# Project Report for DroneBT

George Uy de Ong II

CS193

University of California, Riverside

November 27, 2017

# Outline

*Disclaimer: I am not responsible of your drone, your safety and others,.This is for educational purposes. If you are planning to replicate this project, I advise you to remove the propellers when testing the drone. Also, test it in a safe environment where no person/animal can get hurt or killed. I am also not responsible, if your drone damages your property or someone else's property. Finally, research the drone policy where you are going to fly the drone. Most places do not allow drones and they will fine you or worse destroy your drone!*

# Introduction

In this project report, I will discuss in detail of my drone project for CS193, controlled by an. It is controlled by an Android app that I made. I will discuss a lot of technical terms and I will do my best to make it simple as possible. I will also put a summary of what I did, so you can see the overall "adventure" of this project. After the summary I will discuss my "adventure" in detail and it will be separated in 3 stages. First stage, I will talk about each of the components and how they work individually. Second stage, is the hardware and programming part. This is where I talk about how I build and coded the drone and how I change some of the components. Third stage, is the testing phase. In this part, I will explain how I calibrate the PID gain values and collect data to optimize my drone as best as I could. Keep in mind, I did not went in order when I was working on this drone. Most of the time, I went back and forth for each stage. For simplicity, I will explain it in order, hopefully not losing the audience attention. So let us begin.

## Summary:

In the first stage, I tested the MPU-6050 (gyro and accelerometer), motors, battery, wifi module, bluetooth module, and receiver. After, I created an Android app. In the next stage, I I

connected all the components together and build the drone. In the last stage, I calibrated the PID gain values and I have done a lot of test. Lastly, I will discuss the major problems I had encounter through my “adventure” and what I could have improve.

# Stage 1: Components

## MPU-6050:

In this stage, I had to understand how the MPU-6050 works. Before I begin talking about the MPU-6050. We have to understand some physics. The drone can fly in three dimensions, just like a plane. I refer to this picture to help me keep in track on how the plane moves. There is a yaw, which the drone spins on the y-axis. The roll rotates the drone in the x-axis and the pitch tilts the drone in the y-axis.

The MPU-6050 is a module that is a gyroscope and accelerometer. We will be using the Wire.h library to use the MPU-6050. I started with the gyroscope. The gyroscope sensor’s job is to know the angular velocity. It detects the change of rotational angle per unit of time. Before we keep moving forward, we have to calibrate the gyroscope because when the module is not moving, the raw data of the gyroscope shows it is moving. So we use this line of code to calibrate the gyro by taking the average of 2000 samples and subtracting the raw data.

```
// Takes a bunch of gyro samples and get the average offset
// Collecting 2000 samples
for (calibration_count = 0; calibration_count < 2000 ; calibration_count ++){
  get_gyro_data();                                    // Get gyro data

  // Adding roll, pitch, yaw values to roll_cal, pitch_cal, yaw_call, respectively.
  gyro_axis_cal[1] += gyro_axis[1];
  gyro_axis_cal[2] += gyro_axis[2];
  gyro_axis_cal[3] += gyro_axis[3];
} // end of calibration loop

//Serial.println();
// Getting gyro average offset by dividing 2000 since we collected 2000 samples.
gyro_axis_cal[1] /= 2000;
gyro_axis_cal[2] /= 2000;
gyro_axis_cal[3] /= 2000;
```

We want the drone to be capable to rotate 360°, so we need to look at the MPU-6050 specification, here[1]. If we look at page 12, we want the full scale range to be 500° per second, since we want the drone to move within a 360° range. Also, we have to activate the MPU-6050 register. The register map pdf file is located in this link[2]. To turn on the register we have to send a hex of 0x1B and send bits of 0x08. This will set the full scale range of 500° per second. Additionally, we will have sensitivity scale factor of 65.5° per second. This means that we can convert the raw data from the gyro to° per second. For example, we rotate the gyro 360° in one second and the MPU-6050 will send a raw data of 23580. To convert this raw data to degree per second, we use the following formula:

$$Axis\_angle\_degrees = raw\_output/65.5$$

[1] "MPU-6000 and MPU-6050 Product Specification ... - InvenSense." https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Accessed 6 Dec. 2017.

[2] "MPU-6000 and MPU-6050 Register Map and ... - InvenSense." 14 Nov. 2011, https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf. Accessed 6 Dec. 2017.

Keep in mind, that 500 is the maximum value, which MPU-6050 will having a reading raw data of 32750. This is almost the maximum value of an int. Also instead of using one second, we can take advantage of the refresh rate of the Arduino Uno, which is 250Hz (4ms). We use the following formula:

$$Axis\_angle \mathrel{+}= raw\_gyro\_x * (1/65.5)*(0.004)$$

This formula means we are integrating every 4ms.

Now there is an issue where we tilt the pitch up at an angle of 45°, the gyro shows that output. But after we rotate the yaw 90° to the left, the pitch output is still 45°. The actual pitch should be 0°, while the actual roll is 45° since the gyro is facing in a different direction. Another word, we have to flip the pitch and the roll when the yaw is rotated. To solve this, we use this formula:

$$Angle\_roll \mathrel{-}= angle\_pitch*sin(gyro\_x* (1/65.5)(0.004)*(3.142/180)$$

$$Angle\_pitch \mathrel{+}= angle\_roll*sin(gyro\_y*(1/65.5)(0.004)*(3.142/180)$$

Keep in note, that the sin function takes in radians.

We are done setting up the gyro, now let us set up the accelerometer. The accelerometer sensor job is to measure the tilting motion of the drone. Keep in mind, that the Earth's gravity plays a factor how the drone moves. So, we know the Earth's gravity is 9.807 m/s^2 (about 1 g)[3]. We want to turn on the full scale range for the accelerometer of +-8g (pg 13 in Product Specification pdf). Additionally, the sensitivity scale factor is 4,096 for one g. So, when the accelerometer is setting at rest, the raw data should be around 4,096. To turn on the register and set the full scale range to 8g. We need to send register hex of 0x1C and send bits of 0x10.

[3] "Gravity of Earth - Wikipedia." https://en.wikipedia.org/wiki/Gravity_of_Earth. Accessed 6 Dec. 2017.

I have finish talking about how to set up the accelerometer, now let me talk about some simple physics to understand how the accelerometer is programmed. Let me begin, by looking at the xy-plane. If we tilt the MPU-6050 at 45°, we would get a reading of 2896 for both vector x and vector y. If we use the pythagorean theorem formula, we will get:

$$Total\_vector = sqrt( (raw\_x^2) + (raw\_y^2) )$$

The total vector about 4,096. This is exactly the same when the MPU-6050 is resting flat on the surface. So, for 3 dimensions the total vector is

$$Total\_vector = sqrt( (raw\_x^2) + (raw\_y^2) + (raw\_z^2) )$$

So, how do we convert the raw data to a useful value that we can use? I am glad that you ask. We use the following formula (asin accepts radians only):

$$Angular\_acc\_pitch\_output = asin( raw\_y/ total\_vector * (1/ 3.142) *(1/180) )$$

