FLYING COYOTE

Abdul Nasir, Chi Nuo Yen (Isaac), Jared Cruz, Paul John Evangelista, Trejon Adams, Willy Huang

ENGR 7B: Introduction to Engineering
University of California, Irvine
Winter 2017

Table of Contents:

Executive Summary

The goal of our project this quarter was to build a quadcopter that was capable of flying inside a cage while detecting the location and height of platforms to light up an LED panel based on the readings. This report will go over our project management, design, fabrication, testing, and results of the overall product.

We completed the structure of our quadcopter during week 7. For the design of our quadcopter we put a major focus on the rigidity of the structure in order to achieve stable flight and avoid getting caught in the netting of the cage. We also made sure to have a well sized frame in order to house all of the electronics. Unfortunately, we did not put enough emphasis on wire management within our design so it ended up being more messy than we anticipated.

The initial testing of our arduino software proved to be very successful during the quarter. We tested the code without flight during week 10, and we were able to successfully light up the LED panels.

The hard work that all of our team members put forth was very essential during the duration of the project.

Problem Definition

Introduction:

The goal of this project was to build on the basic quadcopter knowledge that we acquired in Engineering 7A and implement our new knowledge of programming to manufacture a new quadcopter with the capabilities of autonomous mapping.

Technical Background:

The origin of quadcopter can be dated back toward the early 20th century, where several prototypes of multi rotors aircraft were developed. The Oehmichen No. 2 was one of the earlier multirotor crafts that were developed. It flew 1 kilometer in a closed circuit path. However, because of poor performance, high pilot workload, and insufficient demand, the development of quadcopter did not become popular until the beginning of the 21st century. The availability of miniaturized flight controller and microcontroller makes it practical for quadcopter to be built as they give pilots precise control over their quadcopter. Some of the popular brands for quadcopter include Parrot and DJI. Current quadcopters are being built for various purposes including geographical surveying, search and rescue, delivery, and drone racing. For instance, DHL developed a quadcopter drone prototype that is to be used for drone delivery. Quadcopters' versatile capabilities and miniaturized sizes allow difficult tasks to be completed with affordable costs.

Design Requirement:

- The quadcopter must possess four motors with a choice of a set of four 5 inch propellers, 6 inch propellers, and 8 inch propellers.
- Propeller guards and fail-safe control must be included in the quadcopter to ensure the safety of the individuals in close proximity of the quadcopter
- The quadcopter must include an onboard autonomous mapping system powered by the Arduino microcontroller with inputs from three ultrasonic sensors.
- Wires and connectors must be fully insulated
- A organized electrical management is needed to ensure the ease of access for the quadcopter electrical specialist.
- The quadcopter must have landing gears.
- An accelerometer is needed to permit the quadcopter to provide horizontal headings.

- The quadcopter must achieve a flight time of 5 minute with power supply by a battery pack.
- The motor to motor distance must be less than 14 inches.
- Quadcopter must be fabricated in lab, excluding 3D printed parts.
- Combustion engines are not allowed
- The flight altitude must be less than 10 ft.
- Preferred altitude is 3 ft to 8 ft.
- The quadcopter must cost under $500 or $550 for 3D printed parts, and the cost must be broken down into parts list.

Design Description:

Summary of Design:

Flying Coyote's design is a x-configuration quadcopter built with a combination of materials designed for maximum durability and flight time performance. Using an initial quadcopter design from last quarter, revisions were made in order to minimize the mass of the aircraft while still selecting strong and rigid materials in order to ensure that the quadcopter would not only perform well in flight, but also hold up under rough conditions. The estimated mass of the quadcopter is 200 grams, or 0.7 kg, with an edge to edge distance of 15.5 inches. The total cost including all electronic components was $399.98.

Figure 1: Finalized quadcopter design

Design Details:

The main body is an icosagon made with a layered combination of polycarbonate, choroplast, and pink foam for maximum strength under stress and strain. Additionally, the propeller guards have a dual ring design made also of pink foam, while the landing gear are a combination of foam with sheets of polycarbonate glued to each side. The motors are mounted onto the top of the

mainframe inside the prop guards through four 1.5 inch screws secured by nuts on the other side. On top of it is a 6 inch propeller with locknuts to prevent it from sliding. There are a total of 4 motor sets mounted in such fashion. The battery holder was fabricated by a strip of polycarbonate of 0.16 inches, shaped into a rectangular hook through the use of a heatstrip and glued onto bottom side of the frame. The ultrasonic sensor holders were made from polycarbonate pieces with cut-outs for the sensors glued onto the quadcopter frame with L-shaped angle brackets. The Arduino, XBEE shield, ultrasonic sensors, and IMU are mounted together through soldering and pin connections while the Arduino board is mounted directly onto the quadcopter through standoffs and screws.

