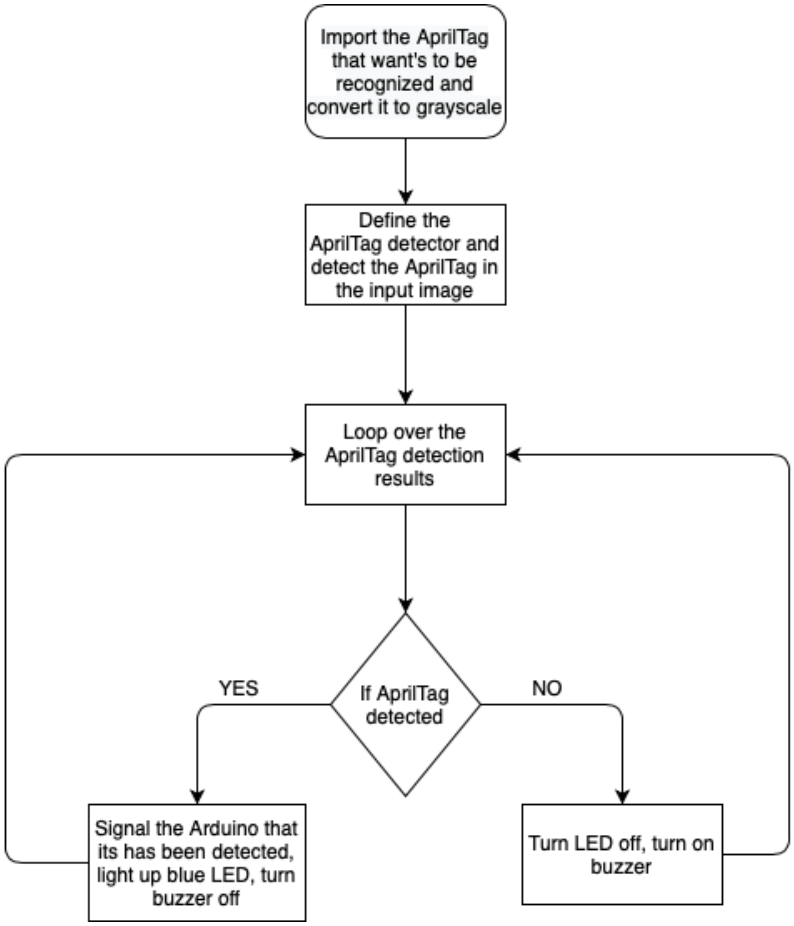

Figure 6: flowchart of AprilTag detection

The first block in the flowchart is the input image from the video capture using python's OpenCV package. The second block uses python's apriltag package to specify which family of AprilTags wants to be detected. In our case, family Tag36h11 is specified. The result is used by combining the input image and the detector to detect the AprilTags in the image, and it returns how many are detected. The third block loops through all the detected AprilTags of the specified family. We only use one AprilTag, so this loop only happens once.

2.3.2 Determining Distance from the Camera to the AprilTag

After detecting the AprilTag, determine the distance it is from the drone. To do this, use Raspberry Pi's camera module and the same detection algorithm from Figure 6.

The first step is to calibrate the focal length of the camera with an input image. The input image has the AprilTag in it and it is a fixed distance from the camera. It also has a fixed width, which can be measured with a ruler. The fixed width is the length of one of the green edges shown in Figure 11 in section 3.1.1. The perceived width, is the width of the image in pixels. To get the perceived width, output the coordinates of each corner point in the AprilTag input image,

calculate the Euclidean distance of each edge, and return the average width. The equation for calculating the focal length is

$$F = (PXD)/W \tag{2.2}$$

where F is the focal length, P is the perceived width, D is the fixed distance between the camera and the AprilTag, and W is the fixed width of the AprilTag.

Knowing the focal length from Equation (2.2), calculate the distance the AprilTag is from the camera in each frame of the video capture. The equation for the distance calculation is

$$D' = (WXF)/new\_P \tag{2.3}$$

where D' is the new distance and new_P is the new perceived width of the AprilTag. The new perceived width calculation is done by taking the max euclidean distance of each edge of the AprilTag. Do this instead of taking the average to take in account if the AprilTag is tilted one way from the center. Initially, we only measured the widths of the top and bottom edges, but this resulted in an inaccurate calculation of the distance.

2.4 Communication System and Remote Control

The communication protocol between each subsystem is shown in Figure 7.

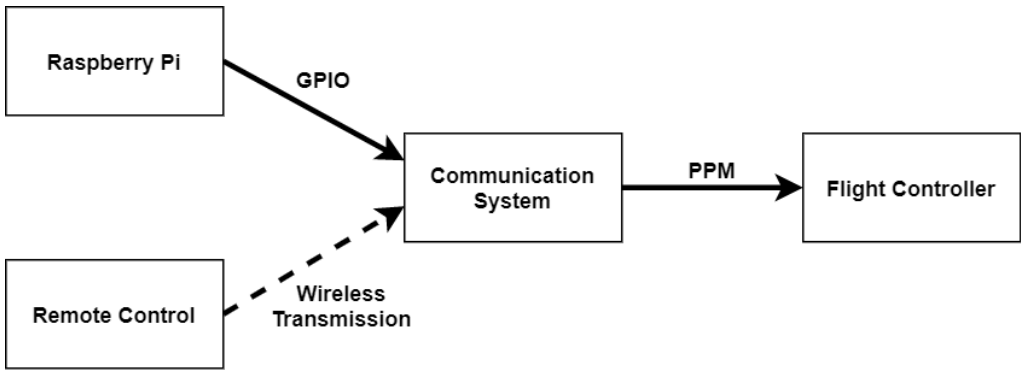

Figure 7: Communication protocols

2.4.1 Drone Movement

The drone has 3 axes of rotation, yaw, pitch and roll which is shown in Figure 8. These are used to maneuver the drone and the throttle is used to increase the speed of the motors to make the drone go in the direction it's facing. The flight controller software expects each of these as inputs with a value between 1000 and 2000. The yaw, pitch and roll have a center value of 1500 and the throttle starts at 1000 and goes to 2000. For our project, we use the radio controller to control the throttle and bring the drone up in the air. Once it's in the air, the Raspberry Pi takes over as the control. Since our drone tracks the AprilTag in front of it, we are mainly concerned with the pitch value which tilts the drone towards the front and back.