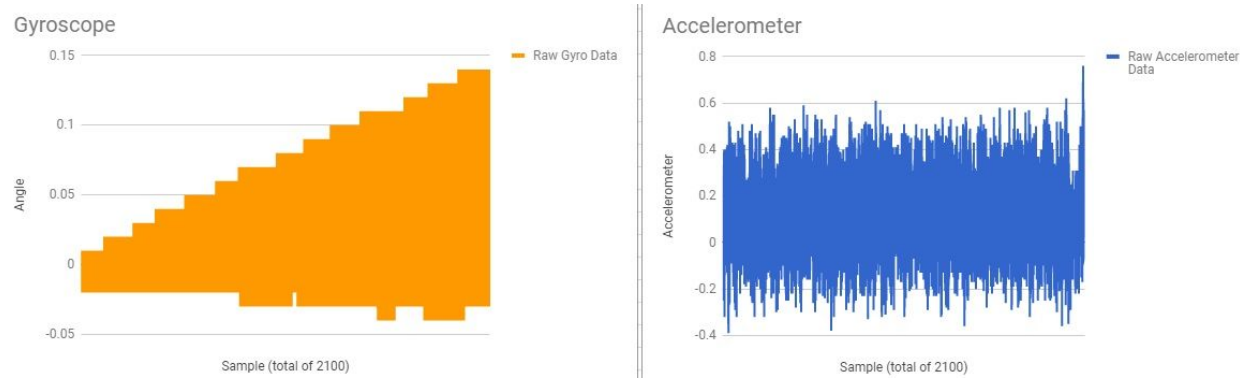
$$Angular\_acc\_roll\_output = asin( raw\_x/total\_vector * (1/ 3.142) *(1/180))$$

I got this formula by collecting the raw x and y data by tilting the MPU-6050 and graphing it. The graph looked like an arcsin or arccos.

We have completed setting up the register for the gyroscope and accelerometer. Now let me discuss a case, where the drone is tilted when it is first initiated. I just have to set the angular accelerometer value to the gyro axis first. Therefore, I use this code:

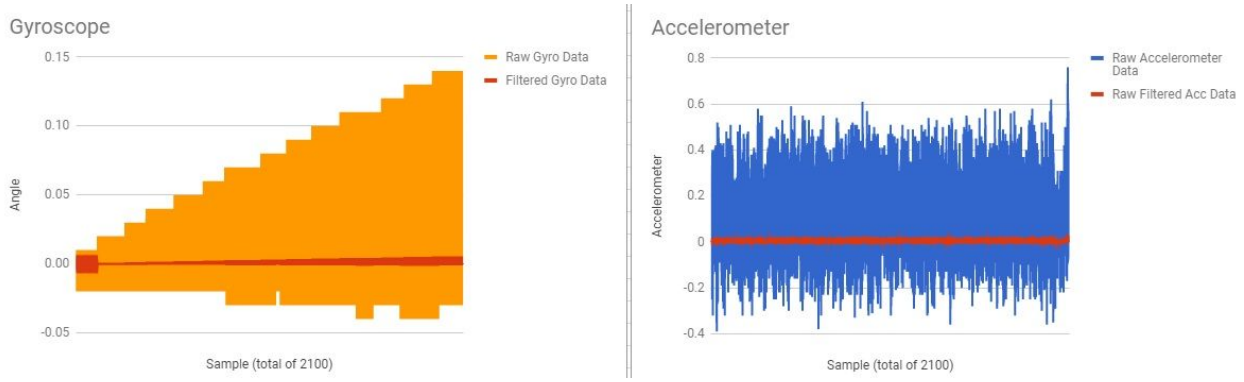
```
// Set gyro angle = accel angle, when starting
// This covers the case when the drone is tilted.
angle_y = accel_angle_y;
angle_x = accel_angle_x;
gyro_angles_set = true;                                    //Set the IMU flag to true.
```

Alright, I hope I did not lose you. We have completed the hardest part of the MPU-6050. Now, I will discuss the vibration noise when the motors spin. To solve this we have to filter it. This is what the data looks like when vibration is present without filtering it.




All I did is get the output and put in google spreadsheet and graph the data.

As you can see there is a lot of noise present. To solve this, we just have to divide to some number that to near zero. Here what the graph looks like when the data is filtered.




For the gyro, that magic number is 0.03819. And for the accelerometer, it is 0.045.

# RC Receiver:

For the receiver, I started with the RC receiver to make sure that the drone flies correctly. I borrowed the code from Joop Brokking[4]. I wanted to make sure the drone moves accordingly.

Here is the code:

```
ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){
    if(last_channel_1 == 0){
      last_channel_1 = 1;
      timer_1 = current_time;
    }
  }
  else if(last_channel_1 == 1){
    last_channel_1 = 0;
    receiver_input[1] = current_time - timer_1;
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){
    if(last_channel_2 == 0){
      last_channel_2 = 1;
      timer_2 = current_time;
    }
  }
  else if(last_channel_2 == 1){
    last_channel_2 = 0;
    receiver_input[2] = current_time - timer_2;
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){
    if(last_channel_3 == 0){
      last_channel_3 = 1;
      timer_3 = current_time;
    }
  }
  else if(last_channel_3 == 1){
    last_channel_3 = 0;
    receiver_input[3] = current_time - timer_3;

  }
  //Channel 4=========================================
  if(PINB & B00001000 ){
    if(last_channel_4 == 0){
      last_channel_4 = 1;
      timer_4 = current_time;
    }
  }
   else if(last_channel_4 == 1){
     last_channel_4 = 0;
     receiver_input[4] = current_time - timer_4;
   }
}
```

[4] "The official YMFC-3D Arduino quadcopter project page ... - Brokking.net." http://www.brokking.net/ymfc-3d_main.html. Accessed 6 Dec. 2017.

The receiver has four channels. Channel 1 is for the row input. Channel 2 is for the pitch input. Channel 3 is for the throttle input, and channel 4 is for the yaw input. The channels are connected to the corresponding pin 8-11. This pins are set to be interrupted. This means that whenever there is a change of value during the middle of the code, hence interrupt, the program stops and does a task that needs its attention, then continues where it was left off.

## Wifi Receiver:

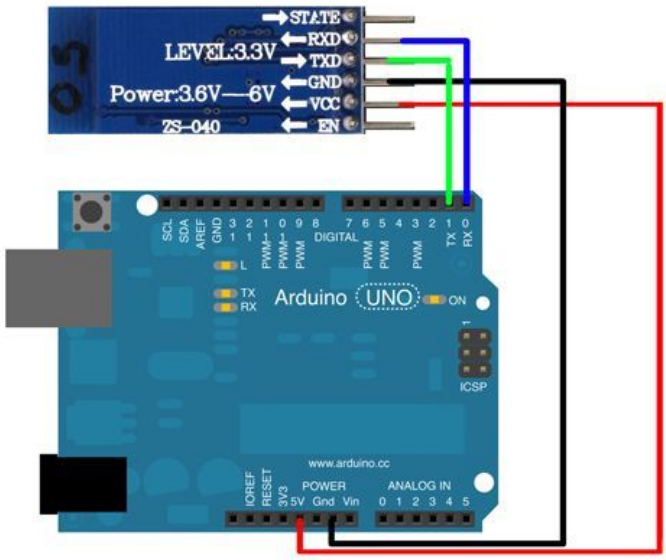
For the wifi receiver, I had trouble receiving data quickly. I needed at least 115200 baudrate. Initially, I designed the code to receive data as fast as the RC controller[5]. I could have change the code to update as slow as the wifi receiver, but this would change my entire code. I also tried to check online for ways how to receive data quickly through wifi, but I could not find a solution.