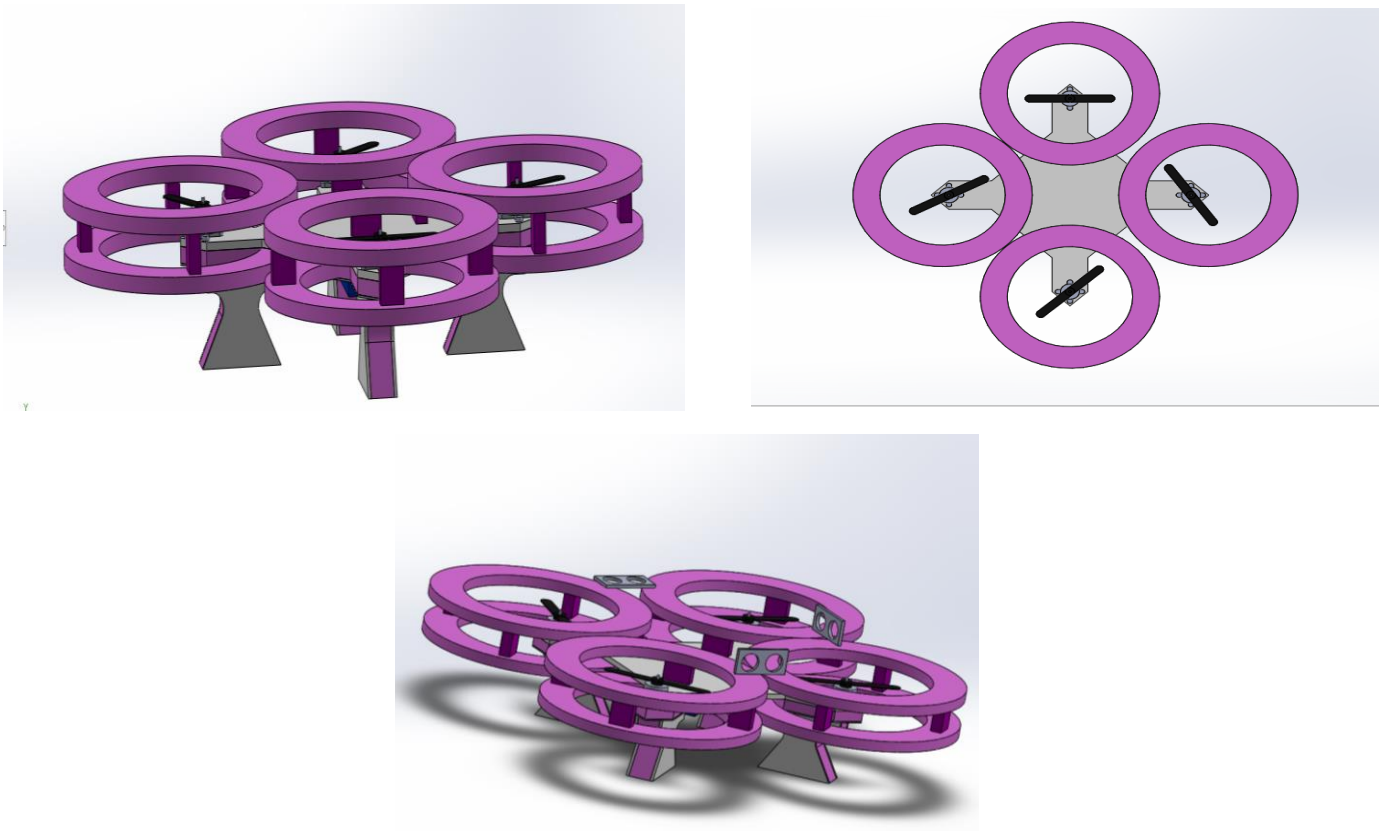


Figure 2: Solidworks models of quadcopter

(See Appendix A for quadcopter CAD drawings with dimensions)

Wiring Diagram:

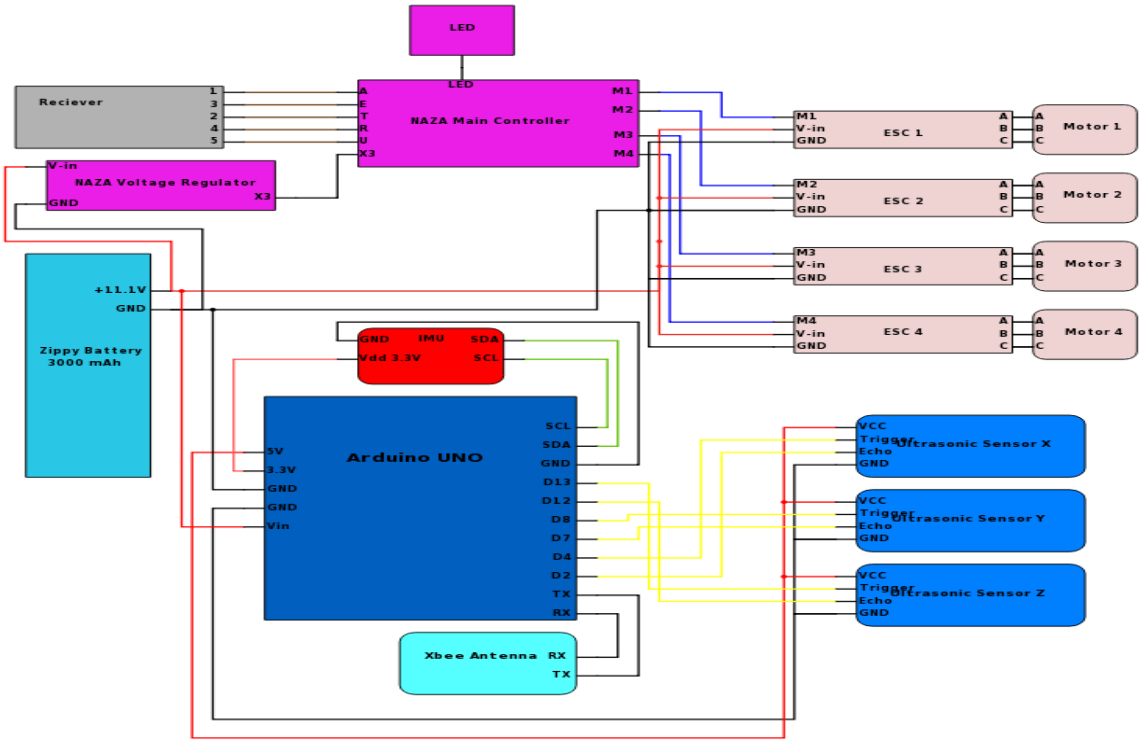


Figure 3: Wiring diagram indicating all electrical components

Algorithm Design:

The mapping algorithm begins by including several libraries needed for utilizing the IMU sensor and the ultrasonic sensors as well as defining the variables in the code as float or integer for the future arithmetic calculations in the program. The void setup will begin its initial heading and serial monitor. If the IMU does not start, it will print several lines to alert the user to check the wiring connections. If the IMU starts, it will proceed to go into the void loop. In the void loop, the 3 ultrasonic sensors will send their input back into the Arduino. If the readings are within a specific range, it will assign the corresponding quadrant from quadrant 1 to quadrant 4 and color ranging from red, blue, green, and white in the program. The quadrant and color will go through a series of calculations to send the right data of quadrant and color onto the external Xbee receiver, in which the Xbee receiver will convert the data and light up the corresponding color in the corresponding panel. In addition, the loop receives constant updates from the IMU regarding the heading of the quadcopter. If it is under a 15 degree negative or positive range, it will alert the pilot that the quadcopter is aligned. If it is outside that range, the Arduino will warn the pilot that the quadcopter is not aligned. The loop will continue repeating at a 10 second speed.

- Pseudo Code:

- Include Library:
    - Ultrasonic
    - Wire
    - SPI
    - SparkFunLSM9DS1
- Define Ultrasonic Input and Output
    - Ultrasonic 1(Downward) output pin is 12 while input pin is 13
    - Ultrasonic 2(Left) output pin is 8 while input pin is 7
    - Ultrasonic 3(Forward) output pin is 4 while input pin is 2
- Define LSM9DS1 as imu
- Define float object: roll, pitch, heading, AX, AY, AZ, MX, MY, MZ, ultra1, ultra2, ultra3, ult1, ult2, ult3;
- Define LSM9DS1_M 0x1E, LSM9DS1_AG 0x6B, PRINT_CALCULATED, PRINT_SPEED 250, and DECLINATION -12
- Define quad and color as integer
- Void Setup
    - Begin Serial monitor at 9600 speed
    - imu.commInterface is IMU_MODE_I2C
    - imu.mAddress is LSM9DS1_M;
    - imu.agAddress is LSM9DS1_AG
    - If IMU do not begin:
    - Serial PrintLn: Failed to communicate with LSM9DS1
    - Serial PrintLn: Double-check wiring
    - Serial PrintLn: Default settings in this sketch will
        - work for an out of the box LSM9DS1
        - Breakout, but may need to be modified
        - if the board jumpers are.
    - Wait forever
- Void Loop:
    - Serial Print Ultrasonic 1 in cm and cm
    - Serial Print Ultrasonic 2 in cm and cm
    - Serial Print Ultrasonic 3 in cm and serial println cm
    - Call printGyro function
    - Call printAccel function
    - Call printMag function
    - Print Attitude including imu.ax, imu.ay, imu.az, -imu.my, -imu.mx, imu.mz
    - Serial println to print next line
    - Convert Serial ultrasonic1 to ult1 variable
    - Convert Series ultrasonic2 to ult2 variable
    - Convert Series ultrasonic3 to ult3 variable
    - Set ultra1 as 0
    - Set ultra2 as 0
    - Set ultra3 as 0
    - Convert ult1 to radians; equal the result to ultra1