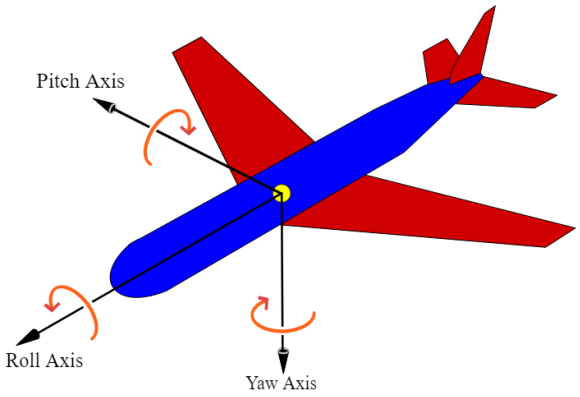

Figure 8: Three axes of rotation

2.4.2 Remote Control

The remote control consists of an Arduino Uno and a NRF24 transceiver module [See Appendix B and Figure 20]. It takes in 5 inputs and transmits 6 values to the communication system. The 6 values sent are yaw, pitch, roll, throttle, aux1 and aux2 and they are sent as a 6 byte data packet. Each byte encodes a value from 1000 to 2000 that is mapped linearly from the byte values of 0 to 255. The two extra values are used for setting modes. Aux1 arms the drone for flying and aux2 sets angle mode which restricts the drone's angle to 25 degrees from the horizontal plane and both of those are binary inputs. Above 1500 sets aux1 and aux2 to high, and below 1500 sets them off. The yaw, pitch and throttle are controlled using 3 potentiometers. Aux1 and aux2 are set by connecting a jumper to either ground or 5 V. The roll is not changed and always set to 1500. Figure 9 shows the sensitivity curves for each axis as well as the throttle presented in the BetaFlight GUI.

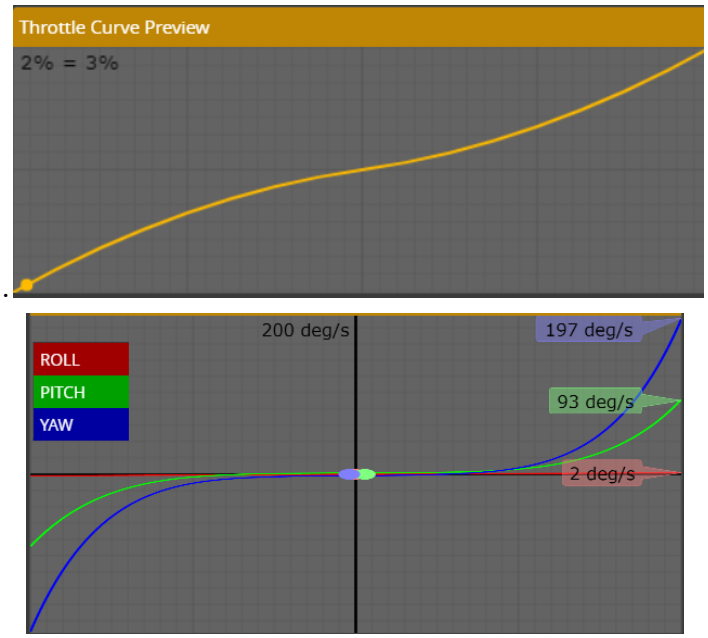

Figure 9: Throttle and 3 axis control sensitivity curves

2.4.3 Communication System

The communication system is the interface between the flight controller and the Raspberry Pi, and it also receives commands from the radio remote control. It consists of an Arduino pro mini and a NRF24 transceiver module on a stripboard that is mounted on the drone [See Appendix B and Figure 21]. This was not part of our original design, but we added it because we were not able to implement autonomous take off and landing for our drone, so this system lets us manually do that. Betaflight does not support serial connection by default, so we are using PPM communication as a substitution. We are using an arduino instead of communicating between the Raspberry Pi and the STM directly because PPM communication requires high accuracy on the timer. We need a microcontroller dedicated to communication without running other high demand programs such as detection algorithms. In addition, using PPM is not reliable on the Raspberry Pi due to strict timing requirements that could not be guaranteed with the linux system running on the Raspberry Pi.

The Arduino is connected to the NRF24 using SPI and when the receiver receives the data from the remote controller, it decodes the data and converts it to a PPM signal to send to the flight controller. The arduino also monitors the input from the Raspberry Pi, and if it is receiving a command to move forward, then the pitch value is increased by 150 before converting it to the PPM signal. If the command is to move backward, the pitch value is decreased by 150. When a radio signal is present, it has the highest priority so that we can always make sure of the safety of the drone during testing. When the radio signal is lost, the arduino will send preset safe values to betaflight to maintain the drone at the same position. The flowchart in Figure 10 shows when data is transmitted to the flight controller.

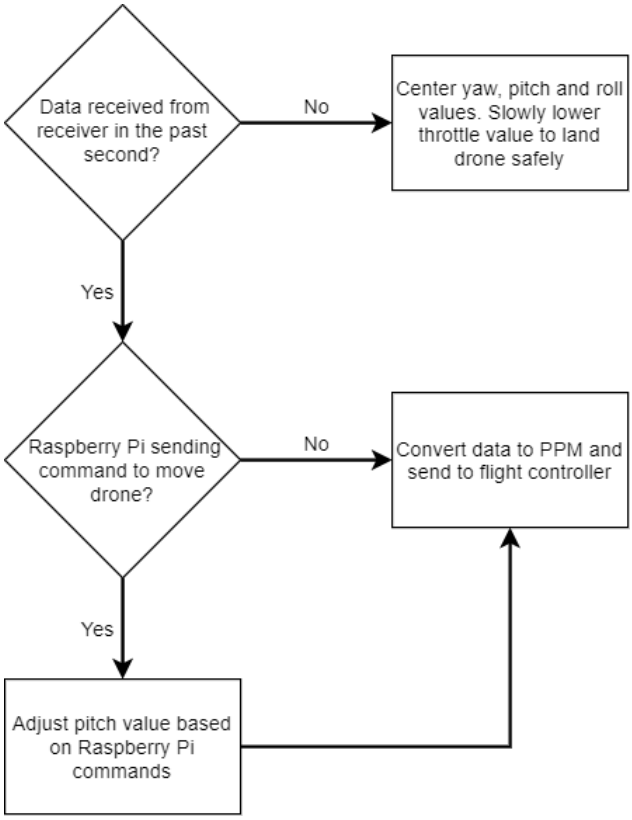

Figure 10: Data transmission flowchart

3. Verification

3.1 AprilTag Verification

We verify the AprilTag detection using two methods. The first method uses the apriltag package to successfully detect the specific AprilTag family, and we draw a bounding box around it, seen in section 3.1.1. To go along with this method, we set up a simple LED circuit which is signaled from the Raspberry Pi's GPIO pins to light up the LED if there is an AprilTag detected, see section 3.1.1. We also have the same circuit for a buzzer to verify that if the AprilTag is not detected, use the GPIO pins to sound off the buzzer. These two tests run simultaneously.