One approach I did was to update the throttle, pitch, yaw, and roll to a url. Although, the refresh rate was not fast enough. I am guessing it is slow, because the url had to load up the browser which take some time. I also tried to look online on how to send data only, but there was no luck. I spent about two weeks and there were no good progress. So, I decided to switch to bluetooth. I chose bluetooth because I had previous experience with it and I had some old codes that I could reuse.

[5] "Turnigy 9X Transmitter Modifications - Oscar Liang." 4 Dec. 2013, https://oscarliang.com/turnigy-9x-transmitter-modifications/. Accessed 6 Dec. 2017.

## Bluetooth Receiver:

Before we start talking about the bluetooth, we have to set the name, password, and the baud rate. Here are the instructions on how to program the bluetooth module[6], but I will also explain how I programmed my bluetooth. The bluetooth I have is a HC-05 ZS-040. This has a button on the front. Below is a schematic of how to connect the bluetooth to the Arduino Uno:




I am not sure, if it works for the other models.

Before powering the bluetooth, you need to ***press and hold the button*** on the front of the bluetooth module. This will set the bluetooth to programming mode. A flashing led indicates it is in programming mode. Open the Arduino IDE and upload a blank sketch. Open the serial monitor. In the bottom right, change the baudrate to 38400. On the top there is a bar where you

[6] "Modifying the AT Codes on a HC-05 With the Code ZS ... - Instructables." http://www.instructables.com/id/Modifying-the-AT-Codes-on-a-HC-05-With-the-Code-ZS/. Accessed 6 Dec. 2017.

can type “AT,” which stands for Attention Command. You may need to send AT several times to get a reply that says “OK.” Next I set the the name of the bluetooth by using this command:

*AT+NAME=DroneBT*[7]

*(set any name you want)*

I set the password of the bluetooth by sending this command:

*AT+PSWD=0909*

*(set to any password you want)*

Lastly, I set the serial parameter by sending this command:

*AT+UART=115200,0,0*

To use the Bluetooth receiver we had to use the SoftwareSerial library. We need to set up pins in the Arduino Uno by using this line *SoftwareSerial bluetooth_serial(0,1).*
This set pin 0 as the receiver pin and pin 1 as a transmitter pin (we do not need this pin, but we still need to set it). After we initialize it, by using this code *bluetooth_serial.begin(115200).*
Before I begin, I will explain on how the app sends data to the bluetooth module. The bluetooth sends a stream of data, although I kept getting some useless data. To solve this issue, the important data will have markers, ‘<’ and ‘>’. The important data is structured this way:

| 1-bit | 3-bytes | 3-bytes | 2-bytes |
|---|---|---|---|
| 0 | Throttle value | Yaw value | CRC value |

[7] "Serial Port Bluetooth Module (Master/Slave) : HC-05 - ITEAD Wiki." 24 Mar. 2017, https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05. Accessed 6 Dec. 2017.

| 1 | Pitch value | Roll value | CRC value |
|---|---|---|---|

The most significant bit will determine which data is being updated. For example, I receive a data of < 0 125 58 4 >. This means the throttle has a raw value of 125 and yaw has a raw value of 58. The last two-bits is the CRC value. I need to check if the data was not corrupted, during communication. Sometimes during the bluetooth communication, we get interference.

Here is one part of the code for the bluetooth:

```
// Code for the receiver
// This code only gets data that between < and >
// For example we get a stream a data in ASCII like, 0 30 100 23 < 1 23 100 4 > 23 4 90
// The code only gets 1 23 100 4
//Serial.println("convert");
while(BTserial.available() > 0 && new_data == false){              // Waiting to get any data from Bluetooth
  // Read from Bluetooth receiver
  temp_rx_byte = BTserial.read();

  if( rx_in_progress == true){                                     // Goes in here when data has a marker starts with '<'
    if(temp_rx_byte != end_marker){                                // Gets data until end marker (>)
      rx_bytes[index] = temp_rx_byte;                              // Gets each string of data between < >
      index++;                                                     // Increment index so we know how long is the data
      if(index >= number_bytes){                                   // Once it is greater than 32 bytes set to 31
        index = number_bytes - 1;
      }
    }
    else{
      rx_bytes[index] = '\0';                                      // when data sending is finish
      rx_in_progress = false;                                      // set to false
      numbers_rx = index;                                          // Save the number of element that data was sent
      index = 0;                                                   // reset index
      new_data = true;                                             // set new data true, to prepare the next stream of data
    }
  }
  else if ( temp_rx_byte == start_marker){                         // when data starts with '<' (0x3C), begin collecting the stream of data
    rx_in_progress = true;
  }
}
```

This code just receives data and gets the important data between ‘<’ and ‘>’ markers. It also counts the number of of bytes are in between the markers.

```
// |FIRST BIT | SECOND SEGM | THIRD SEGM | FOURTH SEGM|
// |    0     |   THROTTLE  |     YAW    |    CRC     |
// |    1     |     PITCH   |     ROLL   |    CRC     |
//
if(new_data == true){
  // The length of data we want should be 9 bits. If not, the rest of data is garbage. So, we ignore them.
  if(numbers_rx == 9){
    for(byte n= 0; n < numbers_rx; n++){
      // Convert from ASCII char to DEC
      switch(n){
        case 0:
          msb = rx_bytes[n] - 48;
         break;

        case 1:
            second_segment = (rx_bytes[n] - 48)*100;
          break;

        case 2:
            second_segment += (rx_bytes[n] - 48)*10;
          break;

        case 3:
            second_segment += (rx_bytes[n] - 48)*1;
          break;

        case 4:
            third_segment += (rx_bytes[n] - 48)*100;
          break;

        case 5:
            third_segment += (rx_bytes[n] - 48)*10;
          break;

        case 6:
            third_segment += (rx_bytes[n] - 48)*1;
          break;

        case 7:
            forth_segment += (rx_bytes[n] - 48)*10;
          break;

        case 8:
             forth_segment += (rx_bytes[n] - 48)*1;
           break;

         default:
            // Serial.println("ERROR");
           break;
      }
    } // end of for loop

    // Checking if we our data was manipulated during transmission. So we use the CRC method
    // Check CRC is match.
    int tempCRC = (msb + second_segment + third_segment)%divisor_crc;

    if(tempCRC == forth_segment){                                    // If match send the data accordly
      check_crc = true;
      //displayCord(msb,second_segment,third_segment,forth_segment );

      // Mapping the cordinates from 0-254 to 1000-2000
      second_segment = map(second_segment, 235,0,1000,2000);
      third_segment = map(third_segment, 0, 245, 1000, 2000);

      if(msb == 0 && check_crc == true){
        // Set throttle
        rx_input[3] = second_segment;

        // Set yaw
        rx_input[4] = third_segment;
      }
      else if( msb == 1 && check_crc == true){
        // Set pitch
        rx_input[2] = second_segment;

        // Set roll
        rx_input[1] = third_segment;
      }
    } // end of if(tempCRC == forth_segment)
    else{
      check_crc = false;
    }
  } // end of if(numbers_rx == 9)
  new_data = false;
} // end of if(new_data == true)
```

In this part of the code, I convert the ASCII to decimals. After, I check if the CRC value matches. The divisor we use to check the CRC is 16. After checking the CRC, we map the raw values from 0-255 to 1000-2000. This concludes the bluetooth part of the project. Next, I will discuss the motors and battery.