- Convert ult2 to radians; equal the result to ultra2
- Convert ult3 to radians; equal the result to ultra3
- If ultra2 >0 and ultra2 <=5 and ultra3>5 and ultra3 <=10
    - quad=1
- If ultra2 >5 and ultra 2 <=10 and ultra3 >5 and ultra3 <=10
    - quad=2
- If ultra2 >5 and ultra2 <=10 and ultra3 >0 and ultra3 <=5
    - quad=3
- If ultra2>0 and ultra2 <=5 and ultra3 >0 and ultra3 <=5)
    - quad=4

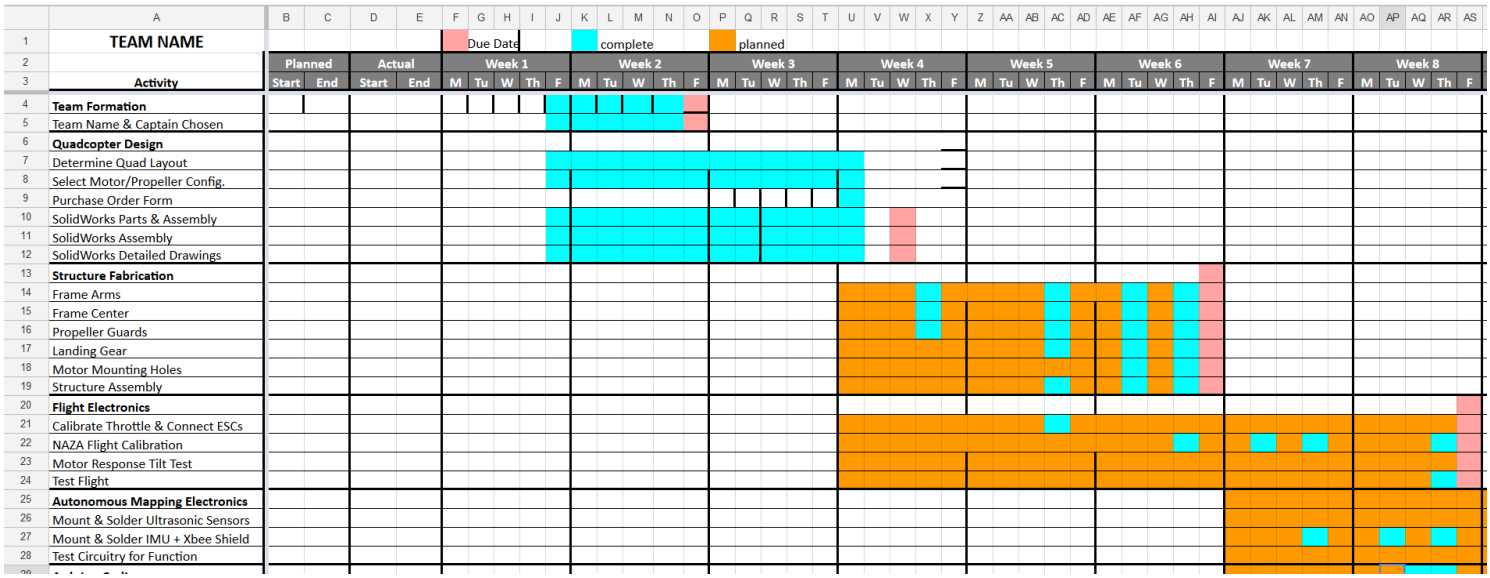
- If 3.5=>ultra1 > 2ft
    - Color is 4 blue
- Else If 5=> ultra1> 3.5
    - color is 3 green
- Else If 6.5=>ultra1> 5
    - color is 2 red
- Else if ultra1>6.5
    - color is 1 white
- String E equals character a
- Serial write a
- Serial write quad with integer division by 256
- Serial write quad with remainder division by 256
- Serial write color with integer division by 256
- Serial write color with remainder division by 256
- Delay 10
- Obtain printGyro from LSM9DS1_Basic_12C sample code
- Obtain printAccel from LSM9DS1_Basic_12C sample code
- Obtain printMag from LSM9DS1_Basic_12C sample code
- Obtain printAttitude from LSM9DS1_Basic_12C sample code
    - within this code, convert 15 degree to radian, modified the sample lines below
    - If heading <-15 degree in radians
        - Serial print “Quad not aligned”
    - If heading >15 degree in radians
        - Serial print “Quad not aligned”
    - Else
        - Serial print “Quad aligned”
- Flowchart will be attached

Action Item Report

Task Assignment:

Team Flying Coyote consists of several positions in regards to the assembly and development of the finalized quadcopter. The positions include the team captain, the mechanical assembly team, the electrical assembly and design team, the software developer, the pilot, and the administrative manager. Lead by Abdul Nasir, the entire group participated in the mechanical fabrication as well as the electrical fabrication of the quadcopter. Trejon Adams led the circuit design, while Chi Nuo Yen developed the majority of the software for autonomous mapping. The administrative manager was Jared Cruz, who designed the final business presentation as well as many of the Action Item Reports. The Coyotecopter was piloted by Trejon Adams for the competition.

