Quadcopter Drone Project

Jonathan Ibarra
California State University East Bay
Computer Engineering Department of CSUEB
Newark, USA
jibarra27@horizon.csueastbay.edu

Paul Nguyen
California State University East Bay
Computer Engineering Department of CSUEB
Santa Clara, USA
pnguyen108@horizon.eastbay.edu

Luis Rosales
California State University East Bay
Computer Engineering Department of CSUEB
Fremont, USA
Lrosales6@horizon.csueastbay.edu

Rishita Kar
California State University
Compture Engineering Department of CSUEB
rkar@horizon.csueastbay.edu

***Abstract*— The Quadcopter Drone Project was researched for a computer engineering (CMPE) capstone requirement. This document covers the structure of the project by reviewing areas such as schematic, layout, and code for a customized printable circuit board (PCB). Communication between hardware and software for the flight controller was designed by students of the CMPE 492/493 courses to enable an automated flight build. This study further focuses on challenges faced throughout the development process, including modifications, redesign, errors, and milestones achieved.**

I. Introduction

Robotics is popular in media and science, technology, engineering, and math (STEM) research and enables students to understand how robots can interact with the physical world through accessible hands-on experience. The Quadcopter Drone Project was inspired by Intel's drone light show during the Winter Olympics 2018 where more than a thousand drones were synced together to display movable images in the sky. The goal of the Quadcopter Drone Project was to build an autonomous quadcopter drone that hovers at a given height via Bluetooth signal. The project was studied by four computer engineering students by separating duties for hardware and software into teams of two. Hardware tasks included creating the top-level block diagram, schematic, and layout for the printable circuit board (PCB) and then soldering electronic components onto the PCB to test. Software converted mathematics and physics for drone movement into code, and tested sensors and microcontroller for the customized PCB. There were challenges that led to design changes that has improved the understanding and overall clarity of the assignment.

II. Top Level Block Diagram

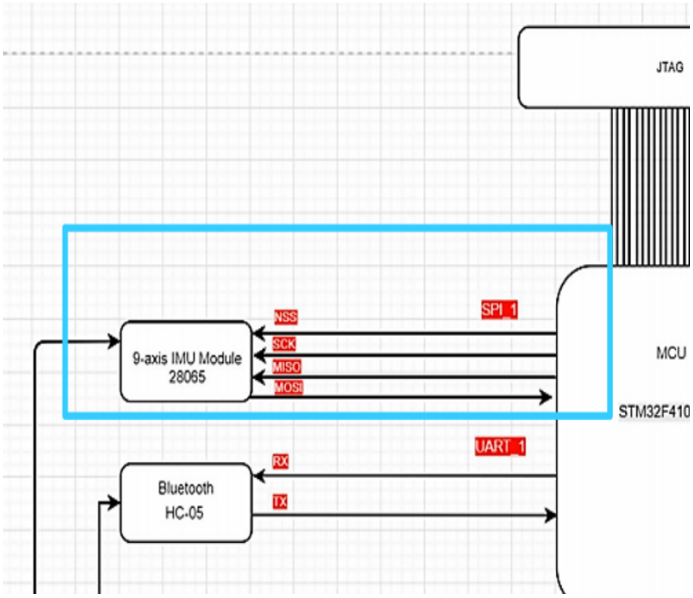
Challenges in the top-level block diagram were evident in the initial draft. There were issues with communication between peripherals and the microcontroller. The goal of the project was to have a quadcopter drone hover and researching for the most effective types of peripherals to achieve the goal helped with understanding which data communication protocols were needed and the limitations to the build.

The final version of the block diagram traces the direction of communication and the types of data communication protocols needed for each device.

The block diagram contains 8 major components that make up the quadcopter drone:

1. Microcontroller - that is programmed to handle calculations and signals
2. Inertial Measurement Unit (IMU) Module - captures external senses with the use of the gyroscope and the accelerometer
3. Bluetooth Module - the communication between the user and the drone
4. JTAG Programmer - for installing all firmware and other necessary software to be able to program the MCU
5. Voltage Regulators (3.3V and 5V) - needed to convert the larger external voltage from the battery, our power supply, which is then fed to other peripherals. This is visible later in the block diagram
6. Electronic Speed Drivers - main purpose is to drive current to the motor but it also is in charge in controlling the behavior of the motor

7. Motors - responsible for providing enough force to the propellers of the drone for lift off or any real time adjustments during flight



A. Problem Discovered

Fig. 1 Block Diagram (Attempt#1)

The problem highlighted above in BLUE resulted in the peripheral not being able to communicate with the microcontroller.

9-axis IMU Module Problem - the data sheet of the IMU module mentioned that the module supported a Serial Peripheral Interface (SPI) communication protocol. Therefore, block diagram 1.0, as seen in Fig. 1., was created under the impression that the communication of data traveling to and from the module was as: master in slave out (MISO) and master out slave in (MOSI). As the layout of the printed circuit board was wired, we realized that one of the four wires connecting from the microcontroller and the IMU Module could not be connected. The reason behind this was that the IMU module was not using the conventional method of the SPI communication protocol. Standard SPI communication has connections/pins for several operations: MOSI, MISO, Slave Select (NSS), and Slave Clock (SCK). The IMU Module datasheet explained that it only had three pins available for SPI. The module had one pin which was meant to share between the MISO and MOSI connection [1,2,4].