## Motors and Battery:

How do we determine how much voltage is in the battery and how we utilize it with the motors? I am glad that you ask. We know that the Lipo battery is a 11.1V. This is a nominal voltage value. When the battery is in 100%, the maximum value is 12.6V. When it is low, the minimum value is 10V. This means that between the 12.6 to 11.1 the motor thrust is stronger when battery is low. This is obvious, but an important information to keep in mind.

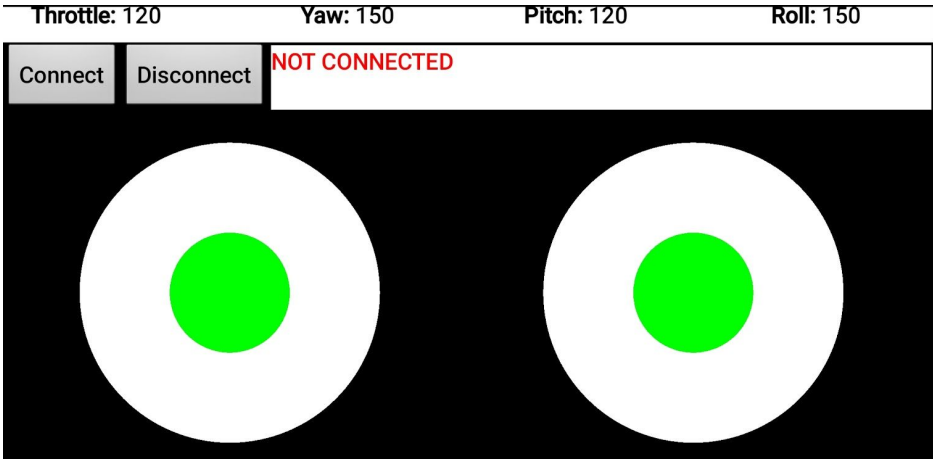
Since the battery has maximum value of 12.6V, I converted it from a range of 0-1023.This is easier range to handle for programming. So, 1260/1023 = 1.2317 at max. Also, there is a diode connected to the battery before it supplies power to the Arduino board. The voltage the diode is about 65nV-70nV. To read the voltage value from the battery, I utilized the *analogRead()* function. So, the code to read the battery voltage is this:

```
// To calculate the battery volatge we also have to consider the diode and the analog read
// Since we are using a Lipo battery that has a maximum voltage of 12.6V and the range of analog we want to read is from 0-1023,
// We divide 1260/1023 = 1.2317
// Also, we have to consider the diode, which is 65nw.
battery_voltage = (analogRead(0) + 65) * 1.2317;
```

# Android App:

I used the MIT App Inventor 2 to create my app[8]. This is the fastest and easier way for me to create an app. My plan was to make the app using App Inventor and if everything works fine, I will convert it to actual Java using Eclipse. The App Inventor is coded in Java too, but it uses blocks to simplify things. After finishing the app, I realize that the app only handles single gestures, instead of multi-gestures. This means that I cannot control the two joysticks at the same time and App Inventor 2 does not handle multi-gestures. Instead of starting over, I kept the app. It still does what I wanted to, but I had difficulties controlling the drone.

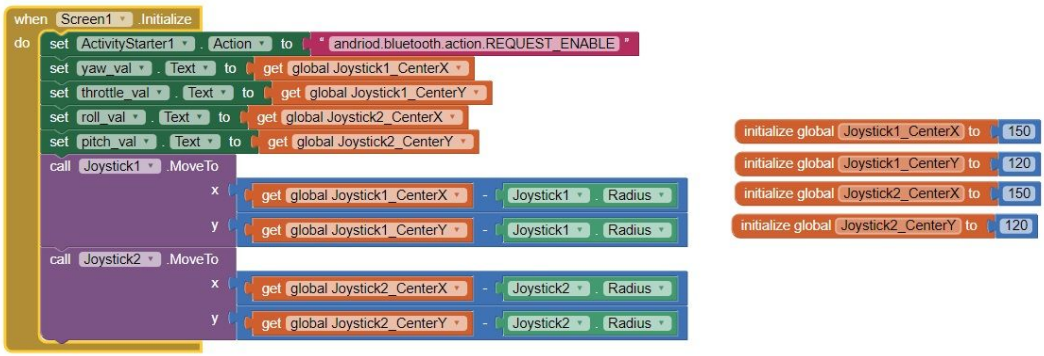
Before I coded the app, I had to create the UI first. This is the design of my app:



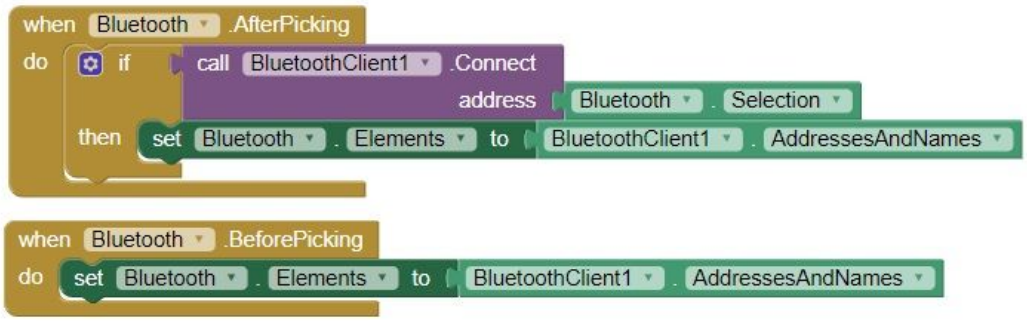

The green circles are the joysticks of the drone controller. The left joystick that controls the throttle (y-axis) and yaw (x-axis). The right joystick controls the pitch (y-axis) and roll (x-axis). The raw values for throttle, yaw, pitch, and roll are show at the top. When pressing the "Connect" button, it will show a list of bluetooth device to connect to. Obviously, when pressing

[8] "MIT App Inventor 2." http://ai2.appinventor.mit.edu/. Accessed 6 Dec. 2017.

the “Disconnect” button, it will disconnect the bluetooth device that is connected to. Now, I will discuss the code. These blocks are the initialize part of the app:

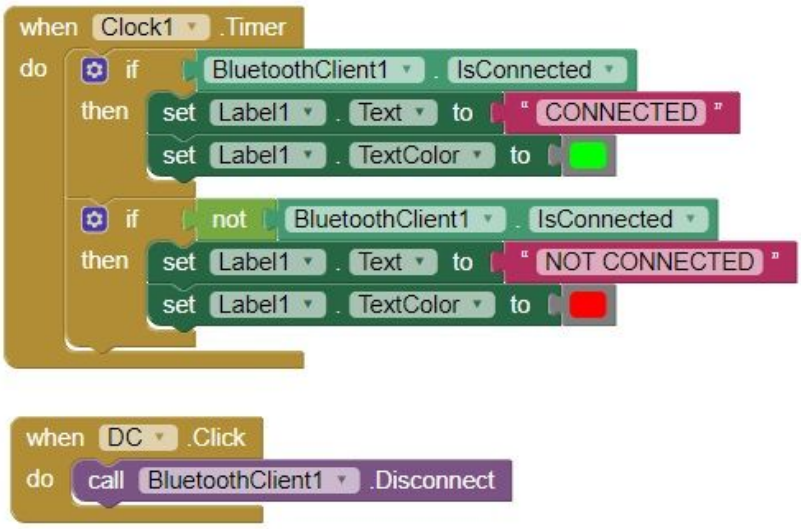



We just set the throttle and pitch value of 120, which is the center at the y-axis. Similarly with roll and yaw, we set the value to 150, which is the center at the x-axis. The purple block is the code to display the position of the green joysticks.