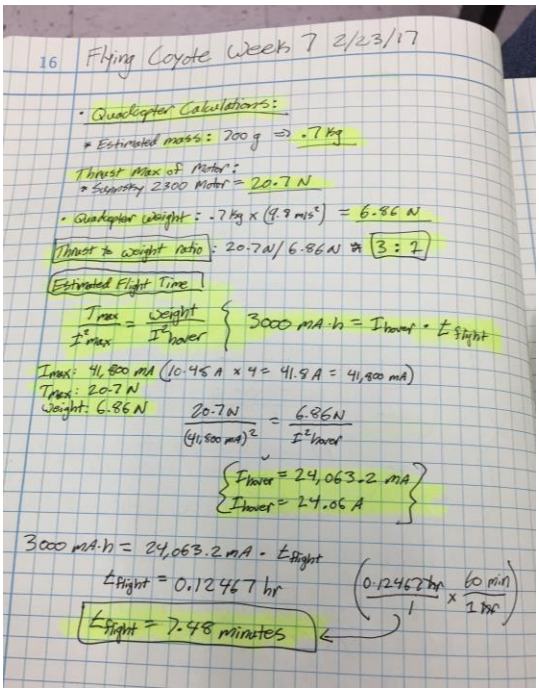
Gantt Chart:

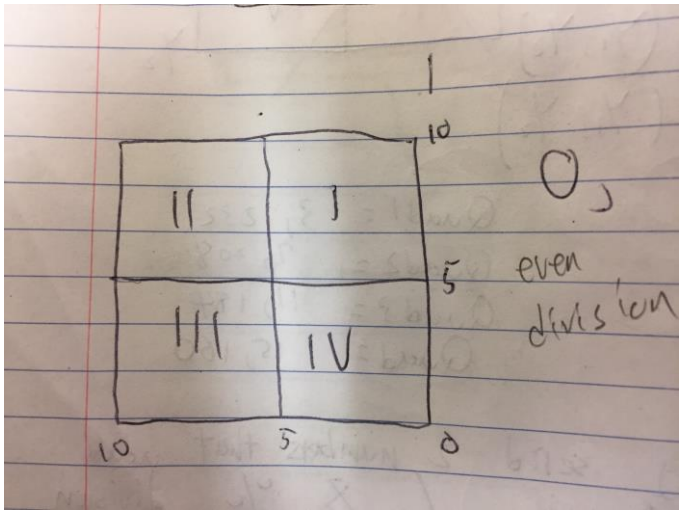


Evaluation

Calculations:

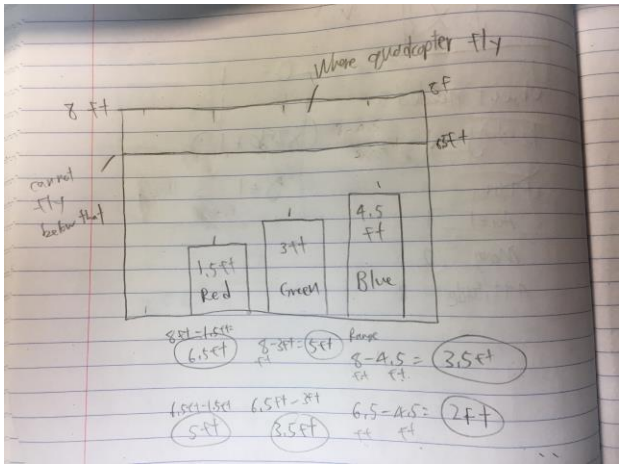
- Quadcopter Mass: 0.7 kg
- Thrust/Weight Ratio: 3:1
- Flight Time ≈ 7.5 minutes





Quadrant Calculation:

- Quadrant 1 is in 0<x<=5 and 5<Y<=10
- Quadrant 2 is in 5<x<=10 and 5<Y<=10
- Quadrant 3 is in 5<x<=10 and 0<Y<=5
- Quadrant 4 is in 0<x<=5 and 0<Y<=5
- For every range, use greater than to the lowest number of the range to avoid the overlaps in the range interval



Color Calculation

- Blue, Color 4, range (2ft<Z<= 3.5ft)
- Green, Color 3, range (3.5ft<Z<=5ft)
- Red, Color 2, range (5ft<Z<=6.5ft)
- White, Color 1, range (<6.5 ft)
- For every range, use greater than to the lowest number of the range to avoid the overlaps in the range interval

Test Plan:

In order to examine the functionality of the quadcopter, three individual component tests and one final integrated test were performed.

The first component was the flight test. After the fabrication of the quadcopter was completed, the pilot will perform a test flight inside a netted cage. The test pilot will control and maintain the thrust to test whether the aircraft can achieve stable flight at 50 percent thrust.

The second test is the XBEE test. The software developer will download the XBEE sample code and modify it. He will assigned specific values to the quadrant and color variables to test whether the LED panel lights up as according to the modified XBEE program. The test will also show whether the XBEE antenna sends signal by plugging one XBEE onto the quadcopter and another onto the LED Panel's arduino board.

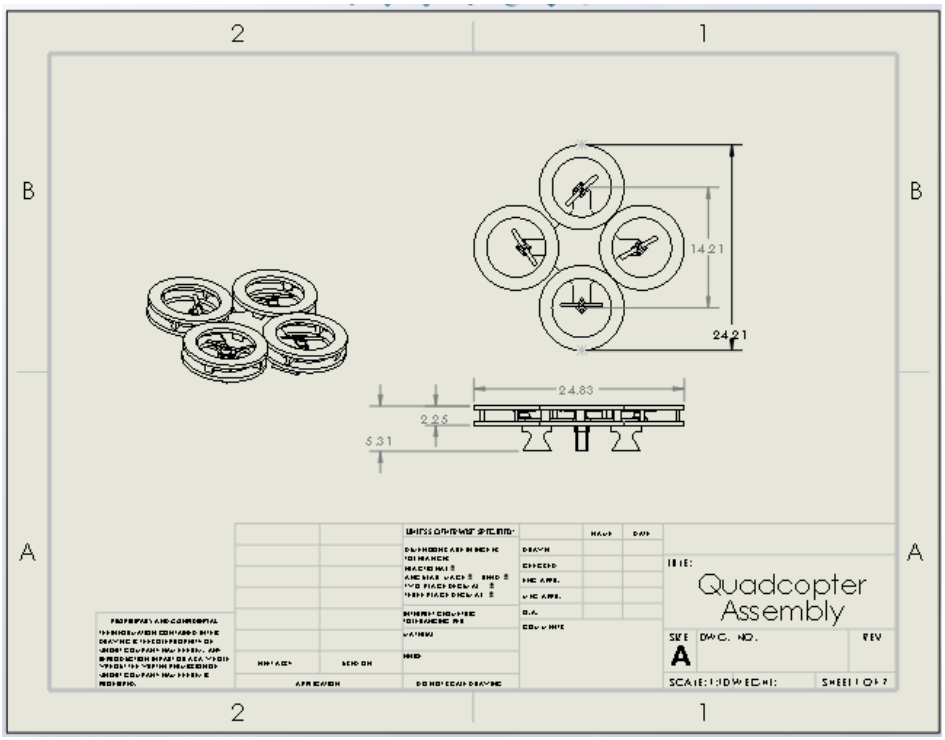
The third test is the ultrasonic sensor mapping test. One team member will hold the quadcopter in hand and orient the quadcopter with the left ultrasonic sensor facing the left wall and the front sensor toward the lab's door. The third ultrasonic sensor will face toward the floor. After the mapping code is uploaded into the Arduino and the second XBEE antenna is mounted on the LED panel's Arduino board, the quadcopter will be moved in such fashion to simulate the quadcopter's flight and provide data on the autonomous mapping capability of the quadcopter. Team Flying Coyote is hoping to see the LED panel light up in different colors and locations according to the quadcopter's height as well as its location within the simulated quadrants.