B. Applied Solution and Final Product

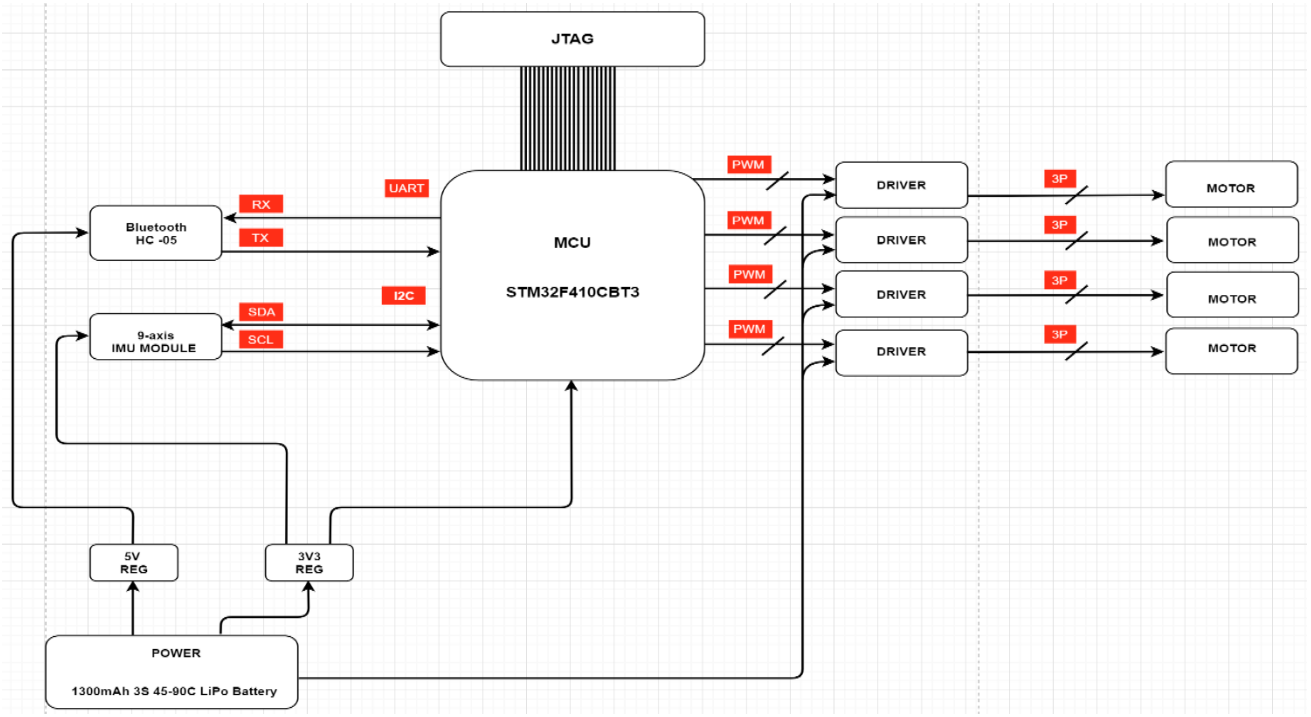
Fig. 2 Block Diagram (Attempt 2)

9-axis IMU Module problem solved - rather than keeping the original idea of using the SPI communication protocol and having to use software to solve the design issue. Therefore, we concluded that the best option was to replace the SPI protocol with the Inter-Integrated Circuit protocol (I2C) since the microcontroller had other available pins that supported I2C. As you can see in figure #, the 9-axis IMU Module implements I2C and labels all data communication protocols that follow the peripheral and microcontroller data sheet outlines[9].

Final Product - Creation of a high-level block diagram that labels all data protocol communications between peripherals and those connected to the microcontroller. All necessary components in the diagram are labeled for the ease of identifying parts used in the quadcopter drone. All new wiring connections have minimized cross wiring sections in order to avoid misconceptions of what it is doing or how it behaves.

III. Schematic

Our project consists of multiple attempts to our schematic. The creation of the schematic relied heavily on data sheets and the recommended schematics given by different manufacturers for selected components. Following this approach led us to overlook certain details in parts that may have been either too powerful for our intended purpose or not needed . Below discusses problems from our initial schematic, resources used to create the schematic, and improvements made in the second schematic.