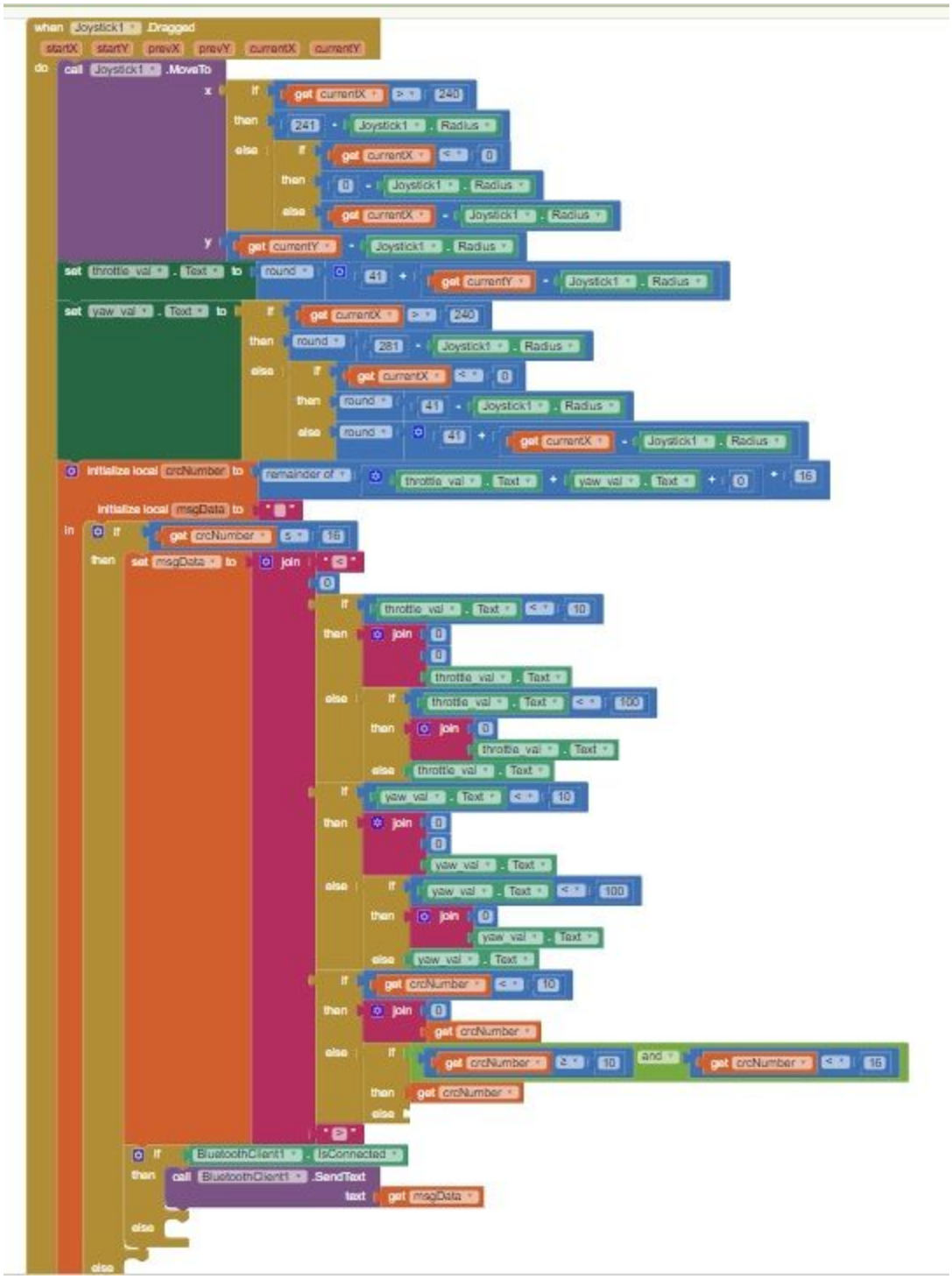



The figure above shows the blocks that can select which device are in the bluetooth list. The block *Bluetooth.BeforePicking*, assign a bluetooth device when the app is started. As you can see, I did not assigned one. So, it will not assigned a bluetooth device.

In these blocks, it just change the display of the app when a bluetooth is connected or not.