We then verify the distance measurement from the camera to the AprilTag by calibrating the focal length of the camera, mease the distance to the AprilTag, and print the result, in feet, to the screen while running the same LED circuit except signal the LED through distances instead of the number of tags detected [See Section 3.1.2].

3.1.1 AprilTag Detection Methods and Tests

From Figure 6 from section 2.3.1, the third block which loops over the detection results from the input image, the apriltag package can specify the corner points of the detected AprilTag [6]. We get the x and y coordinates of each point and draw a line, using OpenCV, from each point to each other point, that is not on a diagonal, to create a bounding box. Figure 11 shows the result of this.

Figure 11: Bounding box result of the detected AprilTag

The red dot in the middle is the center point of the AprilTag, and we draw this to have a clearer picture when testing if the AprilTag is detected.

We implement a simple LED circuit to use as another verification to detect the AprilTag. The results of the AprilTag detection algorithm has a list of how many AprilTags are detected, so if there are multiple detections, the for loop would run for each detection. Each frame captured also returns how many AprilTags are detected. If there are more than 0 detections, use the GPIO pins of the Raspberry Pi connected to a breadboard to light up an LED [See Appendix B and Figure 18 for Pinouts of the Raspberry Pi].

Connect a wire from a GPIO pin to a breadboard with a blue LED, and use the RPi.GPIO python package to send a high signal to the GPIO pin which would then light up the LED if the number of tags detected is greater than 0 [Look at Appendix A for verification]. We use a blue LED because it has a forward voltage of 3.3 V, so no resistor has to be connected in series to reduce the voltage powered from the Raspberry Pi. To make sure the LED test is working, connect a buzzer to a different GPIO pin, and it will go off if there are zero AprilTags detected. These two tests run simultaneously, and it confirms that our detection technique worked.

3.1.2 AprilTag Distance Calculation Tests

The focal length calibration is verified with an input image that has the AprilTag in the image. For our calibration, the fixed distance from the AprilTag to the camera is 24 inches and the fixed width of the AprilTag is 5.9 inches. When calculating the perceived width, we get an average pixel length of 136 pixels. The result of the focal length calculation using Equation (2.2) is

$$F = (136.19*24)/5.9 = 554 \qquad (3.1)$$

where F is the focal length of the camera. The new distance calculation with the known width and focal length using Equation (2.3) is

$$D' = ((5.9*554)/new\_P)/12 \qquad (3.2)$$

where D’ is the new distance and new_P is the new perceived width of the AprilTag. The division by 12 is done to convert the distance to feet. To test for the distance, print the calculation to the screen along with the bounding box of the AprilTag. Figure 12 shows the results of this.

Figure 12: Distance calculation test results

The top left figure shows the distance of the AprilTag when it is positioned upright and straight from the camera, the bottom left figure shows the distance when the AprilTag is rotated, and the bottom right figure shows the distance when the AprilTag is tilted forward. To verify that our distances are accurate, we set up a tape measure to compare the actual distances to the measured distances. We tested this at increments of 1 feet, and our results are shown in Appendix B and Figure 19.
The further the AprilTag is from the camera, the higher the error percentage is. The error percentage calculation is

$$\text{Error \%} = ((\text{Actual} - \text{Measured})/\text{Actual}) * 100 \qquad (3.3)$$

The max error percentage we calculated is 3% at 10 feet. Since we use a ruler to determine the distance the AprilTag is in the input image when calibrating the focal length and a ruler to measure the width of the tag, there are small human errors associated with this. This is why there is an error percentage of 3%. We could reduce this error by recalibrating the focal length with different input images to get a different perceived width, but there will always be human errors involved in measurement readings.

The Raspberry Pi also has to detect if the drone is 4 to 5 feet away from the camera, greater than 5 feet, and smaller than 4 feet away [See Appendix A for Verification]. The same LED circuit is used as in section 3.1.1 except this time it would send a high signal to the GPIO pin for each

of the distance measurements. We ran the distance measurement verification, as shown in Figure 12, with the LED circuit. We did three different tests, one for each distance measurement.

3.2 Communication Verification

We verified that the communication system is working and can send commands to BetaFlight based on the results of the distance algorithm, from sections 2.3.2, received from the Raspberry Pi. Figure 13 shows what happens to the pitch value in BetaFlight when the AprilTag is further than 5 feet from the camera.

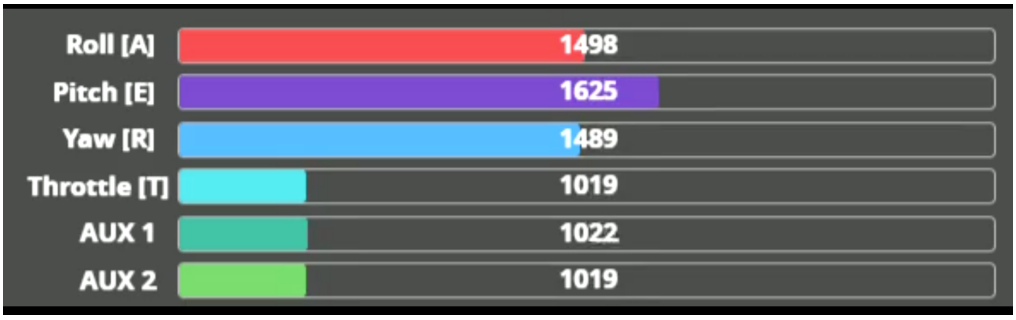

Figure 13: BetaFlight pitch value when the drone is further than 5 feet from the AprilTag

Figure 14 shows what happens to the pitch value when the AprilTag is within 4 feet of the camera.

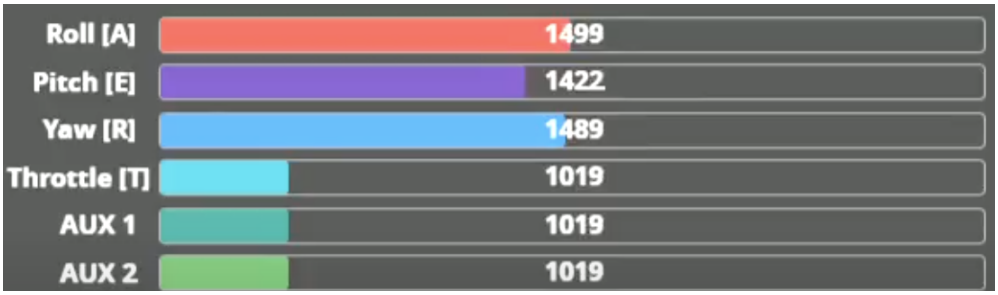

Figure 14: BetaFlight pitch value when the drone is within 4 feet of the AprilTag

We verified that we can establish communication between the Arduino and Raspberry Pi via two GPIO pins. It receives 2-bit digital signals correctly by printing the corresponding mode that the signals represent (00, 01, and 10).