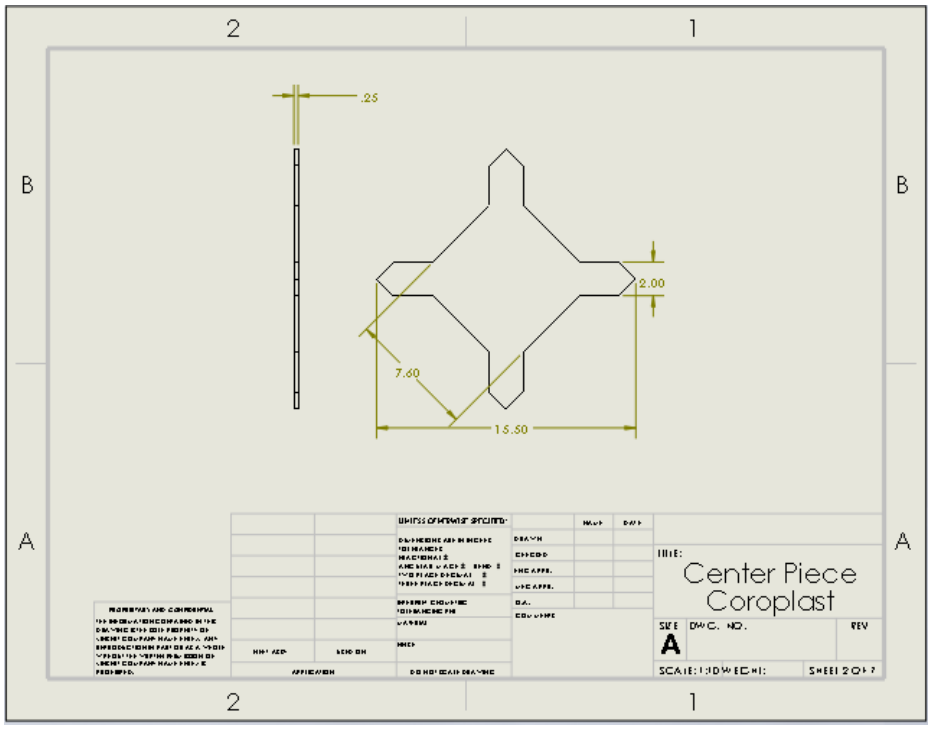
In the integrated test, the test pilot will apply sufficient thrust so the quadcopter can achieve stable flight at the height where the white polycarbonate panels are mounted. In the test, he will attempt to hover over the four divided sections of the netted cage to test the flight and the mapping capability of the quadcopter. His task is to fly into different sections of the cage to see if the LED panel outside the cage will light up. It should light up in various sections one at a time according to the location of the quadcopter. If it happens, the code is working. If it does not happen, then the code has to be revised.

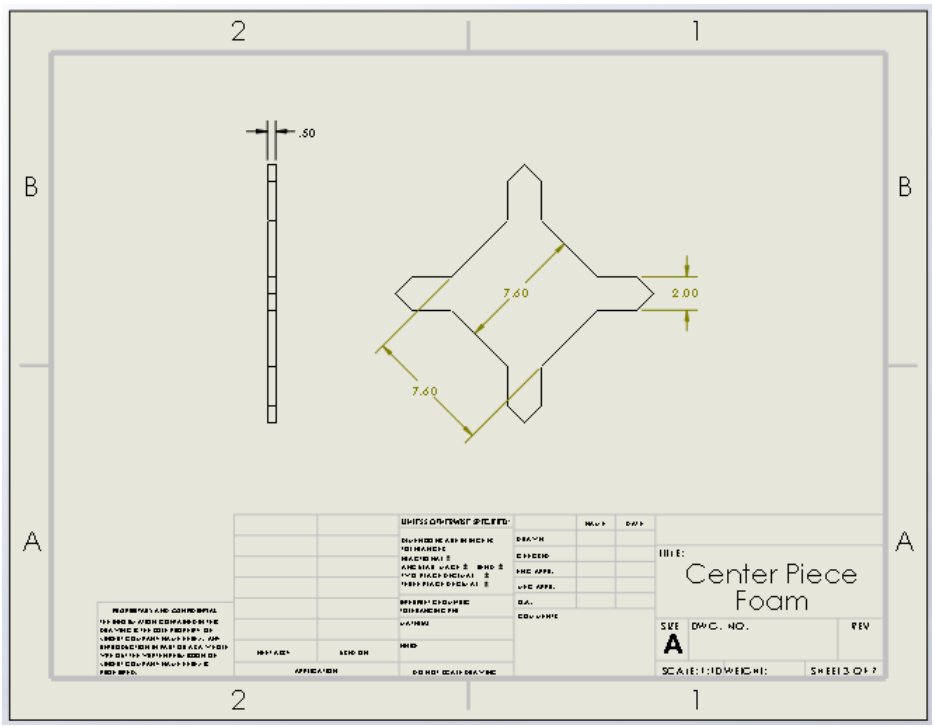
Result & Discussion:

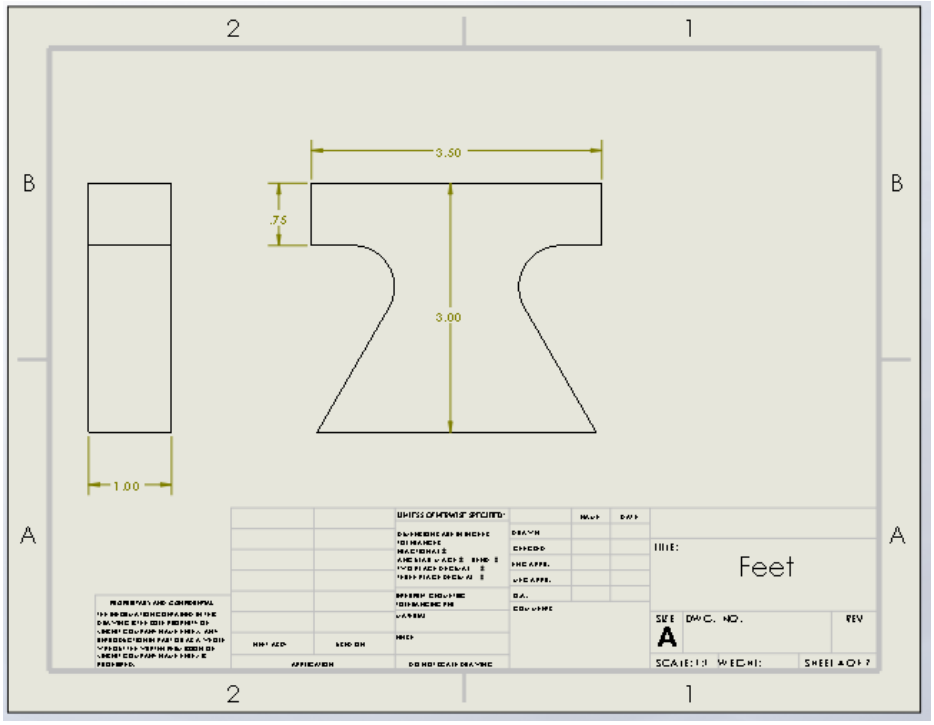
Following the initial test plan, multiple flight tests were performed and the team's flight calculations were clearly reflected in the flight tests. One issue that Flying Coyote did face however was spending too much time on the fabrication aspects of the quadcopter and not leaving sufficient time for the development of the software components of the project. As a result, the team had only the last two weeks of the quarter to finalize the software development for the autonomous mapping features of the quadcopter. In the end, the software that was developed was not completely operable, but it was able to light up several of the LED panels when the quadcopter was idle and not in flight. However, the code would not operate when the quadcopter was actually in flight.