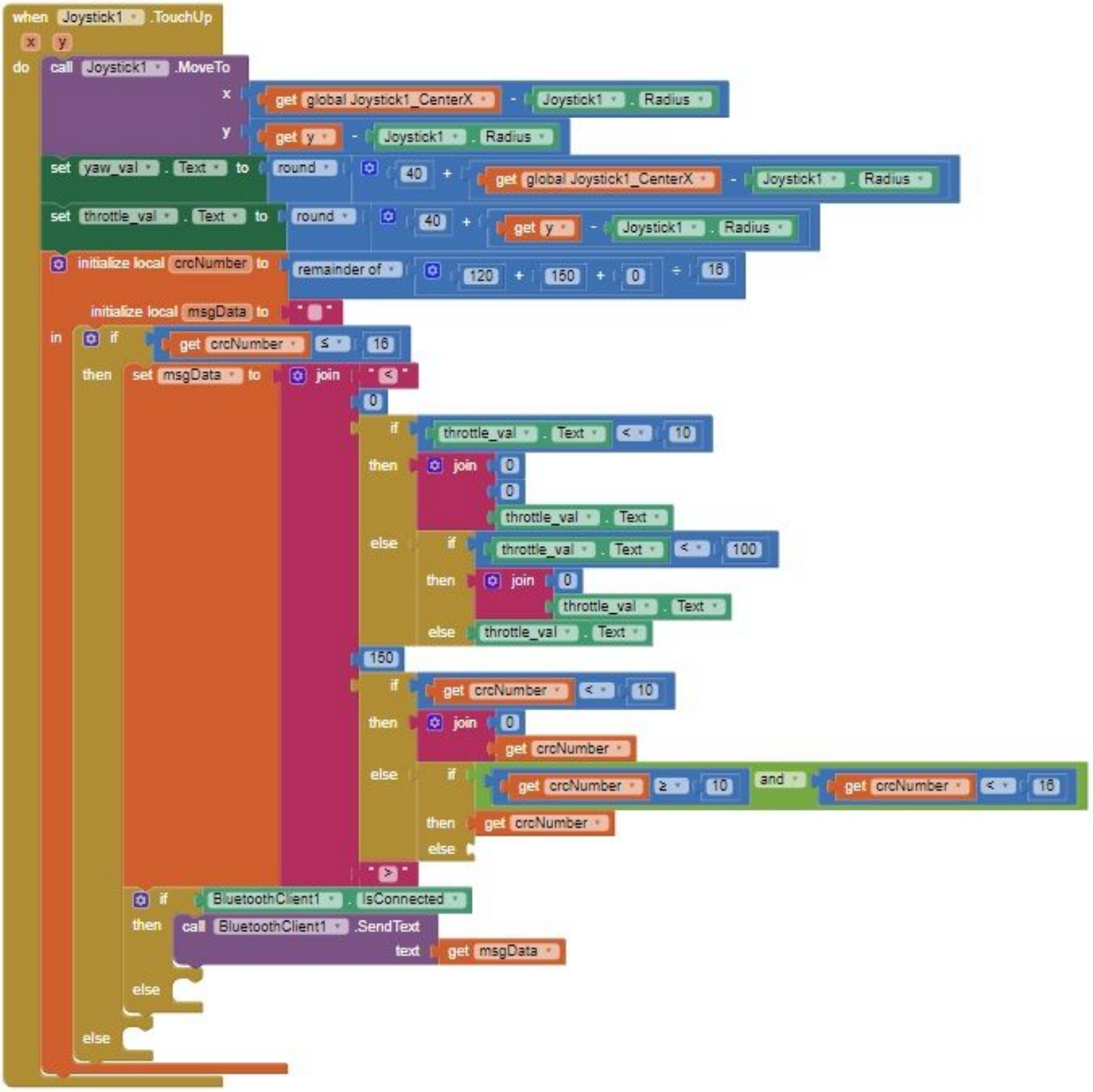



At the bottom figure,the entire block is the code for when the user moves on of the joysticks, it updates the x and y axis value and send it through bluetooth. At the same time, it updates the display. Joystick1 is for the left joystick that handles throttle and yaw. The Joystick2 is for the right joystick that handles pitch and roll.




Also, the figure above sends a structured data, which I explained in the bluetooth receiver.

If you ever held an RC controller, you will notice that the pitch, roll, and yaw returns to the center of its position. This is what the entire block does:




It updates the values of the x and y axis when the user lets go of the joystick. At the same time, it updates the display of the joysticks positions to the center. Similarly to the *Joystick1.Dragged*, the data is structured before sending it through bluetooth.

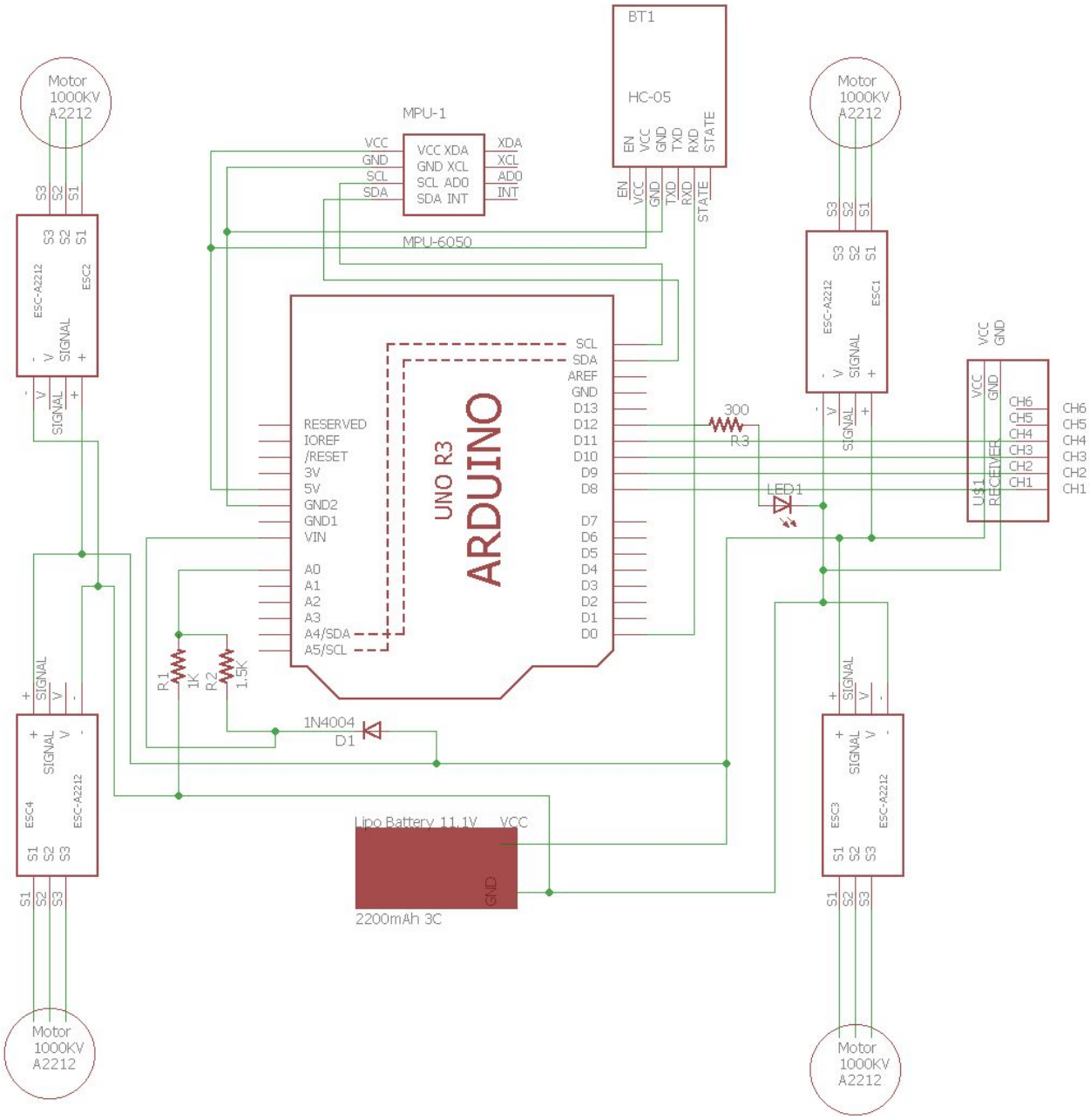
# Stage 2: Hardware and Programming

After discussing how each components work, now I will talk about how I built the drone! Here is the list of parts that I used to build the drone:

| Part | Quantity |
|---|---|
| Drone Frame | 1 |
| Lipo Battery 11.1V 2.2Ah 3C | 1 |
| ESC 30A | 4 |
| Motor A2212 1000KV | 4 |
| Arduino Uno | 1 |
| MPU-6050 | 1 |
| Bluetooth Module HC-05 | 1 |
| RC Receiver | 1 |
| Resistor 1K Ohms | 1 |
| Resistor 1.5K Ohms | 1 |
| Resistor 300 Ohms | 1 |

| | |
|---|---|
| Diode 1N4004 | 1 |
| Red LED | 1 |
| RC controller | 1 |
| Lipo charger w/ power supply and adapters | 1 |
| Propellers 6045 | 8 |
| Android phone (Note 8) | 1 |

Below, is the schematic of how each components is connected:




*Note: This schematic is my second and finalize drone. My first drone crashed during flight testing. For any projects,* ***safety is the number one priority****. If not handled carefully, you may damage your properties or others. I am speaking from experience. Also, you may hurt or kill someone.*

For the code, I combined the sample programs from the Stage 1: Components. In the DroneBT_v1.0, I commented what each part of the code does. Since there is a lot of things to cover, I will just discuss only the important parts of the code.I will talk about three sections of the code, Initialize, Planning, and Control.