We verify that the RC receiver is receiving correct inputs from the remote control's RC transmitter and throttle control [See Appendix B and Figure 20 for the Remote Control Unit] by verifying it on the BetaFlight GUI]. Figure 15 shows what happens to the throttle value when it is turned up from the remote control.

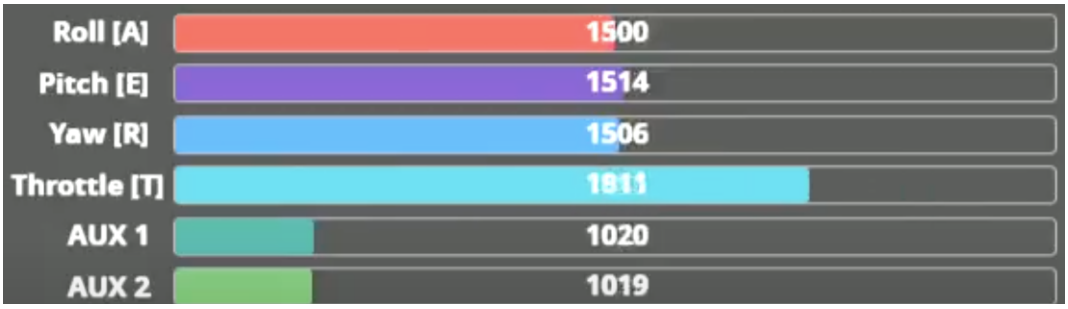

Figure 15: Throttle value when turning the throttle knob

Lastly, we verified that the integrated system of Arduino, Raspberry Pi and BetaFlight worked as intended. As shown in Figure 16, BetaFlight is able to receive the preset signals from the Arduino via PPM. When the command, sent from Arduino, is supposed to move the drone forward, stay, and move the drone backward in that order, as we move the AprilTag from more than 5 feet away from the camera to within 4 feet of the camera. There is an offset error around a value of 20, which can be compensated by deducting this offset in the Arduino code to decrease the error to below 1%.

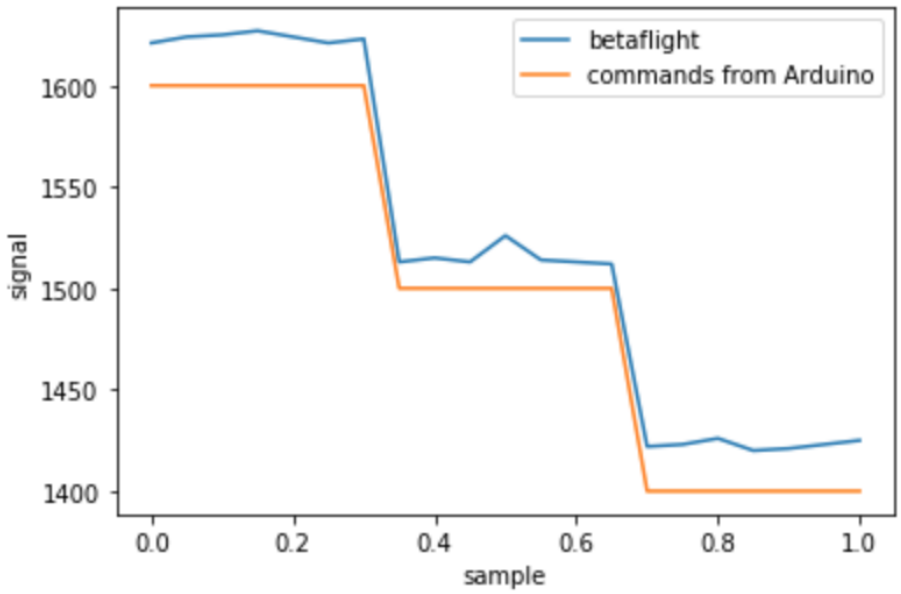

Figure 16: Test result on Integrated system

4. Costs

4.1 Labor

Development of this project took place over a course of 12 weeks, with an average time of 8 hours/week/person. The estimated salary for a graduate student from ECE is $44/hour, and a factor of 2.5 to account for equipment cost and overtime. The overall labor cost is $31,680 which is shown in Equation #.

3 students * $44/hour * 8 hours/week * 12 weeks * 2.5 (overhead included) = $31,680 (4.1)

4.2 Parts

The list of parts used in our project are shown in Table 1.

Table 1: List of parts

Item	Description	Vendor	Quantity	Total Cost
STM32F405RGT6W	STM32 microcontroller	DigiKey	2	$29.88
BMP280 breakout board	Pressure sensor	Amazon	1	$5
ICM20948	IMU	Mouser	2	$11.82
Sparkfun ICM20948 breakout board	IMU	Amazon	1	$16
Miscellaneous capacitors and resistors	SMD 0805 capacitors and resistors	DigiKey	-	$15.45
SSB43L-E3/52T	Schottky diode	DigiKey	3	$1.77
TLV76733DGNR	3.3 V linear voltage regulator	Mouser	3	$2.58
ECS-CTE-8.00-10-TR	8 mhz oscillator	DigiKey	3	$0.81
B82464G4472M000	4.7 uH inductor	Mouser	2	$3.72
LSM0805463V	Blue LED	DigiKey	3	$1.41
MIC5225-1.8YM5-TR	1.8V linear voltage regulator	DigiKey	3	$1.23
ADP2303ARDZ-5.0-R7	5V buck convertor chip	Mouser	3	$9.84

PRPC040SACN-RC	Pin headers	DigiKey	2	$1.80
Jumper cables	-	Amazon	1	$7.98
SEN0304	Ultrasonic Sensor	Mouser	1	$12.90
SD1614T5-B5ME	Buzzer	Mouser	2	$2.50
0565790519	USB connector	DigiKey	3	$2.28
BSS138	MOSFET logic level shifter	Mouser	3	$1.41
JST-XH 3S 4 wire connector	Battery cell voltage monitoring connector	DigiKey	3	$0.72
PCB	-	JLCPCB	5	$16
DJI F450 clone	Drone Frame	Amazon	1	$20
Propellers	-	Amazon	8	$13
Motors	-	Amazon	4	$37
ESCs	-	Amazon	4	$44
Battery	-	Amazon	1	$34
NRF24 Transceivers	-	Amazon	2	$9.49
Raspberry Pi	-	Amazon	1	$70
Camera	-	Amazon	1	$13
PDB-XT60	5 V step-down converter/power distribution board	Amazon	1	$11.59
Total Cost of Parts				**$397.18**

4.3 Total Cost

The total design and labor cost is show in Equation #.

$$\text{Total cost} = \$31{,}680 + \$397.18 = \$32{,}077 \quad (4.2)$$

5. Conclusion

5.1 Accomplishments

We successfully tested the AprilTag detection and distance calculations while signaling the arduino the distances measured to then convert those numbers into values for the microcontroller to adjust the drone when needed. We successfully simulated the flight of the drone with all the subsystems connected without the use of the propellers to see if everything was interfaced correctly. The pitch values were adjusted appropriately when the ArilTag was varying distances from the camera, using the BetaFlight GUI, and the orientation of the drone was successfully changed.