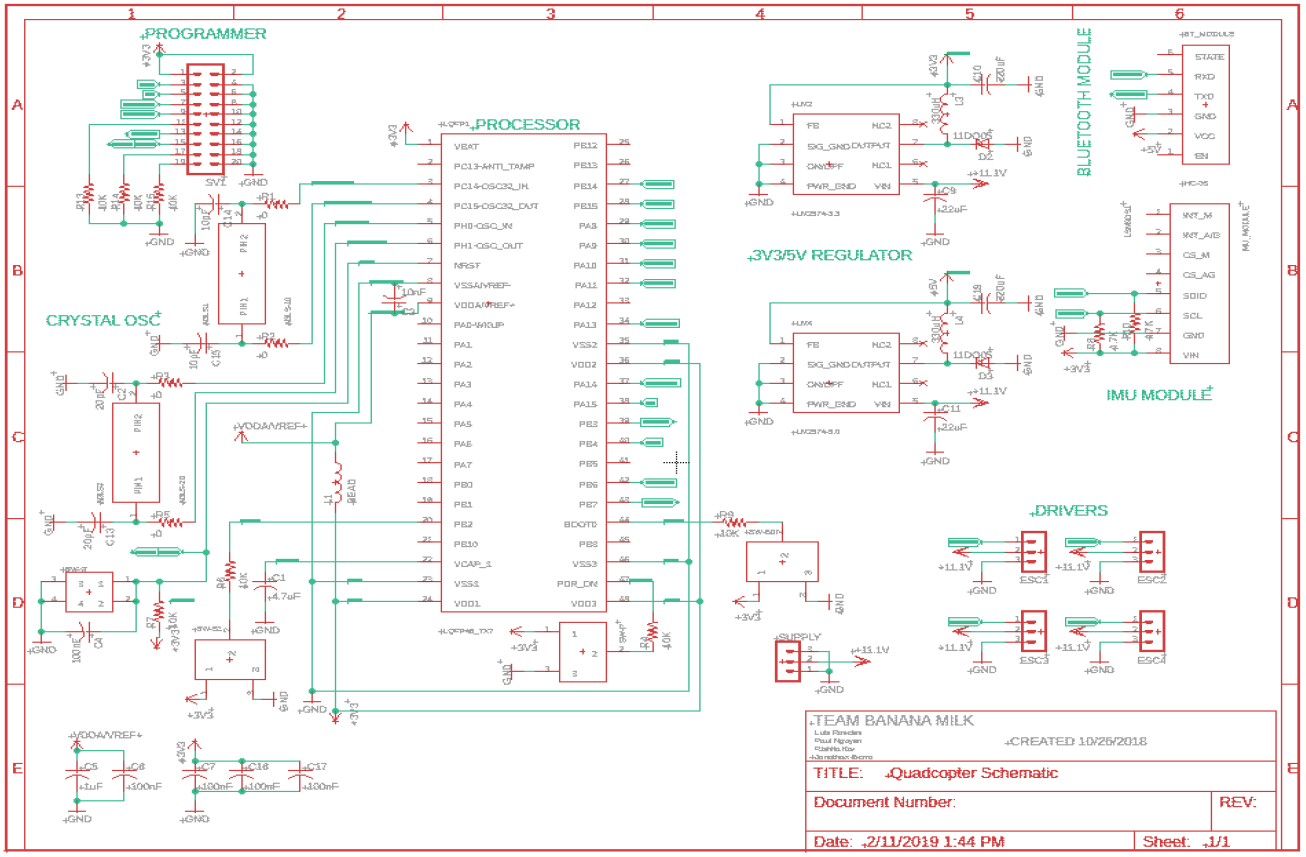


Fig. 3 Initial Schematic

A. High-Level Design Translated for Technical Application

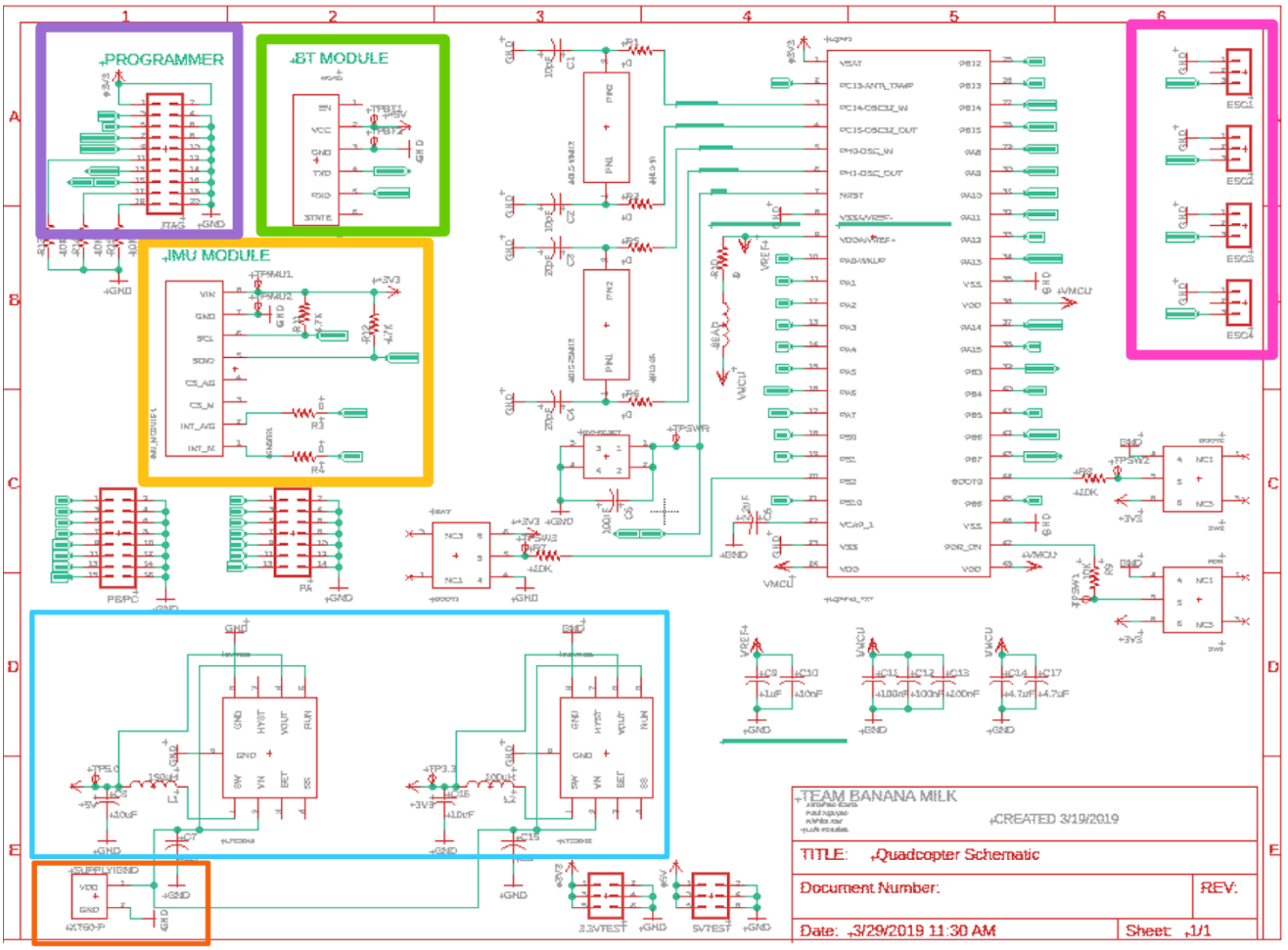


Fig. 4 Final Schematic

Fig. 4. displays the five major sections that make up the final design of the block diagram. Each section will be compared to the block diagram, Fig. 2.

- PURPLE - This is the schematic for the programmer. We know from the block diagram, figure 2, that we are working with a serial communication connection from the microcontroller (MCU) to the JTAG connection. Which is a connection along with labeling of wires, 20 wires, on the pins of the MCU and that of the JTAG so that the MCU can properly communicate with the J-Link programmer[11].
- BLUE - The switching regulators circuit necessary for the microcontroller and peripherals to operate. The regulators are actually the exact same part, but their internal circuits are configured in such a way so that they can output their fixed/designated voltage output. In Fig. 2. we can see that the 3.3V regulator connects to the MCU and to the IMU Module directly. Most of the components that are being used for the project involve 3.3V power supply. The 5V regulator connects from the power supply and then directs its voltage supply into the input of the Bluetooth module. Both circuits use a combination of a low pass filter and the capacitor decoupling technique. This application has allowed it so that the AC transient to become smaller. This type of circuit helps filter out the alternating frequency from high and low so that we only allow designated frequencies through the circuit.
- ORANGE - The power supply section, in Fig. 2., shows that it connects to the voltage regulators and to the drivers. This component is for an XT60 connector that is taking in the power supply and feeding it into the regulator and drivers.
- YELLOW - The circuit for the IMU Module. Part of the schematic for the IMU Module and throughout the rest of the major connections you see that there are several test points on major connections such as ground and power supplies. The IMU Module currently has internal pull up resistors, but for the device to be able to handle running longer period of time and higher speeds it requires to have two external pull up resistors. One resistor attached between the SCL pin then to the 3.3V power supply on the module and the second resistor attached between the SDIO pin then to the 3.3V supply.
- GREEN - The schematic for the Bluetooth module to operate is given. Based on the high-level block diagram, that the Bluetooth to communicate with the MCU needs the Universal Synchronous Asynchronous Receiver Transmitter (USART) data communication protocol. Overall, this schematic is handling data communication between the MCU and the Bluetooth module via the receiver and the transmitter signals/wires as seen in Fig. 4. The component is powered by the 5V regulator[3].
- PINK - The component for the ESC, also known as the drivers for the motors. In the high-level block diagram, we had a Pulse Width Modulation (PWM) data communication protocol to establish a line of data where the MCU sends data to the drivers. Generally, the communication of this data is sent in one direction, MCU to the ESC's, unless specified otherwise. The schematic here is simple because for our application we only care about the communication channel line and grounding the ESC's. The ESC has three wires GND, VDD, and PWM that connect to pin headers on the PCB in the initial schematic (Fig. 3). In the final schematic we omitted the VDD wire since we are using the voltage regulators to distribute power to the peripherals (Fig. 4).

B. Resources used in the creation of the schematic

The combination between data sheets of all devices and the suggested typical application schematic (known as the *getting started* resource sheet) provided by the STMICROELECTRONICS MCU manufacturer helped in completing the schematic for the quadcopter drone.