## Initialize Section:

The initialize part is the setup(). We first initialize all the variables.

```
// Local variables
// Mapping range high, center, and low for throttle, pitch, roll, and yaw
int data[13] = { 1010,  1999,  1599,          // Roll: low | high | center
                 1010,  1999,  1500,          // Pitch: low | high | center
                 1000,  1999,  1502,          // Throttle: low | high | center
                 1010,  1990,  1612 };        // Yaw: low | high | center
int countPause = 0;                           // A counter to display ". . . ."
// Setting variables
auto_lvl = true;
new_data = false;
check_crc = false;
```

Next, we get some data from the EEPROM that I collected from the RC controller and MPU-6050. The EEPROM has the maximum and minimum values from the RC controller. Also, it has data from the MPU-6050.

The bottom figure, we take 2000 samples and average of the gyroscope and accelerometer and calibrate the values.

```
// Takes a bunch of gyro samples and get the average offset
// Collecting 2000 samples
for (calibration_count = 0; calibration_count < 2000 ; calibration_count ++){
  // Flashing LED that indicates drone is calibrating
  if(calibration_count % 15 == 0){
    digitalWrite(12, !digitalRead(12));
  }
  if(countPause < 200){
    countPause++;
  }
  else {
   // Serial.print(".");
    countPause = 0;
  }
  get_gyro_data();                              // Get gyro data

  // Adding roll, pitch, yaw values to roll_cal, pitch_cal, yaw_call, respectively.
  gyro_axis_cal[1] += gyro_axis[1];
  gyro_axis_cal[2] += gyro_axis[2];
  gyro_axis_cal[3] += gyro_axis[3];

  // Setting motors to high and low to avoid beeping.
  PORTD |= B11110000;
  delay(1);
  PORTD &= B00001111;
  delay(3);
} // end of calibration loop
```

Lastly, we just wait until the user change the position of the left joystick to the bottom-left.

```
// Waiting until user set the throttle to the lowest position
// This is for safety to avoid drone flying up right after calibration.
while(rx_ch3 < 990 || rx_ch3 > 1120 || rx_ch4 < 1400){
  start++;
  rx_ch3 = convert_channel(3);                  // Convert BT throttle signals to 1000 to 2000
  rx_ch4 = convert_channel(4);                  // Convert the actual receiver signals for yaw to the standard 1000 - 2000us
```

## Planning Section:

In the planning section, it is mostly focus on calculating the gyroscope and accelerometer to the corresponding movements from the receiver. In the Stage 1: Component, I explained the codes for the MPU-6050 and bluetooth HC-05. The code for the bluetooth is the same, but I just put it in the function called, convert_channel(). Channel 1 is for roll, channel 2 is for pitch, channel 3 is throttle, and channel 4 is yaw. The function converts the raw values to 1000-2000 us. For example, the bluetooth receiver gets a raw throttle value of 120 and the conversion value for it is 1500.

Here is code for the MPU-6050 code, when put together:

```
// The MPU-6050 sensitivity scale is 65.5 = 1 degree per sec
// This information is located in the MPU-6050 datasheet at page 12 in parameter called gyroscope sensitivity.
input_gyro_yaw = (input_gyro_yaw * 0.7) + ((gyro_yaw / 65.5) * 0.3);
input_gyro_roll = (input_gyro_roll * 0.7) + ((gyro_roll / 65.5) * 0.3);
input_gyro_pitch = (input_gyro_pitch * 0.7) + ((gyro_pitch / 65.5) * 0.3);


// The refresh rate is every 4ms = 250Hz.
// We can get the gyro output for every 4ms. Therefore, we do (raw_gyro_data/250/65.5) = travel angle or
// 0.0000611 = 1 / (250Hz / 65.5)
// Calculate the traveled roll and pitch angle then add them to the corresponding x and y axis (x = roll, pitch = y).
angle_y += gyro_pitch * 0.0000611;
angle_x += gyro_roll * 0.0000611;

// Since the sin functions accept radians only, we have to convert 0.0000611 to radians.
// 0.000001066 = 0.0000611 * (pi/180) radians
angle_y -= angle_x * sin(gyro_yaw * 0.000001066);   // If the drone turns in the yaw direction when the roll becomes the pitch
angle_x += angle_y * sin(gyro_yaw * 0.000001066);   // Similar as for the prev line of code, but pitch becomes the roll

// Calcualte the total accelerometer vector
// v_vector = sqrt(x^2+y^2+z^2)
total_accel = sqrt((accel_x*accel_x)+(accel_y*accel_y)+(accel_z*accel_z));


if(abs(accel_y) < total_accel){                                    // Check if accel is less than total to avoid NaN because of asin
  accel_angle_y = asin((float)accel_y/total_accel)* 57.296;        // Convert 180/pi to radians = 57.296. Calculate the pitch angle.
}
if(abs(accel_x) < total_accel){
  accel_angle_x = asin((float)accel_x/total_accel)* -57.296;       // Calculate the roll angle.
}

// This substart the actual level of the drone when it is placed in a leveled surface.
accel_angle_y -= SPIRIT_Y;                                 // Substart the actual level in y axis
accel_angle_x -= SPIRIT_X;                                 // Substart the actual level in x axis

angle_y = angle_y * 0.03819 + accel_angle_y * 0.045;       // This corrects the drift of the drone in y-axis.
angle_x = angle_x * 0.03819 + accel_angle_x * 0.045;       // This corrects the drift of the drone in x-axis.

pitch_lvl = angle_y * 15;                                  // Calculate the angle correction for the y-axis(pitch).
roll_lvl = angle_x * 15;                                   // Calculate the angle correction for the x-axis(roll).
```

Again, refer to MPU-6050 at Stage 1. I explained it in detail what each line of code does.

## Control Section:

The control section is the part where we control each of the ESC and motors the way we want it to. These four lines of codes will tell how fast the motor should move, depending on the PID output values for the pitch, roll, and yaw:

```
esc1 = throttle - output_pitch + output_roll - output_yaw; // Front right motor: ESC1 pulse calculation
esc2 = throttle + output_pitch + output_roll + output_yaw; // Rear right motor: ESC2 pulse calculation
esc3 = throttle + output_pitch - output_roll - output_yaw; // Rear left motor: ESC3 pulse calculation
esc4 = throttle - output_pitch - output_roll + output_yaw; // Front left motor: ESC4 pulse calculation
```

Keep in mind, that the voltage change as the battery is being used up. These several lines of code handle those changes:

```
if (battery_voltage < 1240 && battery_voltage > 800){      // Battery must be connected
  // Consider the voltage drop for each ESC pulse
  esc1 += esc1 * ((1240 - battery_voltage)/(float)3500);
  esc2 += esc2 * ((1240 - battery_voltage)/(float)3500);
  esc3 += esc3 * ((1240 - battery_voltage)/(float)3500);
  esc4 += esc4 * ((1240 - battery_voltage)/(float)3500);
}
```

Finally, sometimes we get numbers beyond 2000 or below 1000. These lines of codes handle those conditions.

```
  // When the ESC is below 1100, keep the motors running.
  if (esc1 < 1100){
    esc1 = 1100;
  }
  if (esc2 < 1100){
    esc2 = 1100;
  }
  if (esc3 < 1100){
    esc3 = 1100;
  }
  if (esc4 < 1100){
    esc4 = 1100;
  }

  // ESC max value is 2000
  if(esc1 > 2000){
    esc1 = 2000;
  }
  if(esc2 > 2000){
    esc2 = 2000;
  }
  if(esc3 > 2000){
    esc3 = 2000;
  }
  if(esc4 > 2000){
    esc4 = 2000;
  }
}
else{
  // If drone not flying, set ESC to 1000, to avoid annoying beeps
  esc1 = 1000;
  esc2 = 1000;
  esc3 = 1000;
  esc4 = 1000;
} // end of if and else control motors
```

*Note: Before I test the drone with the bluetooth, I tested with an RC controller to make sure all the motors moves accordingly. I borrowed the receiver code from Joop Brokking. In the drone schematic, you saw the receiver connected to pin 8-11. Refer to Stage 1: Component, RC Receiver.*

# Stage 3: Testing

In this stage, we will be testing the drone a lot and this is the most tedious process. But, in the end it was worth it!