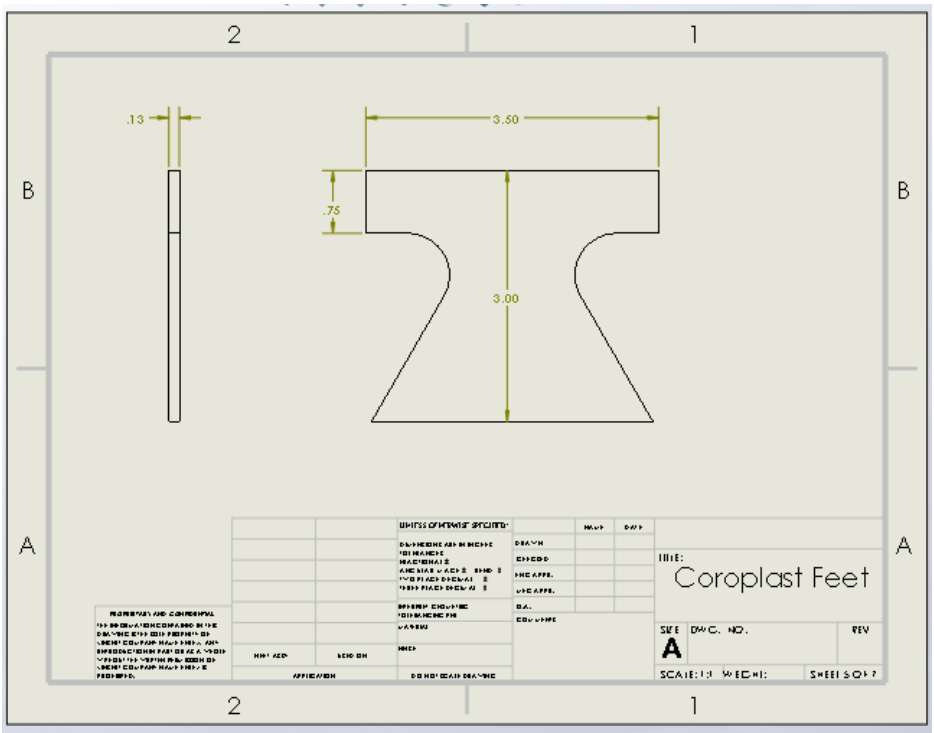
Appendix A: Solidworks Drawings

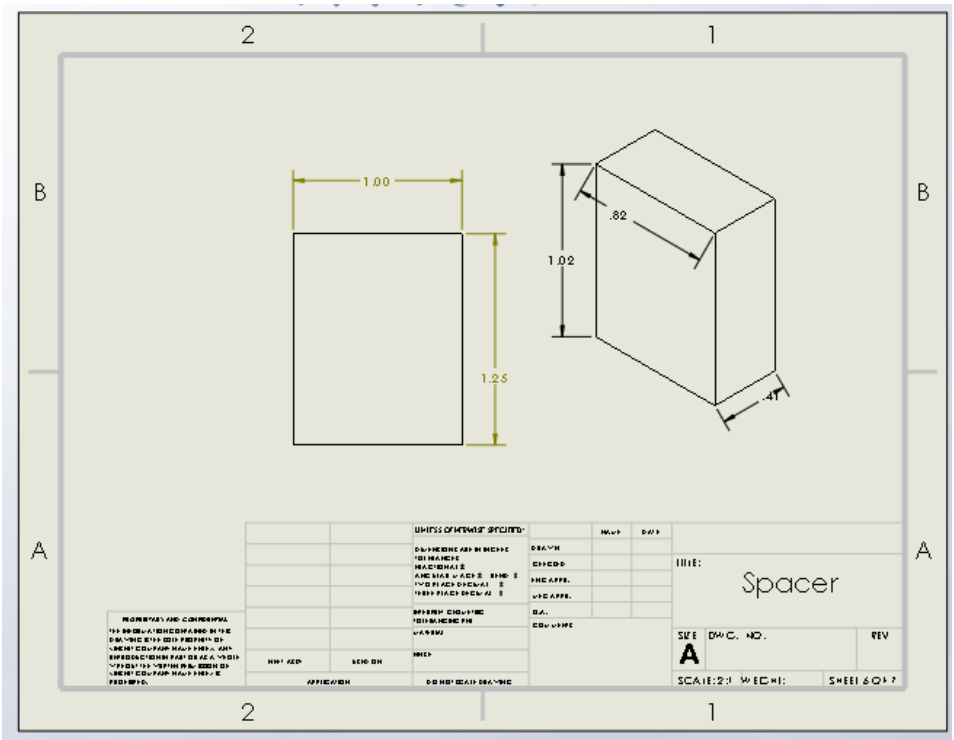


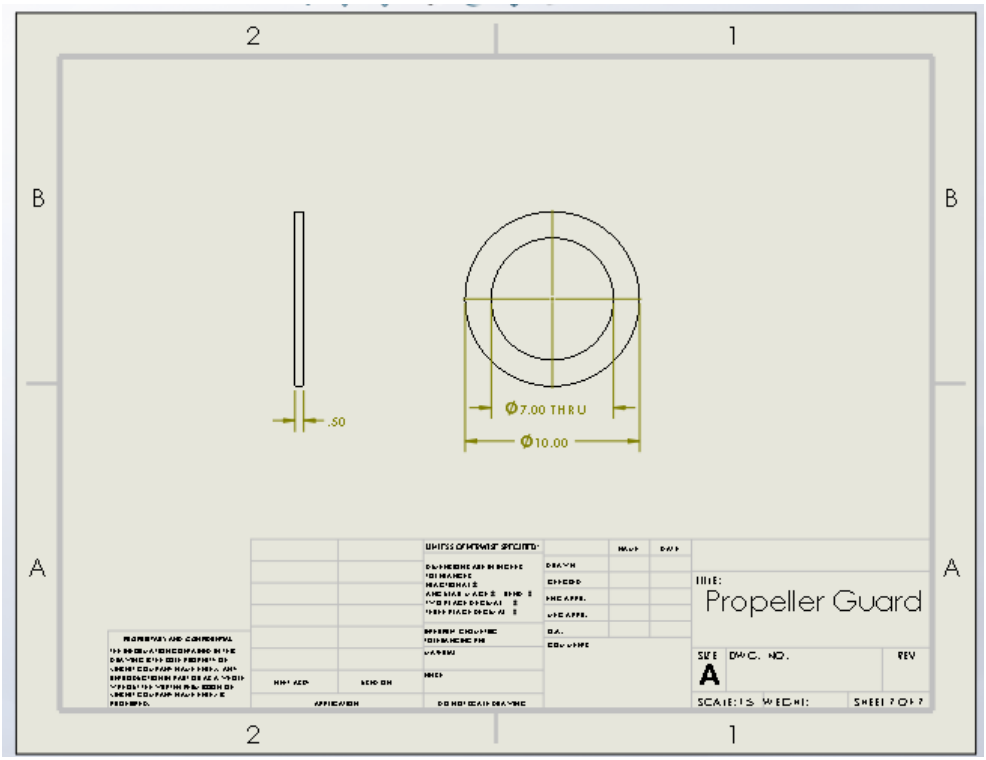



.50
7.60
2.00
7.60
Center Piece
Foam
A


3.50
.75
3.00
1.00
Feet
A


2
1
B
A
.13
3.50
.75
3.00
Coroplast Feet
SIZE
A
SCALE:1:1 WEIGHT:
SHEET 1 OF 1


2
1
B
A
1.00
1.25
1.02
.82
.41
Spacer
SIZE
A
SCALE:2:1 WEIGHT:
SHEET 1 OF 1


2
1
B
A
.50
Ø7.00 THRU
Ø10.00
Propeller Guard
SIZE
A
SCALE: 1:5 WEIGHT:
SHEET 1 OF 1

Appendix B: Bill of Materials

Quantity*	Unit	Company*	Item Description*	Catalog #*	Unit Price*	Estimated Extended Price*
2		Amain Performance Hobbies	HQ Prop 6x3 Carbon Mix Propeller (Black) (2)	HQ-P010506300	$1.79	$3.58
4		Buddy RC	SUNNYSKY X2204S KV2300 II BRUSHLESS MOTOR	SNS-X2204S-KV2300	$12.99	$51.96
4		HobbyPartz	Exceed RC ESC Speed Controller	07E04-Proton-30A	$11.12	$44.48