5.2 Uncertainties

Due to BetaFlight's IMU driver not supporting our IMU, we had to update the driver code to match our IMU. When testing the IMU's gyroscope and accelerometer, there were spikes in the data that should not be there when the drone is not moving and oriented upright. Therefore, the flight of the drone was not perfectly stable and we inevitably crashed the drone while only breaking the leg stands of the drone frame. Due to this, further testing of the flight of the drone was halted because we didn't want any of our parts to break. Therefore, we were unable to test the flight of the drone with the use of the AprilTag detection and distance algorithm to move the drone forward or backward in the air.

5.3 Ethical Considerations

Since the goal of our project is to implement a new approach for the user to control the drone, we assume that the action of the drone under ethical evaluation is based on the action of the person controlling the drone. The most relevant codes of ethics for this project are IEEE Code of Ethics #1 and #9 [7], which stresses the importance of ensuring the safety of the public and their properties. Both the drone and its operator must follow the IEEE Code of Ethics and develop safety protocols in case of failures such as communication failure, drone is out of battery or controller malfunction. To mitigate such situations, our arduino code has a signal that can detect if the RC receiver is disconnected from the protoboard. If this happens, it changes the throttle value to 1400 which will lower the drone to the ground at a constant speed.

For the purpose of this project, we assume that the usage of the drone is recreational. Thus, under Federal Aviation Administration's (FAA) policies on Recreational Flyers [8], our project can only be flown at or below 400 feet above the ground when in uncontrolled airspace. In addition, under university policy Fo-05, in order to test our project on campus, we gained approval from the Division of Public Safety by submitting a request to dpscomments@illinois.edu [9].

Usage of Li-Ion batteries also raises several safety concerns [10]. First, they need to be stored at room temperature and not under direct sunlight or other heat sources. Second, Li-Ion battery cells should never be stored fully charged.

Due to the high current applied to the motors, we also need to follow the safety rules regarding lab fire safety and electrical safety. To mitigate potential hazards while testing, we tested without the propellers on the drone until we were able to test it outside once it was ready to fly.

5.4 Future Work

The software issue with the IMU driver must be debugged and resolved. The gyroscope sometimes did not calibrate on initial bootup, and this happened more times than not. After doing research on what IMU to use, this was not the right one to use as it produced too much noise. Figure # shows the gyroscope's measurements when the drone is not moving and positioned upright.

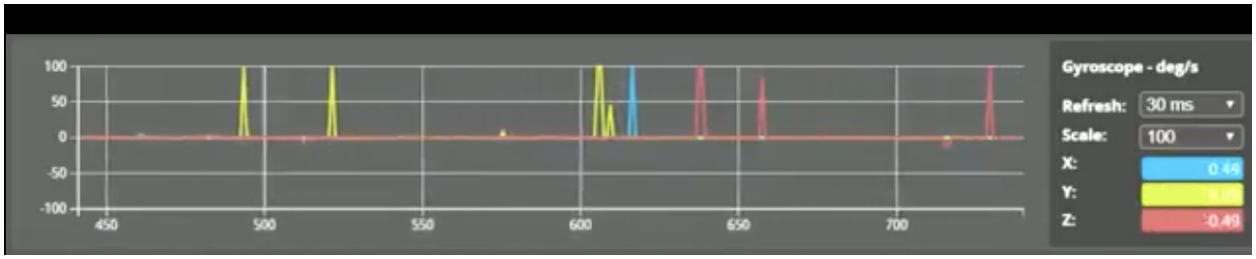

Figure 17: Gyroscope reading when drone is upright on the ground

Additional PID tuning needs to be applied in BetaFlight to adjust each axis of the drone.

An ultrasonic sensor subsystem would also be necessary for obstacle avoidance to mitigate the potential of crashing into an object during flight.

To add to the AprilTag detection, angle detection can be implemented. This can then signal the drone to rotate based on the angle the AprilTag is rotated using the yaw axis in BetaFlight. We could also add additional AprilTags which will signal the drone different commands. For example, make the drone spin in a circle or fly the drone up a certain distance and have it come down the same distance.

References

[1] E. Magid, 2018. *ARTag, AprilTag and CALTag Fiducial Marker Systems: Comparison in a Presence of Partial Marker Occlusion and Rotation*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/318871910_ARTag_AprilTag_and_CALTag_Fiducial_Marker_Systems_Comparison_in_a_Presence_of_Partial_Marker_Occlusion_and_Rotation [Accessed 4 March 2021].

[2] J. McCorvey, "DJI's Spark Is The First Drone You Can Control With Hand Gestures", *Fast Company*, 2021. [Online]. Available: https://www.fastcompany.com/40525592/djis-spark-is-the-first-drone-you-can-control-with-hand-gestures [Accessed: 04- Mar- 2021].

[3] K. Natarajan, T. D. Nguyen and M. Mete, "Hand Gesture Controlled Drones: An Open Source Library," *2018 1st International Conference on Data Intelligence and Security (ICDIS)*, South Padre Island, TX, USA, 2018, pp. 168-175, doi: 10.1109/ICDIS.2018.00035.

[4] D. Etherington, "TechCrunch is now a part of Verizon Media", *Techcrunch.com*, 2021. [Online]. Available: https://techcrunch.com/2020/04/27/mit-muscle-control-system-for-drones-lets-a-pilot-use-gestures-for-accurate-and-specific-navigation/ [Accessed: 04- Mar- 2021].

[5] betaflight/betaflight: Open Source Flight Controller Firmware https://github.com/betaflight/betaflight

[6] A. Rosebrock, "AprilTag with Python - PyImageSearch", *PyImageSearch*, 2021. [Online]. Available: https://www.pyimagesearch.com/2020/11/02/apriltag-with-python/ [Accessed: 05- May- 2021].

[7] "IEEE Code of Ethics", *Ieee.org*, 2021. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html [Accessed: 04- Mar- 2021].

[8] "Recreational Flyers & Modeler Community-Based Organizations", *faa.gov*, 2021. [Online]. Available: https://www.faa.gov/uas/recreational_fliers/ [Accessed: 04- Mar- 2021].