- Data sheets - the data sheets for all the devices provided typical application schematic. Links to components used and their datasheets are referenced in the COST OF MATERIALS section.
- *Getting started* resource sheet - This sheet was the most helpful because it provided the schematic for the MCU and the JTAG. It gave insights of how and why certain pins were wired in the configuration provided. https://www.st.com/content/ccc/resource/technical/document/application_note/76/f9/c8/10/8a/33/4b/f0/DM00115714.pdf/files/DM00115714.pdf/jcr:content/translations/en.DM00115714.pdf

C. Problems and Applied Solutions

Problem 1 - original voltage regulators (LM2574-3.3YN) were not appropriate for our application because the values of capacitors and inductors used were more than enough for the project, and the current output could damage the microcontroller.

Solution 1- the switching voltage regulator was exchanged to a reasonable component for the project. The new voltage regulators, 3.3V and 5V (LTC3642), now supports 50mA which is more than sufficient for all our peripherals and the MCU.

Problem 2 - In the initial schematic the VDD, GND, and PWM wires of the ESC were connected to the MCU. The error was the VDD wire connected directly to the MCU which would supply 5V to MCU which would damage the MCU. The MCU is only able to take 1.8V to 3.6V.

Solution 2 - omitted connecting the wire from the ESC's to the MCU to avoid frying the processor in the final schematic (Fig. 4).

Problem 3 - there was no consistency with resistor, capacitor, and inductor component sizes.

Solution 3 - resistor and capacitor were adjusted to 0805 size and inductors were adjusted to 1812 size.

Problem 4 - power supply connector in the first schematic was not appropriate for the PCB design. Pin headers will not be able to handle the inputted current and voltage.

Solution 4 - the power supply connector was made to support an XT60 connector.

Problem 5 - unused pins on the MCU were not being connected to any pin headers and it would be useful for debugging and testing additional components.

Solution 5 - adding pin headers for unused signal pins that connect to the MCU, test points for major components, and test pins to help with debugging.

IV. Layout

Remaking the schematic meant that the layout previously used with the old schematic design needed to be re-spun once to change errors and make modifications on the PCB after physically working with the first board. The new layout made improvements to routing, spacing, and clarity.

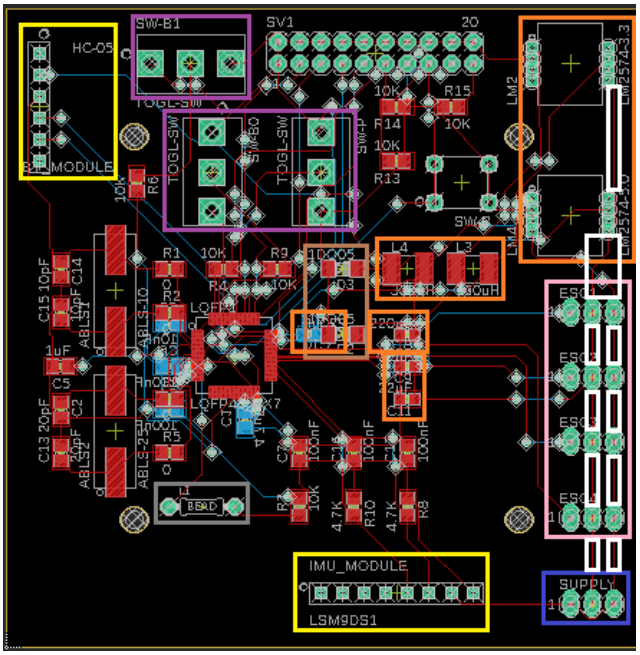


Fig. 5 First PCB Layout Design

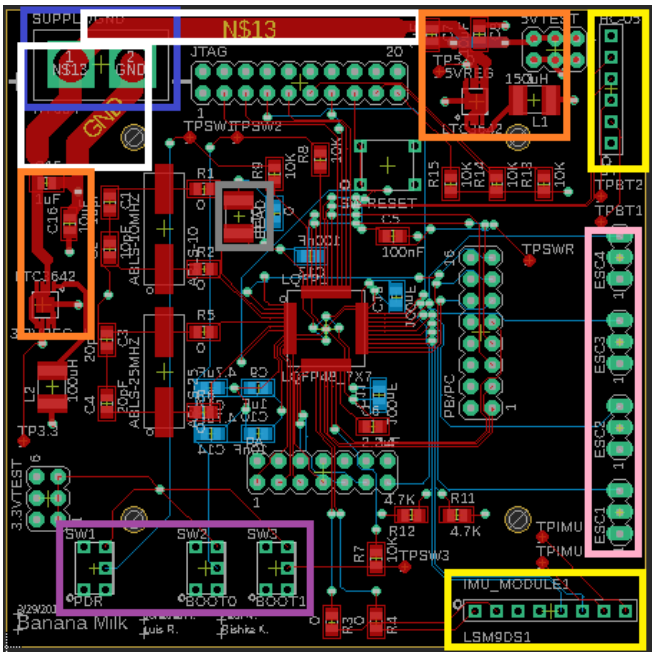


Fig. 6 Second (re-spun) PCB Layout Design

A. *Features of PCB Board*

- 75x75mm board
- Drill holes marked for attaching to drone chassis
- Four-layer PCB board: top, ground, supply, and bottom
- Design rule checking used: cmpe492-4Layer.dru file

B. *Differences Between Layout Designs*

The main differences from each layout are highlighted in each figure. The boxes in each figure are color coordinated based on specific changes.

- BLUE - The power supply connector was changed from 3x1 male pin header to an XT60 connector footprint because the pin headers are not able to handle the inputted current/voltage.
- WHITE - Wire width for power/ground lines was increased from 0.1524mm. to 2.54mm to address current/voltage.
- ORANGE - 3.3V and 5V voltage regulators were changed in the second design and capacitors and inductors needed to be updated for the LTC3642 component.
- YELLOW - Diameter length of the through hole pads for inertial measurement unit (IMU) and Bluetooth module was increased to fit pin headers.
- PINK - Orientation of electronic speed controller pins was changed to fit a right-angled 3x1 male pin headers.
- PURPLE - The toggle switch component was not the appropriate weight and size for the project, and it was altered to a more compact piece in the second design.
- GREY - Ferrite bead component was replaced with a L1812 size.

- BROWN - On the first layout the Zener diodes highlighted were removed after changing the voltage regulator components.

C. *Improvements in Second Layout*

Building a method for creating the layout helped make routing easier and tackled spacing issues that were apparent in the first layout:

1. The microcontroller was the first footprint placed on the board and extending straight wires from the pins of the microcontroller helped wiring around the component.
2. Placing pieces that are recommended to be close to the microcontroller such as crystal oscillators and decoupling capacitors.
3. Designating areas for different power sources. Most of the components of our design required 3.3V, so a small area on the top right corner was designated only for 5V components.
4. The location of major components that do not include resistors, capacitors, and inductors were placed on sides where they would route to the microcontroller that require the least amount of vias.
5. Resistors, capacitors, and inductors were then placed.
6. Connecting wires starting with the extended straight wires from the microcontroller pins.