Before I flew the drone, I used the RC controller to test if the motors move the way I wanted it to. At first, I did not attach the propellers to the motors.I carefully listened to each of the motors as I move each joysticks. Next, I added the propellers to the drone. I mounted my drone to a flight test jig that I made. It is simple two chairs that are tied up to the drone with some ropes. After confirming the motors move correctly, I am ready to use the bluetooth receiver and app. I repeated the test as before, but this time using the bluetooth receiver and app. After confirming the drone's move correctly with the bluetooth, I was ready to calibrate the PID gain values.

PID stands for Proportional Integral Differential. In simple terms, we want the drone to fly smoothly and steadily. There are five steps on how I adjust the PID drone.

Step 1: I set the yaw P-gain to 4.0, I-gain to 0.02, and D-gain to 0.0. This is just for the bases.
Step 2: I incremented the roll D-gainl by 1.0. The drone can fly, but it was not steady. I keep incrementing the D-gain value, until the drone wobbles when flying. I decrease the value by 10%.
Step 3: I repeat the similar instruction as step 2, but this time I increment the I-gain by 0.01, until the drone wobbles.

Step 4: I repeat the similar instruction as step 2, but this time I increment the P-gain by 0.1, until the drone wobbles. From this point, the drone should be flying more steadily and smoothly than before.

Step 5: Lastly, I repeat step 2-4 to fine tune the drone.

After a hundred of flight test, I settled with these values:

```
////////////////////////////////////////////////////////////////////////////////////
//  PID VARIABLES
////////////////////////////////////////////////////////////////////////////////////
float p_roll = 2.6;                       // P gain roll
float i_roll = 0.07;                      // I gain roll
float d_roll = 20.0;                      // D gain roll
int MAX_ROLL = 100;                       // Absolute max value for PID for roll

float p_pitch = p_roll;                   // P gain pitch (Same values as roll)
float i_pitch = i_roll;                   // I gain pitch (Same values as roll)
float d_pitch = d_roll;                   // D gain pitch (Same values as roll)
int MAX_PITCH = MAX_ROLL;                 // Absolute max value for PID for pitch

float p_yaw = 4.0;                        // P gain yaw
float i_yaw = 0.02;                       // P gain yaw
float d_yaw = 0.0;                        // P gain yaw
int MAX_YAW = 100;                        // Absolute max value for PID for yaw
```

These concludes all the three stages.

# Problems:

I encountered a lot of problems through my adventure with this project. I will discuss the major problems. Initially, I started this project back in Summer 2016. As you know, my first prototype had an accident. But before the accident, I had problems with the components. First, I was using a different gyroscope, L3GD20H. This module did not have an accelerometer. Second

issue, I was reusing motors I had which was 2100KV[9]. These motors were meant for racing drones and it was an overkill for this project. Similarly with the battery, I was using a 14.8V 2.2Ah Lipo battery and it was too heavy. After the first drone crashed, I went back to the drawing board and I decided to purchase a 1100KV motors and 11.1V 2.2Ah Lipo battery.

Another major problem that I encountered is that the components fail, after the drone crashes during test flight. I did not expect to spend a lot for spare parts, but a lesson to keep in mind for future projects. I had to change the MPU-6050 component three times. I had to change one set of ESC and motor. Lastly, I had to build a second drone.

I mentioned before safety is a number one priority. I cannot emphasize enough! When I was calibrating the PID, I did not had a test flight jig for my drone yet. I had to hold the drone wiht one hand and I had to feel its movement, while controller the RC controller with my other hand. I was checking, if it was moving correctly. Unfortunately, I cut my fingers twice.

Lastly, where I live have strict drone policy. I could not fly the drone in the parks near me. I had test the drone in my house. My backyard was too small for my drone to fly around. This is a problem because the drone requires a large spacious area to reduce the number of drone crashes.

[9] "Cobra CM-2206/20 Multirotor Motor, Racing Edition, Kv=2100 ...." http://innov8tivedesigns.com/cobra-cm-2206-20-multirotor-motor-kv-2100. Accessed 6 Dec. 2017.

# Conclusion:

This project took me about three to four months. I have learned how to use the gyroscope and accelerometer. I also understand how a plane and a drone fly in a three dimension space. Also, I figured out how to send and receive data using the CRC method. Most importantly, I learned how to collect large data and graph them to see patterns and formulate equations. If I were to improve the project, I would focus on optimizing my CRC code. Additionally, I would redesign my app to support multi-gestures. Lastly, I would want to add more sensors to the drone, so it can fly autonomously. So in the future, I can have several drones that can communicate with each other and map out the environment.

# Reference


1. "MPU-6000 and MPU-6050 Product Specification ... - InvenSense." https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Accessed 6 Dec. 2017.
2. "MPU-6000 and MPU-6050 Register Map and ... - InvenSense." 14 Nov. 2011, https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf . Accessed 6 Dec. 2017.
3. "Gravity of Earth - Wikipedia." https://en.wikipedia.org/wiki/Gravity_of_Earth. Accessed 6 Dec. 2017.
4. "The official YMFC-3D Arduino quadcopter project page ... - Brokking.net." http://www.brokking.net/ymfc-3d_main.html. Accessed 6 Dec. 2017.
5. "Modifying the AT Codes on a HC-05 With the Code ZS ... - Instructables." http://www.instructables.com/id/Modifying-the-AT-Codes-on-a-HC-05-With-the-Code-ZS/. Accessed 6 Dec. 2017.
6. "Serial Port Bluetooth Module (Master/Slave) : HC-05 - ITEAD Wiki." 24 Mar. 2017, https://www.itead.cc/wiki/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05. Accessed 6 Dec. 2017.
7. "MIT App Inventor 2." http://ai2.appinventor.mit.edu/. Accessed 6 Dec. 2017.
8. O., Carter. "DIY Drones - What You Need To Know To Build One." *Lemon Drone*. N.p., 03 Apr. 2017. http://www.lemondrone.com/blog/diy-drones-how-to-build-quadcopter/ Accessed. 06 Dec. 2017.

9. Benson, Coleman. "How to Make a Drone/UAV." *Robot Shop*. N.p., 29 Oct. 2014. https://www.robotshop.com/blog/en/make-uav-lesson-1-platform-rtf-arf-kit-custom-13989 Accessed. 6 Dec. 2017.

10. "Cobra CM-2206/20 Multirotor Motor, Racing Edition, Kv=2100 ...." http://innov8tivedesigns.com/cobra-cm-2206-20-multirotor-motor-kv-2100. Accessed 6 Dec. 2017.