[9] "Aerial Activities Over, On, or In Campus Property – Campus Administrative Manual", *Cam.illinois.edu*, 2021. [Online]. Available: https://cam.illinois.edu/policies/fo-05/ [Accessed: 04- Mar- 2021].

[10] "Lithium-Ion & Lithium Polymer Cells and Batteries Safety Precautions", *Ultralifecorporation.com*, 2021. [Online]. Available: https://www.ultralifecorporation.com/PrivateDocuments/BR_LiIon_Safety_Precautions.pdf [Accessed: 04- Mar- 2021].

Appendix A - Requirement and Verification Table

Requirement	Verification	Verified
Measuring output voltage from the 3 V and 5 V step down converters within ± 5%	Use a digital multimeter and connect it to the output wire from each breakout board. Determine if the measured voltage is 3 V and 5 V respectfully for each board.	Yes
Detect the AprilTag in real time	Connect a wire from pin 16, GPIO 23, to a breadboard in series with a blue LED and ground. Connect another wire to pin 18, GPIO 24, to a breadboard in series with a buzzer and ground. Hold the AprilTag in front of the camera and see if the LED lights up.	Yes
Determine the distance the AprilTag is from the camera within 3% error	Position a measuring tape from the camera and lengthen it to 10 feet. Set up the same LED circuit used in the detection for AprilTag verification. Hold the AprilTag at increments of 1 feet away from the camera and compare the distance calculated to the actual distance and calculate the error percentage.	Yes
Must be able to detect rotational movement at max 1000 degrees per second with an accuracy of 15%.	Connected IMU to an Arduino and set gyroscope sensitivity to 1000 dps. Move IMU with the same time as moving a smartphone with an IMU and compare data.	Yes
The STM32 FPU should be able to make fast calculations on the sensor data to go through the PID control loop and send commands to the ESC at 9 kHz	CPU load under the Betaflight GUI is under 15% with the PID loop running.	Yes
The Betaflight should be able to change the pitch value to 1600 when the AprilTag is above 5 ft away from the camera and 1400 when below 4 ft away with an error rate below 3%.	Move the AprilTag to above 5 ft away from the camera and below 4 ft away from the camera and record the values on BetaFlight GUI.	Yes

Appendix B - Testing Procedure Diagrams

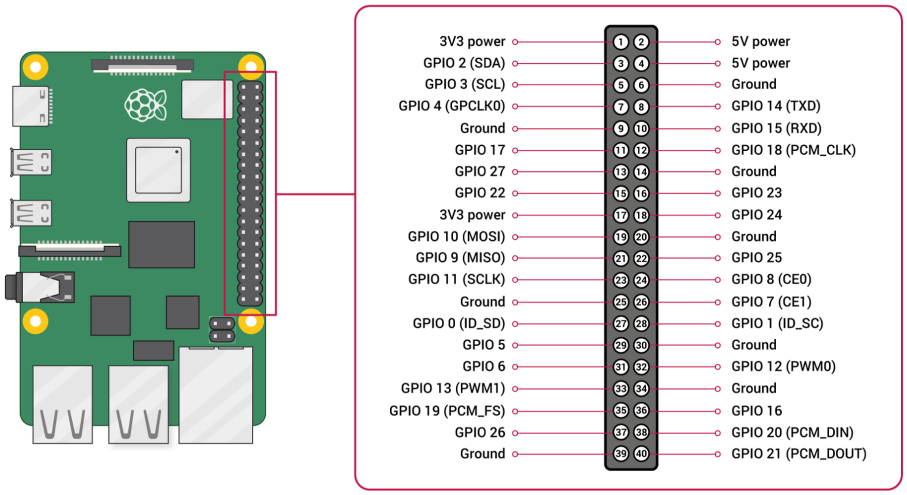

Figure 18: Pinout of the Raspberry Pi

Actual Distance	Measured Distance
1	0.99
2	1.97
3	2.95
4	3.94
5	4.85
6	5.85
7	6.80
8	7.77
9	8.75
10	9.70