In addition to following the method above, the first layout had issues in the following areas that were improved in the second re-spin:

- Layers: The most notable fault of the first layout was that the design did not take advantage of the four-layer feature of the PCB. Routing in the first design was strictly done on the top and bottom layers and the ground and supply layers were hardly used. This made routing more challenging than it needed to be.
- Vias: Hand soldering on the first PCB design was challenging because some vias were too close to some components that caused issues with shorting. Vias were carefully created with a generous amount of space from components in the second design.
- Test points: Test points were added to the second design to test if power was supplied correctly in different areas of the board.
- Unused pins: Not all pins on the microcontroller were used, so pin headers were connected to these unused pins that can be effective for testing or adding new components.
- Component sizes: Resistors and capacitors were all adjusted to meet the 0805 size for hand soldering.

The second layout was adjusted to solder components easily by hand. Issues with soldering were observed from the physical board of the first design and were improved in the second design.

There was an issue with the second layout that could be improved in a third re-spin and that included adding a 3.3V connection to the VDD pins of the MCU. Connecting 3.3V to the VDD pins fixes problems for boot-up and programming. Another issue was creating an incorrect footprint for the reset button. The reset button needed to be oriented 90 degrees with the pin orders changed.

V. Cost of Materials

Bill of materials - $206.59
https://docs.google.com/spreadsheets/d/1VZDD2ZmMi-CXOhbZjpYoq57kjEzAHEIRHx6kEwua0M4/edit?usp=sharing
Cost to produce board (JLC PCB) - $34 for 10 boards

VI. Quadcopter Software

A. *Physical Theory*

The structure of the quadcopter uses two pairs of identical fixed pitched propellers; two clockwise and two counterclockwise. Fig. 6.1 The quadcopter flies in its own reference plane and demonstrates roll, pitch and yaw movement in the x, y and z axis respectively. Fig. 6.2.

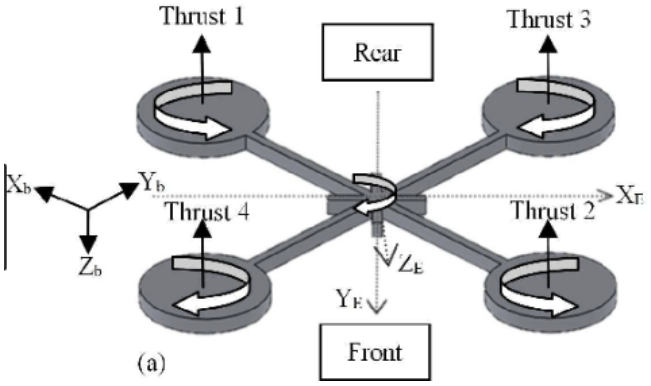


Fig. 6.1 Structure and direction of blade rotation in a Quadcopter

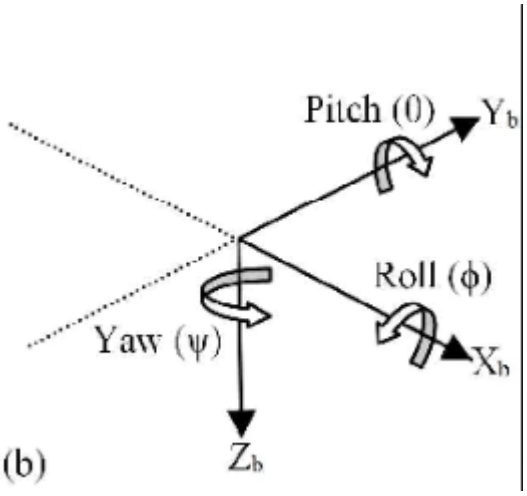


Fig. 6.2 Axis of Roll, Pitch and Yaw angles

B. Physics Behind Movement

As mentioned in section VI. Part A, there are different types of movement a quadcopter has. These include taking off, touching down, hovering and lastly, rolling, pitching and yawing in their respective axis. How these movements are carried out depends on the velocity of the rotating blades to change the thrust force generated by the quadcopter to carry out the respective movements. [8]

1) Taking off, Touching down and Hovering

To get the drone to take off, all four blades need to rotate rapidly with the same velocity to generate a thrust force greater than the force generated by the weight of the quadcopter. Fig. 6.3 Similarly to touch down, the blades need to decrease their velocity to generate a thrust force less than the force generated by the weight of the quadcopter. Finally, whilst airborne, the quadcopter "hovers". This is also called the equilibrium condition where the blades rotate with a velocity that generates a thrust force which is equal to the weight of the quadcopter. Fig. 6.4 [8]

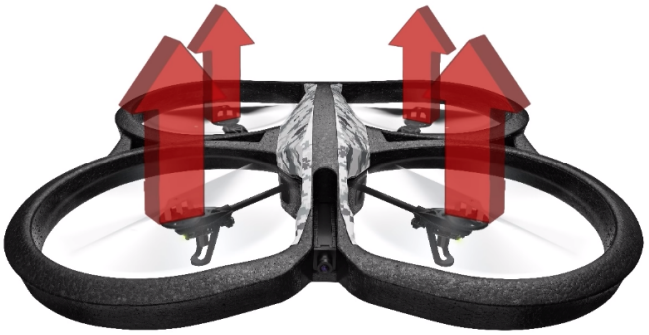


Fig. 6.3 Quadcopter take off condition

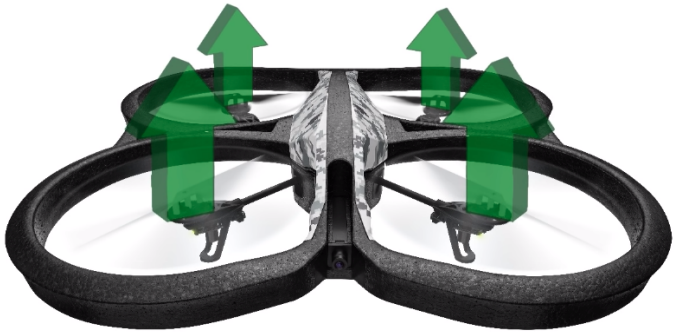


Fig. 6.4 Quadcopter hover condition

2) Roll

Considering we are facing the quadcopter and we are talking with respect to the quadcopter's frame

The quadcopter rolls (tilts left or right) about the x-axis. To roll to its left, the quadcopter increases the velocity of the front left and back blades and decreases the velocity of the front and back right blades. This is because the quadcopter tries to keep its net torque zero, thus resulting in a leftward net force. Fig. 6.5 To roll to its right, the quadcopter does the opposite.