1		Robot Shop	RadioLink T8EHP-E 2.4G 8CH Transmitter w/ R7EH Receiver	RB-Rlk-03	$49.99	$49.99
1		Century Helicopter Products	NAZA-M Lite Controller	Part# DJINZMLT	$69.00	$69.00
3		GetFPV	3.5mm Gold "Bullet" Connectors (12 Pair)	1516	$2.59	$7.77
1		Hobby King	ZIPPY Flightmax 3000mAh 3S1P 20C Battery	Z30003S-20	$14.88	$14.88
1		Sparkfun	ARDUINO UNO R3	DEV-11021	$24.95	$24.95
1		Sparkfun	XBee Shield	WRL-12847	$14.95	$14.95
1		Sparkfun	XBee Antenna	WRL-08665	$24.95	$24.95
1		Sparkfun	Inertia Measurement Unit	SEN-13284	$24.95	$24.95
2	16" x 16"	UCI LAB	1/4" white coroplast		$1.94	$3.88
2	16" x 16"	UCI LAB	1/16" Polycarbonate		$3.61	$7.22
10	16" x 16"	UCI LAB	Pink Insulating Foam		$0.89	$8.90
5		adafruit	ESC Extension Cable	972	$1.95	$9.75
4		RobotShop	Ultrasonic Sensor	RB-Ite-54	$2.50	$10.00
					Subtotal	**$371.21**
I Need My Order Delivered By This Date:*			**02/02/17**			
					***Tax rate:** 7.75%	$28.77
					TOTAL ORDER PRICE	**$399.98**