Figure 19: Actual distance vs measured distance

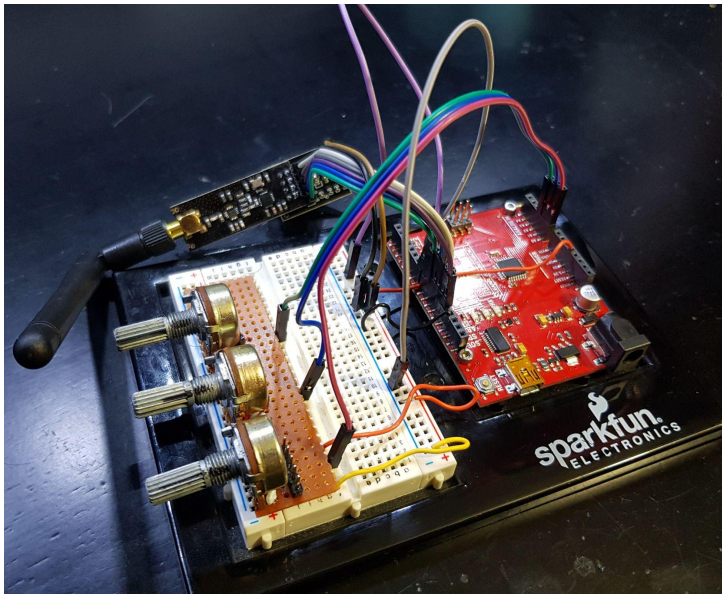

Figure 20: Remote Control Unit

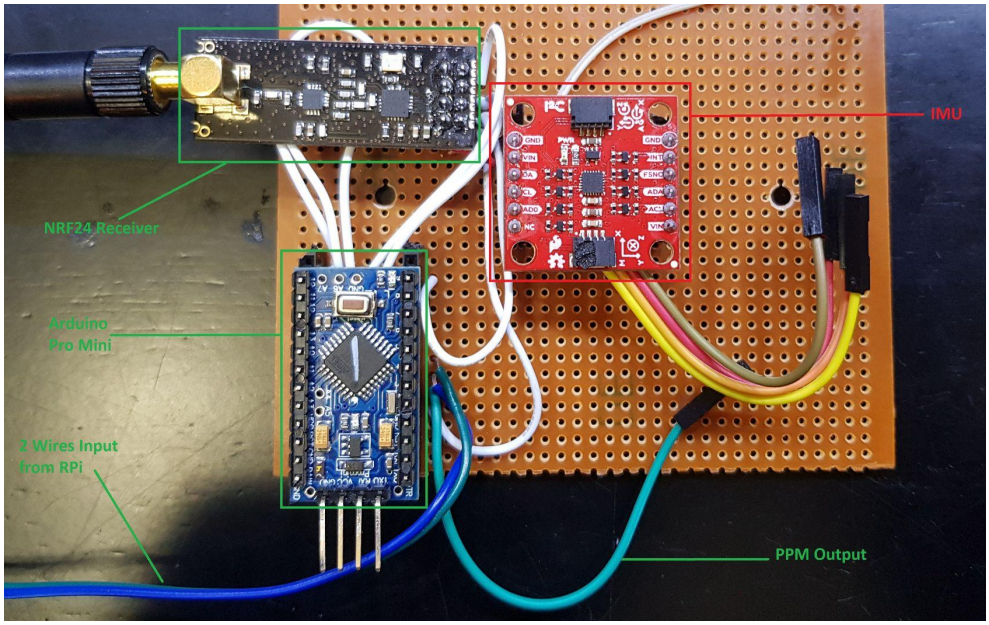

Figure 21: Communication system and external IMU on breakout board

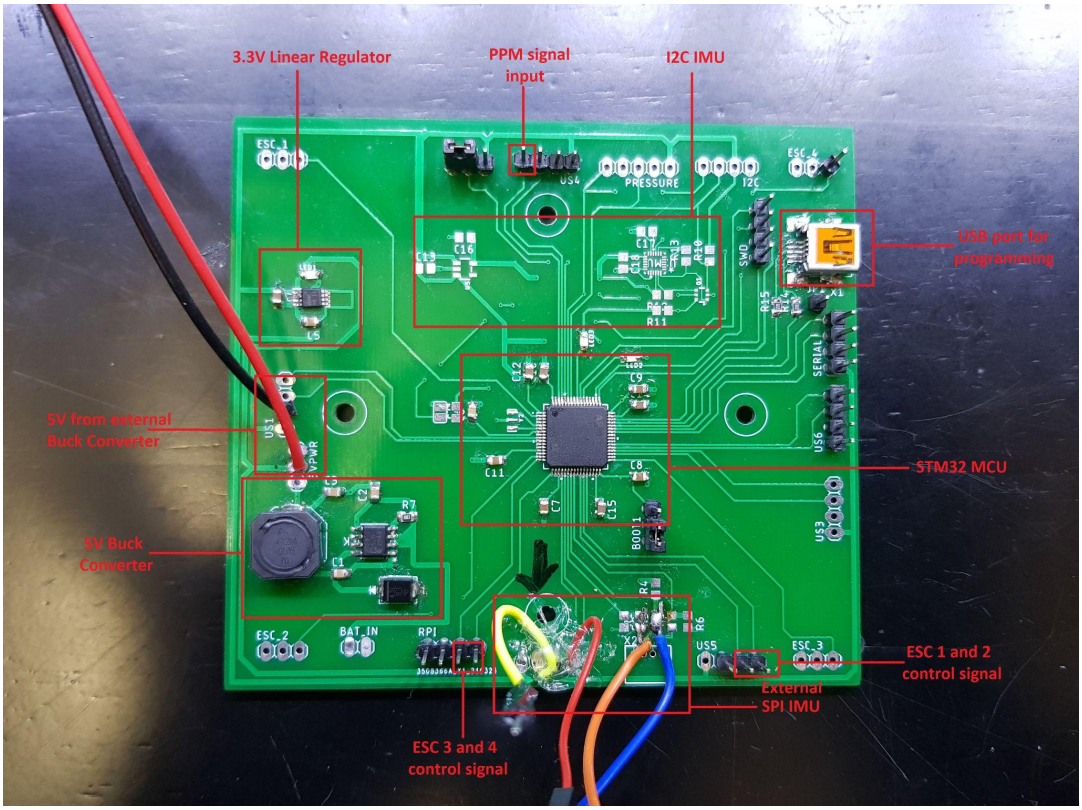

Figure 22: Assembled PCB

Appendix C - Schematics and PCB

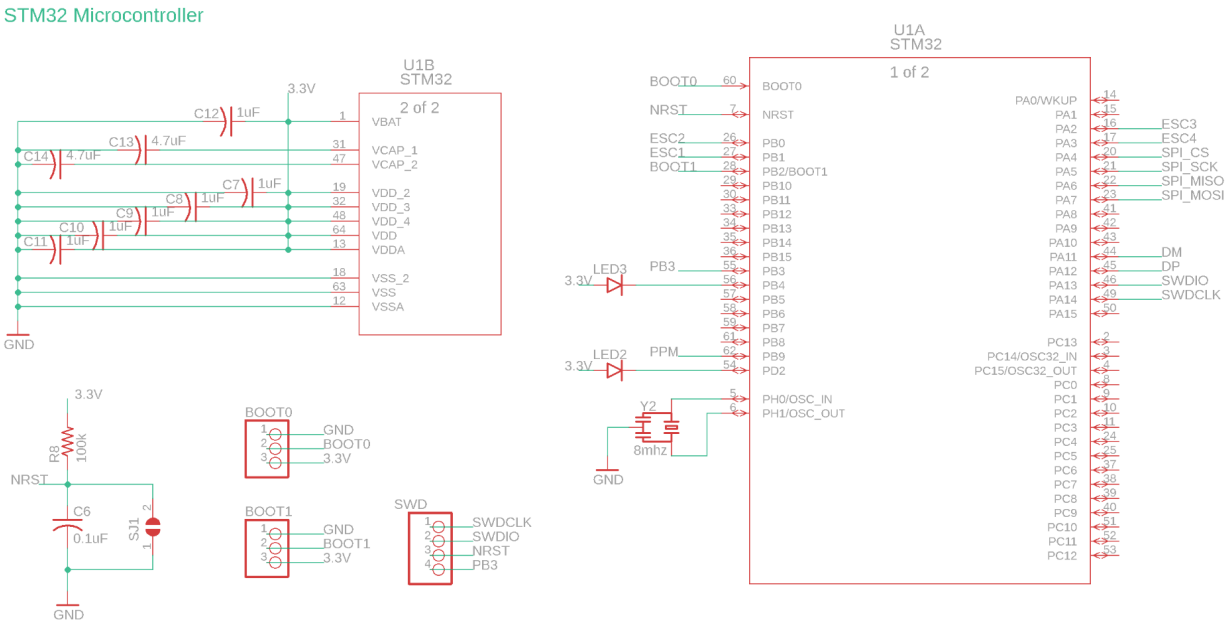