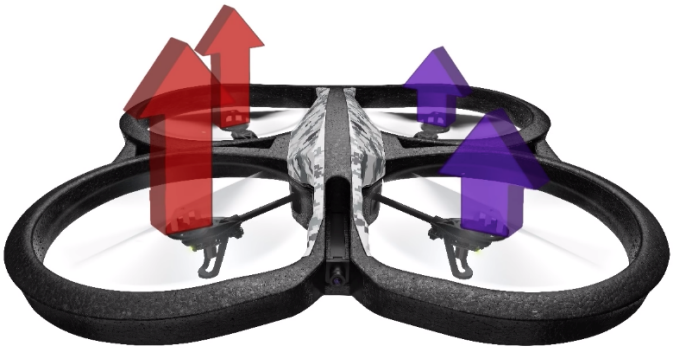


Fig. 6.5 Quadcopter roll left condition

3) Pitch

Considering we are facing the quadcopter and we are talking with respect to the quadcopter's frame

The quadcopter pitches (tilts forward or backward) about the y-axis. To pitch forward to move towards us, the quadcopter increases the velocity of the back left and back right blades and decreases the velocity of the front left and front right blades. This creates a net forward force which causes the drone's nose to pitch downward. Also, decreasing the velocity of the front blades makes the quadcopter conserve angular momentum thus causing it to pitch. Fig. 6.6 To pitch backward away from us, the quadcopter does the opposite.

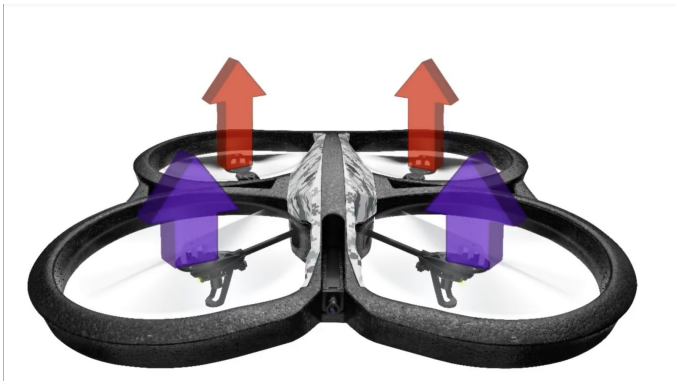


Fig. 6.6 Quadcopter pitch forward condition

4) Yaw

Considering we are facing the quadcopter and we are talking with respect to the quadcopter's frame

The quadcopter yaws (rotates clockwise or counterclockwise) about the z-axis. To yaw clockwise, we increase the velocity of the counterclockwise rotating blades and decrease the velocity of the clockwise rotating blades. The reason being that the quadcopter tries to keep the net upward or downward force zero. The increase in velocity of the counter-clockwise blades generates a counter-clockwise angular momentum, hence, to balance it, the quadcopter rotates clockwise to conserve angular momentum. Fig. 6.7 To yaw counterclockwise, the quadcopter does the opposite.

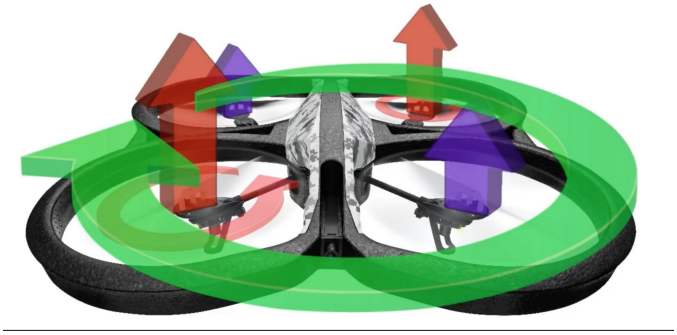


Fig. 6.7 Quadcopter yaw clockwise condition

C. *Mathematical Application of Physical Theory Autonomous Flight*

The first step in successful quadcopter flight is the stability of the quadcopter in space while maintaining its position. Controlled variations from this set point will naturally translate into motion. Prior explanation of physical theory explains not only how the motor outputs can applied for motion, but also for correction. The same thrusts applied for roll, pitch, and yaw will be applied to correct deviations from an inertial reference plane. Therefore, before we venture into quadcopter movement, we will discuss the PID loop utilized to calculate motor outputs for quadcopter stability. Since, the physical theory has provided the relevant context to understand the underlying mathematics, we will be able to explain the software directly from the mathematics. Fig. 7.8

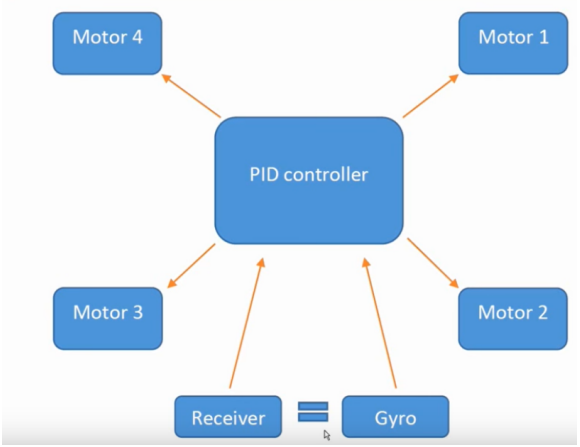


Fig. 7.8 General overview of the PID control on drone motors

D. *Coding the Motor Inputs*

We will begin at the end; with the formula for each motor output. The output for each motor is as follows:

t5 = dutyCycle + (pid_out_roll/100) - (pid_out_pitch/100) - (pid_out_yaw/100); // checking esc 1
t6 = dutyCycle + (pid_out_roll/100) + (pid_out_pitch/100) + (pid_out_yaw/100); // checking esc 2
t7 = dutyCycle - (pid_out_roll/100) + (pid_out_pitch/100) - (pid_out_yaw/100); // checking esc 3
t8 = dutyCycle - (pid_out_roll/100) - (pid_out_pitch/100) + (pid_out_yaw/100); // checking esc 4

To simplify our understanding of autonomous stabilization, we categorize necessary adjustments to the X, Y, and Z planes with their respective Roll, Pitch, and Yaw deviations and their axes'.

Since the X plane deals with pitch, motor outputs relative to calculated error will affect only the front and rear motors. Since the Y plane deals with yaw, motor outputs relative to calculated error will affect only the clockwise and counterclockwise motors. And since the Z plane deals with roll, motor outputs relative to calculated error will affect only the right and the left motors.

Now that we understand how these corrections are applied, we will go into more detail about the different type of corrections applied to these axes. As the PID loop implies, we will explain the proportional integral and derivative calculations and their resulting corrections. Fig. 7.9

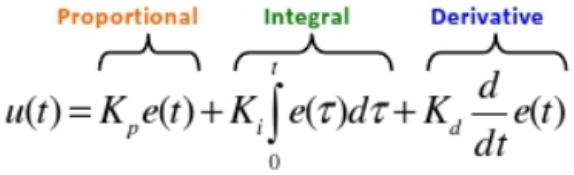


Fig. 7.9 Formula for the PID loop

E. *Roll, Pitch, and Yaw calculations using PID Loop*

To make things easier to understand, let us frame each possible correction in the form of a question. We use proportional adjustment when we want to know how intense a correction should be. We use integral adjustment when we want to know how to reduce the duration in which the quadcopter has deviated from the set point. And we use derivative adjustment when we want to know how fast we should react to changes deviation values. Mathematically, we take advantage of computing power and keep things simple by approximating each adjustment arithmetically with the following equations.