Appendix C: Arduino code

```
#include <Ultrasonic.h>      //add needed library
#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h>

Ultrasonic ultrasonic1(12,13); //ultrasonic 1 downward output 12, input 13
Ultrasonic ultrasonic2(8,7); //ultrasonic 2 left output 8, input 7
Ultrasonic ultrasonic3(4,2); //ultrasonic 3 front output 4, input 2

LSM9DS1 imu; // Name IMU object "imu"

float roll;
float pitch;
float heading;
float AX;
float AY;
float AZ;
float MX;
float MY;
float MZ;

float ultra1, ultra2, ultra3, ult1, ult2, ult3;

#define LSM9DS1_M  0x1E
#define LSM9DS1_AG  0x6B
#define PRINT_CALCULATED // This line is active - the more-useful calculated values
will print - see below
// #define PRINT_RAW // This line is not active (commented out)
#define PRINT_SPEED 250
#define DECLINATION -12 // Irvine, CA declination

int quad, color;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);//send serial monitor reading
  imu.settings.device.commInterface = IMU_MODE_I2C;
  imu.settings.device.mAddress = LSM9DS1_M;
```

```
  imu.settings.device.agAddress = LSM9DS1_AG;
  if (!imu.begin()) // This line means "If the IMU does NOT(!) begin, print the following
message..."
  {
        Serial.println("Failed to communicate with LSM9DS1.");
        Serial.println("Double-check wiring.");
        Serial.println("Default settings in this sketch will " \
              "work for an out of the box LSM9DS1 " \
              "Breakout, but may need to be modified " \
              "if the board jumpers are.");
        while (1); // wait forever
  }
}

void loop()
{
  Serial.print(ultrasonic1.Ranging(CM)); //monitor 3 sensors input in centimeters
  Serial.print(" cm");
  Serial.print(ultrasonic2.Ranging(CM));
  Serial.print(" cm");
  Serial.print(ultrasonic3.Ranging(CM));
  Serial.println(" cm");

  printGyro();  // Print "G: gx, gy, gz"
  printAccel(); // Print "A: ax, ay, az"
  printMag();   // Print "M: mx, my, mz"

  // Print the heading and orientation for fun!
  // Call print attitude. The LSM9DS1's magnetometer x and y
  // axes are opposite to the accelerometer, so my and mx are
  // substituted for each other.
  printAttitude(imu.ax, imu.ay, imu.az, -imu.my, -imu.mx, imu.mz);
  Serial.println();

  ult1=ultrasonic1.Ranging(CM); //convert serial read to variables
  ult2=ultrasonic2.Ranging(CM);
  ult3=ultrasonic3.Ranging(CM);

  ultra1 = 0;  //set ultra1,ultra2, ultra3 to 0
  ultra2 = 0;
```

```
  ultra3 = 0;

  ultra1 = ult1 * .032084; //cm to ft conversion
  ultra2 = ult2 * .032084;
  ultra3 = ult3 * .032084;

  if (ultra2 >0 and ultra2 <=5 and ultra3 >5 and ultra3 <=10) { //boolean for ranges of
quadrant in ft
        quad = 1;
  }
  else if (ultra2 >5 and ultra2 <=10 and ultra3 >5 and ultra3 <=10) { //use sensors
greater than initial range to prevent overlap
        quad = 2;
  }
  else if (ultra2 >5 and ultra2 <=10 and ultra3 >0 and ultra3 <=5) {
        quad = 3;
  }
  else if (ultra2 >0 and ultra2 <=5 and ultra3 >0 and ultra3 <=5) {
        quad = 4;
  }

  if (ultra1 >2 and ultra1 <= 3.5) {
        color = 4;//blue
  }
  else if (ultra1 >3.5 and ultra1 <= 5) {
        color = 3;//green
  }
  else if (ultra1 >5 and ultra1 <= 6.5) {
        color = 2;//red
  }
  else if (ultra1 >6.5) {
        color = 1;//white
  }

  /* ***** The following 6 lines of code are IMPORTANT and must be executed in this
EXACT order ***** */
  char a = 'E'; // Define character E to be used as indicator - see next line
  Serial.write(a); //FIRST: Send 'E' which indicates a new set of data is being transmitted
  Serial.write(quad/256); // returns INTEGER of division (e.g. 7/3 = 2)
```

```
  Serial.write(quad%256); // returns the REMAINDER of quadrant number divided by
256 (e.g. 7%3 = 1)
  // THIRD: Send the color number as two bytes
  Serial.write(color/256);
  Serial.write(color%256);
  delay(10);

}

void printGyro()
{
  // To read from the gyroscope, you must first call the
  // readGyro() function. When this exits, it'll update the
  // gx, gy, and gz variables with the most current data.
  imu.readGyro();

  // Now we can use the gx, gy, and gz variables as we please.
  // Either print them as raw ADC values, or calculated in DPS.
  Serial.print("G: ");
#ifdef PRINT_CALCULATED //library for calcGyro
  // If you want to print calculated values, you can use the
  // calcGyro helper function to convert a raw ADC value to
  // DPS. Give the function the value that you want to convert.
  Serial.print(imu.calcGyro(imu.gx), 2);
  Serial.print(", ");
  Serial.print(imu.calcGyro(imu.gy), 2);
  Serial.print(", ");
  Serial.print(imu.calcGyro(imu.gz), 2);
  Serial.println(" deg/s");
#elif defined PRINT_RAW
  Serial.print(imu.gx);
  Serial.print(", ");
  Serial.print(imu.gy);
  Serial.print(", ");
  Serial.println(imu.gz);
#endif
}

void printAccel()
{
```

```
  // To read from the accelerometer, you must first call the
  // readAccel() function. When this exits, it'll update the
  // ax, ay, and az variables with the most current data.
  imu.readAccel();

  // Now we can use the ax, ay, and az variables as we please.
  // Either print them as raw ADC values, or calculated in g's.
  Serial.print("A: ");
#ifdef PRINT_CALCULATED//to use calcAccel function, call the library
  // If you want to print calculated values, you can use the
  // calcAccel helper function to convert a raw ADC value to
  // g's. Give the function the value that you want to convert.
  Serial.print(imu.calcAccel(imu.ax), 2);
  Serial.print(", ");
  Serial.print(imu.calcAccel(imu.ay), 2);
  Serial.print(", ");
  Serial.print(imu.calcAccel(imu.az), 2);
  Serial.println(" g");
#elif defined PRINT_RAW
  Serial.print(imu.ax);
  Serial.print(", ");
  Serial.print(imu.ay);
  Serial.print(", ");
  Serial.println(imu.az);
#endif

}

void printMag()
{
  // To read from the magnetometer, you must first call the
  // readMag() function. When this exits, it'll update the
  // mx, my, and mz variables with the most current data.
  imu.readMag();

  // Now we can use the mx, my, and mz variables as we please.
  // Either print them as raw ADC values, or calculated in Gauss.
  Serial.print("M: ");
#ifdef PRINT_CALCULATED //call the library to use calcMag
  // If you want to print calculated values, you can use the
```

```
  // calcMag helper function to convert a raw ADC value to
  // Gauss. Give the function the value that you want to convert.
  Serial.print(imu.calcMag(imu.mx), 2);
  Serial.print(", ");
  Serial.print(imu.calcMag(imu.my), 2);
  Serial.print(", ");
  Serial.print(imu.calcMag(imu.mz), 2);
  Serial.println(" gauss");
#elif defined PRINT_RAW
  Serial.print(imu.mx);
  Serial.print(", ");
  Serial.print(imu.my);
  Serial.print(", ");
  Serial.println(imu.mz);
#endif
}

// Calculate pitch, roll, and heading.
// Pitch/roll calculations take from this app note:
// http://cache.freescale.com/files/sensors/doc/app_note/AN3461.pdf?fpsp=1
// Heading calculations taken from this app note:
// http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-
documents/Defense_Brochures-documents/Magnetic__Literature_Application_notes-
documents/AN203_Compass_Heading_Using_Magnetometers.pdf
void printAttitude(float ax, float ay, float az, float mx, float my, float mz) {

  float roll = atan2(ay, az); //find degree of roll
  float pitch = atan2(-ax, sqrt(ay * ay + az * az)); //degree of pitch

  float heading;
  if (my == 0) {
        heading = (mx < 0) ? 180.0 : 0;
  }
  else {
        heading = atan2(mx, my); //degree of heading
  }

  heading -= DECLINATION * PI / 180; //heading - declimatnio in radians

  int data;
```

```
  if (heading < -0.261667) { //-15 degree in radians
        Serial.println("Quad not aligned");
  }
  else if (heading > 0.261667) { //15 degree in radians
        Serial.println("Quad not aligned");
  }
  else {
        Serial.println("Quad aligned");
  }

  // Convert everything from radians to degrees:
  heading *= 180.0 / PI;
  pitch *= 180.0 / PI;
  roll  *= 180.0 / PI;

  Serial.print("Pitch, Roll: ");
  Serial.print(pitch, 2);
  Serial.print(", ");
  Serial.println(roll, 2);
  Serial.print("Heading: ");
  Serial.println(heading, 2);
}
```

Appendix D: References

Reference Used:

- ENGR 7B Design Report Template W17 - eee.uci.edu
- Team WeeSnaw's Design Report docs.google.com
- Team Flying Coyote Preliminary Design Presentation
- Digikey.com - Free Online Schematic Drawing Tool
- draw.io - flowchart diagram tool