Figure 23: STM32 Microcontroller

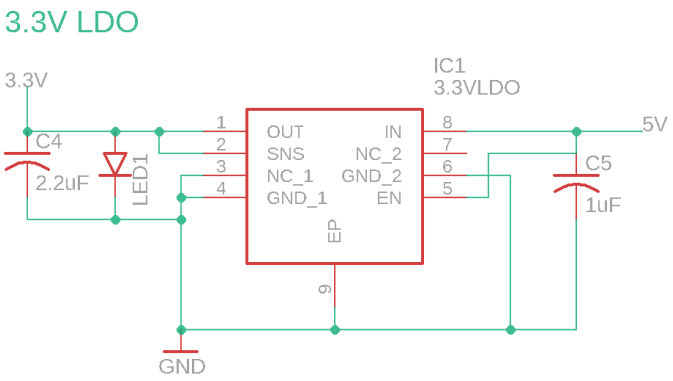

Figure 24: 3.3 V linear regulator

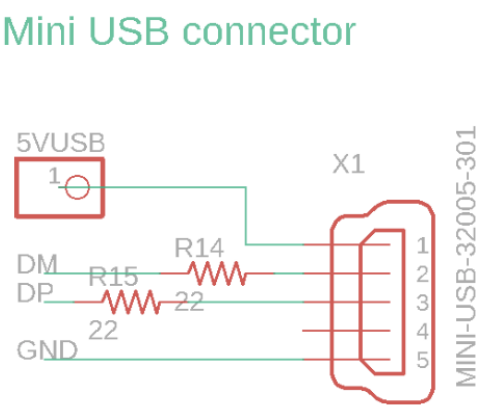

Figure 25: Mini USB Connector

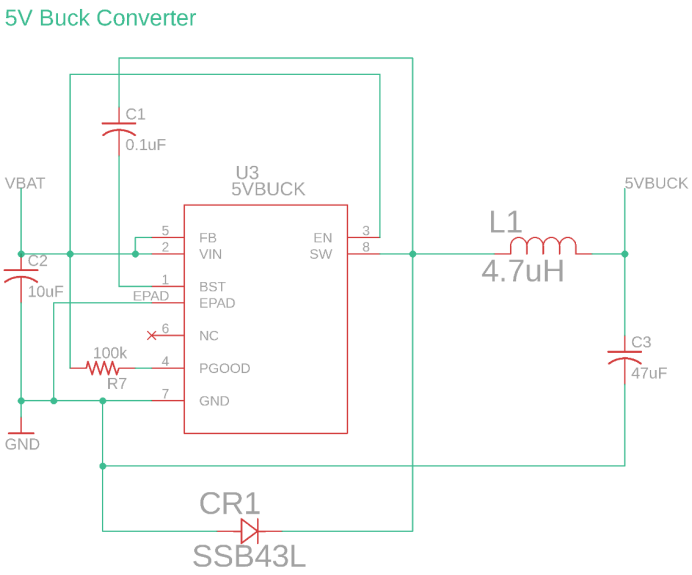

Figure 26: 5 V buck converter

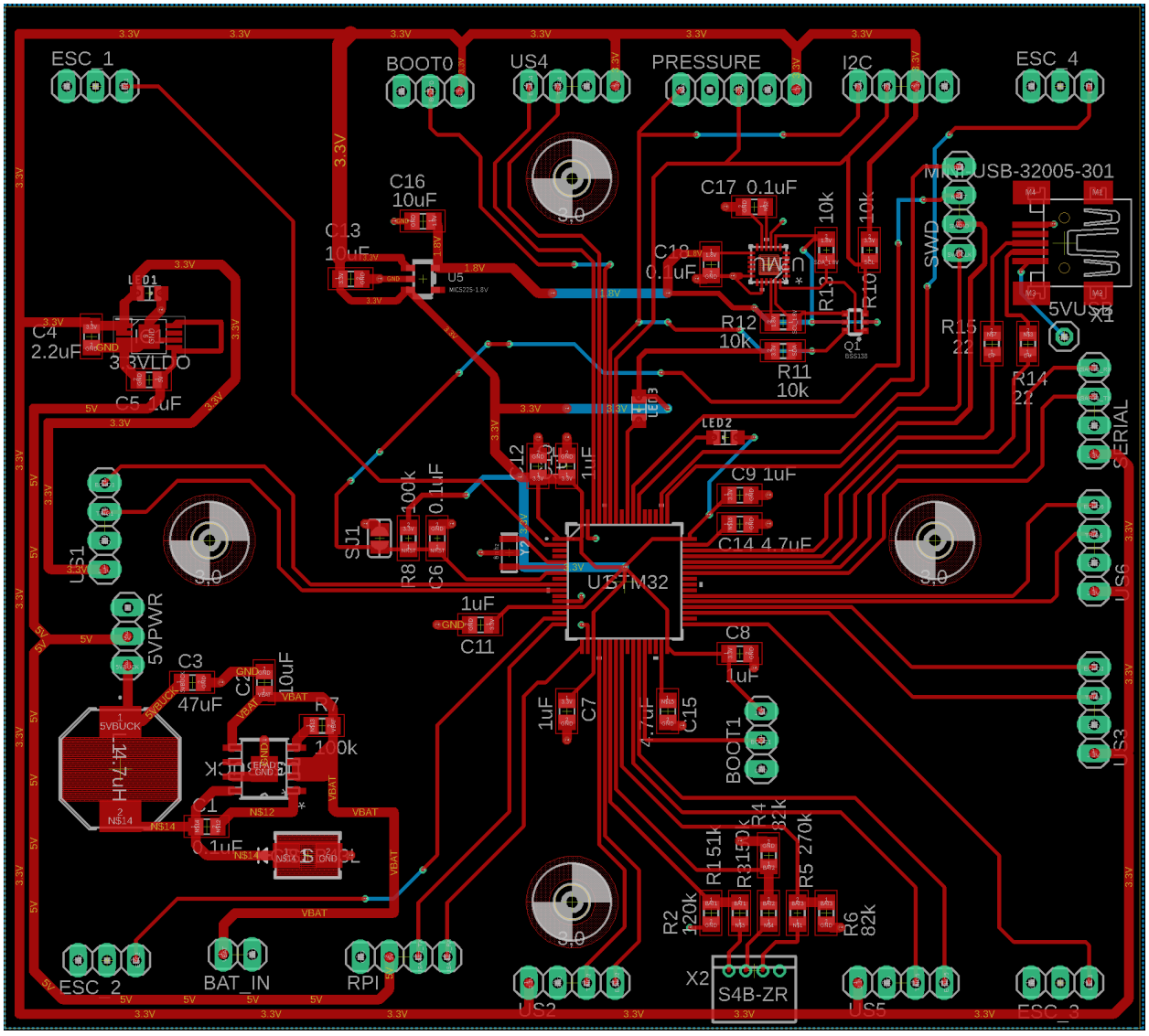

Figure 27: PCB layout top view without ground plane shown