Observe that a proportional adjustment is simply error multiplied by our proportional gain or 'how intense we want our correction to be'. As things become a bit more sophisticated with integral adjustments, we simply sum corrective outputs with errors accrued over time to approximate our integral. Derivative adjustments, in line with our physical understanding that the derivative represents rate of change, is the difference between error values multiplied by 'how fast our quadcopter should react'. This allows us to simplify the math with the following software calculations. Fig. 7.10.

- Roll Calculations

```
// roll calculations
pid_error_temp = gyro_roll_input - pid_roll_setpoint;

t2 = pid_i_gain_roll * pid_error_temp;
pid_i_mem_roll += t2;

if (pid_i_mem_roll > pid_max_roll)
{
  pid_i_mem_roll = pid_max_roll;
}
else if (pid_i_mem_roll < pid_max_roll * -1)
{
  pid_i_mem_roll = pid_max_roll * -1;
}

pid_out_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll + pid_d_gain_roll * (pid_error_temp - pid_d_last_roll_error);

if (pid_out_roll > pid_max_roll)
{
  pid_out_roll = pid_max_roll;
}
else if (pid_out_roll < pid_max_roll * -1)
{
  pid_out_roll = pid_max_roll * -1;
}

pid_d_last_roll_error = pid_error_temp;
```

- Pitch Calculations

```
// pitch calculations
pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;

t2 = pid_i_gain_pitch * pid_error_temp;
pid_i_mem_pitch += t2;

if (pid_i_mem_pitch > pid_max_pitch)
{
  pid_i_mem_pitch = pid_max_pitch;
}
else if (pid_i_mem_pitch < pid_max_pitch * -1)
{
  pid_i_mem_pitch = pid_max_pitch * -1;
}

pid_out_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch + pid_d_gain_pitch * (pid_error_temp - pid_d_last_pitch_

if (pid_out_pitch > pid_max_pitch)
{
  pid_out_pitch = pid_max_pitch;
}
else if (pid_out_pitch < pid_max_pitch * -1)
{
  pid_out_pitch = pid_max_pitch * -1;
}

pid_d_last_pitch_error = pid_error_temp;
```

- Yaw Calculations

```
// yaw calculations
pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;

t2 = pid_i_gain_yaw * pid_error_temp;
pid_i_mem_yaw += t2;

if (pid_i_mem_yaw > pid_max_yaw)
{
  pid_i_mem_yaw = pid_max_yaw;
}
else if (pid_i_mem_yaw < pid_max_yaw * -1)
{
  pid_i_mem_yaw = pid_max_yaw * -1;
}

pid_out_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw + pid_d_gain_yaw * (pid_error_temp - pid_d_last_yaw_error);

if (pid_out_yaw > pid_max_yaw)
{
  pid_out_yaw = pid_max_yaw;
}
else if (pid_out_yaw < pid_max_yaw * -1)
{
  pid_out_yaw = pid_max_yaw * -1;
}

pid_d_last_yaw_error = pid_error_temp;
```

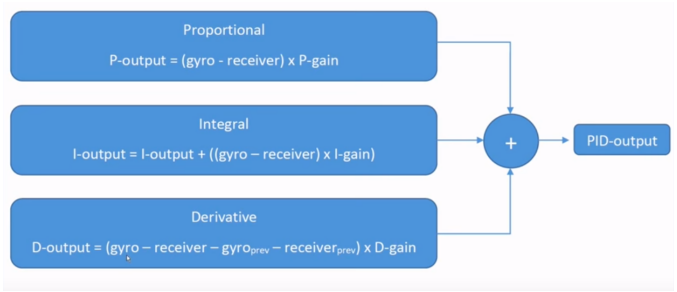


Fig. 6.10 PID Calculation for roll, pitch and yaw

F. PID Error Calculations

Now that we have established the physical theory and have explained the mathematics behind the motor outputs necessary to maintain equilibrium, one can now easily understand the required imbalance necessary to impose to on the quadcopter to create motion. As we have seen, the integrity of the PID calculations depend on the accurate sensing of quadcopter orientation with respect to the inertial reference frame and error implied by its difference. Changing the set point of this inertial reference frame is the 'required imbalance' imposed to control quadcopter motion.

- pid_error_roll = angle_roll_gyro - pid_roll_setpoint;
- pid_error_pitch = angle_pitch_gyro - pid_yaw_setpoint;
- pid_error_yaw = angle_roll_gyro - pid_roll_setpoint;

The last set of mathematical calculations involve the translation of raw gyroscope and accelerometer values into angle measurements with respect to the inertial reference frame. Since we have only recently powered and booted our official board, we will discuss the derivation of this angle with our prototype gyroscope and accelerometer that we have prototyped with the Arduino IDE. [5][6]

G. Other Calculations

- Reading Sensor Values: Arduino libraries are used to read input values from the gyroscope and accelerometer. Code from example sketches provided by Arduino, namely sensorapi and acceldemo are combined and utilized. Values are given in radians per second. Since degrees per second is easier to understand we multiply the values by 180/pi. Max Value, Min Value and Resolution can now be printed and understood in degrees per second.

- Calculating Angles: In order to calculate error values needed for PID input, sensor values need to be utilized to derive angular displacement with respect to the inertial reference frame. Angular displacement calculated from a gyroscope tend to drift with consequent loss of accuracy. On the other hand, angular displacement calculated from an accelerometer can be compromised by noise generated from the quadcopter. To compensate for both downsides, angular displacement calculated by the accelerometer is used to correct drift from the gyroscope. The updated angular displacement values from the updated gyroscope provide noise resistant reliable values. We will now discuss the calculation of angular displacement with the accelerometer and gyroscope respectively.

- Accelerometer and Gyroscope Calculations:

```
// Fixing angles using calibration code and raw gyro outputs
gyro_fixed_x = myGyroRaw.x - gyro_x_cal;
gyro_fixed_y = myGyroRaw.y - gyro_y_cal;
gyro_fixed_z = myGyroRaw.z - gyro_z_cal;

acc_fixed_x = myAccelRaw.x - acc_x_cal;
acc_fixed_y = myAccelRaw.y - acc_y_cal;
acc_fixed_z = myAccelRaw.z - acc_z_cal;
```

- Gyroscope PID Inputs:

```
// gyro PID input
gyro_roll_input = (gyro_roll_input * 0.7) + ((gyro_x / 65.5) *0.3);
gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_y / 65.5) *0.3);
gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyro_z / 65.5) *0.3);
```

VII. MILETSONES

The following is the list of milestones achieved by the team so far in their progression through the CMPE 492-493 project term:

- MILESTONE 1: Proposal

The following milestone was an overview of what the team wanted to design for the CMPE 492 project and what its functionality might be. This was achieved in the first two weeks of the first semester.

- MILESTONE 2: Work Distribution and Behavioral Specification

The following milestone had a top level block diagram, depicting the quadcopter, its subsystems and how they would communicate with each other

- MILESTONE 3: BOM, Subsystems, code workflow and pseudocode

The following milestone had the team create a bill of materials for the potential parts they will be using for the PCB. It also involved talking about each subsystem in detail and giving some form of pseudocode on how we are to communicate with the values given by each of the subsystems.

- MILESTONE 4: Designing PCB and Prototype code

The following milestone involved designing the PCB along with its subsystems on Eagle. The Schematic and layout of the PCB is designed in this milestone and after approval, sent for manufacturing. The milestone also involved having functional code on prototype boards like an Arduino.

- MILESTONE 5: Soldering parts

Once the manufactured PCB was received, the team spent the remaining time in the semester to solder parts on to the board.

- MILESTONE 6: Getting the MCU to power up and boot

Once the voltage regulators were soldered into the board along with the MCU, power was supplied to the board and its circuitry was checked. Test points were tested for connections.

Following that, the pinouts were soldered onto the board for the booting process. The team failed there and after troubleshooting, we realised that the MCU Vdd was not receiving the 3.3V. To correct the problem, we soldered a wire from to the via to supply the MCU with the appropriate voltage. We attempted to boot the MCU and successfully booted it.

- MILSTONE 7: Boot code for software into MCU

Using STM 32 Cube MX and Keil tools, we successfully generated a boot code which would flash an LED thus making sure that we can communicate with our hardware using software.

- MILESTONE 8: – BIOS level monitor or "operating system": After generating the code, we ran our "blink program" and successfully flashed an LED thus moving on further towards implementing code for the remaining subsystems.

- MILESTONE 9: Individual subsystem tests: After generating code for each of the subsystems, we tested them using a bunch of programs, for example to test whether we were able to send/receive Bluetooth signals, we were using the Keil IDE to program and Realterm to see our outputs. Similarly, using the Keil IDE we ran the code to give us motor, PID and sensor outputs and we saw our values on STM Studio.

- MILESTONE 10: Integration of subsystems: After testing the code for the individual subsystems, we had to bring in all of the code together in order to see our project work when all of the subsystems are working together. For this, we sent a Bluetooth signal which would turn on the quadcopter, arm the motors and increment the speed in a particular time interval until it reaches a threshold. After that, the motors will decrement their speed in the same time interval until it completely stops.

- MILESTONE 11: Full Application:

After integration of all subsystems, we are tuning the PID loop in our flight controller to achieve stable flight. So far we can see the motors change according to how we move our quadcopter thus proving that the PID loop successfully applies corrections to the motors with respect to how the quadcopter moves.

- MILESTONE 12: Project demonstration, technical report, and oral presentation

The presentation is on the day this journal is being submitted. We look forward to showing the implementation of our drone.

VIII. Reach Goals

Our reach goal for this project would be to have the drone GPS enabled to receive pre-programmed coordinates via Bluetooth signal. The signal will be sent using a phone app or a laptop and will resemble high level commands which are easy for the user to send, for example the user might just have to enter coordinates or tell the drone how far to fly and the drone will do as signaled to.

As far as reaching the following goals go, our team member Luis has investigated and somewhat started structuring the Android app that will send high level commands to the quadcopter. The team was a little behind and had to put the reach goal on hold in order to achieve the other milestones.

IX. Conclusion

The Quadcopter Drone Project is challenging and equally enjoyable. The MCU provides computational power and storage. A warning for future engineers and software developers, review any and all documentation related to devices. It is essential to brainstorm prior to the selection of devices. Rule of thumb, at least in our case, start with the microcontroller and expand on necessary peripherals based on the needs of the application of the device. The two sensors: the Bluetooth and the inertial motion unit provide input commands, acceleration, angular velocity, and magnetic field strength. The four motors receive output information and enables our quadcopter to fly.

This journal documents a top-level schematic, PCB layout, subsystem block diagrams and software theory and code that will utilized to stabilize and control the quadcopter.

Proceeding into the next stages, we hope to program the drone itself while carrying out tests on the side to have it fully functional by presentation day. Software development is progressing steadily, and multiple boards have been soldered to carry out tests. We are expecting to fly the Banana Milk's Quadcopter by the end of Spring 2019.

X. References

[1] Circuit Basics, “Basics of the SPI Communication Protocol,” end.com, Feb. 13, 2016. [Online]. Available: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol. [Accessed March 12, 2019].

[2] Circuit Basics, “Basics of the I2C Communication Protocol,” end.com, Feb. 13, 2016. [Online]. Available: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol. [Accessed March 12, 2019].

[3] GM Electronic, “HC-05 Bluetooth Module,” 772-148 datasheet, Nov. 2008.

[4] J. Beningo, “USART vs UART: Know the difference ,” end.com, Sep. 21, 2015. [Online]. Available: http://www.edn.com/electronics-blogs/embedded-basics/4440395/USART-vs-UART--Know-the-difference. [Accessed March 12, 2019].

[5] J. Brokking, YMFC - 3D part 5 - Quadcopter PID controller and PID Tuning, May 8, 2015. Accessed on: March 12, 2019. [streaming video]. Available: youtube.com database

[6] J. Brokking, YMFC - 3D part 6 - Build your own Arduino quadcopter flight controller with source code, May 16, 2015. Accessed on: March 12, 2019. [streaming video]. Available: youtube.com database

[7] Linear Technology, “High Efficiency, High Voltage 50mA Synchronous Step-Down Converter,” LTC3642 datasheet, Nov. 2008.

[8] N. Johnson-Laird, Basic Physics of Drones, March 9, 2016. Accessed on: March 12, 2019. [streaming video]. Available: youtube.com database

[9] Parallax Inc, “Parallax LSM9DS1 9-axis IMU Module,” 28065 datasheet, Nov. 2008.

[10] Robotics Stack Exchange, “How to convert PID outputs to appropriate motor speeds for a quad copter,” robotics.stackexchange.com, Oct. 14, 2014. [Online]. Available:http://robotics.stackexchange.com/questions/4721/how-to-convert-pid-outputs-to-appropriate-motor-speeds-for-a-quad-copter. [Accessed March 12, 2019].

[11] STMicroelectronics, “Getting Started with STM32F4xxxx MCU hardware development,” AN4488 datasheet, Nov. 2018.

[12] STMicroelectronics, “Arm-Cortex-M4 32b MCU+FPU, 125 DMIPS, 128KB Flash, 32KB RAM, 9 TIMs, 1 DAC, 1 LPTIM, 9 comm. interfaces ,” 028094 Rev 6 datasheet, Nov. 2017.