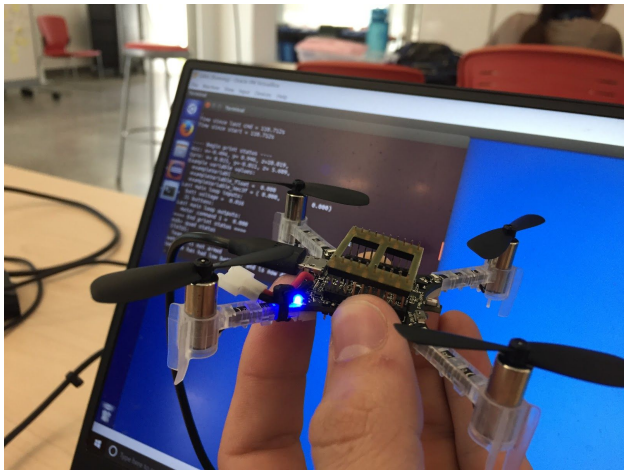
ME136 Lab 3

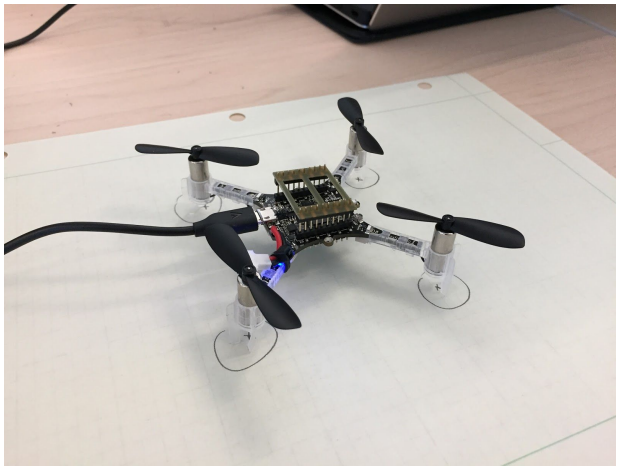
Group 6

Trey Fortmuller
Joey Kroeger
David Dominguez Hooper

1. In the this lab we implemented various estimators. We tested the estimators on each axis that utilize only the rate gyro with the quadcopter in our hands, simply rotating the quadcopter on each axis to ensure our estimators had the correct signs and scale. We used the table as a reference for a plane normal to the force of gravity.

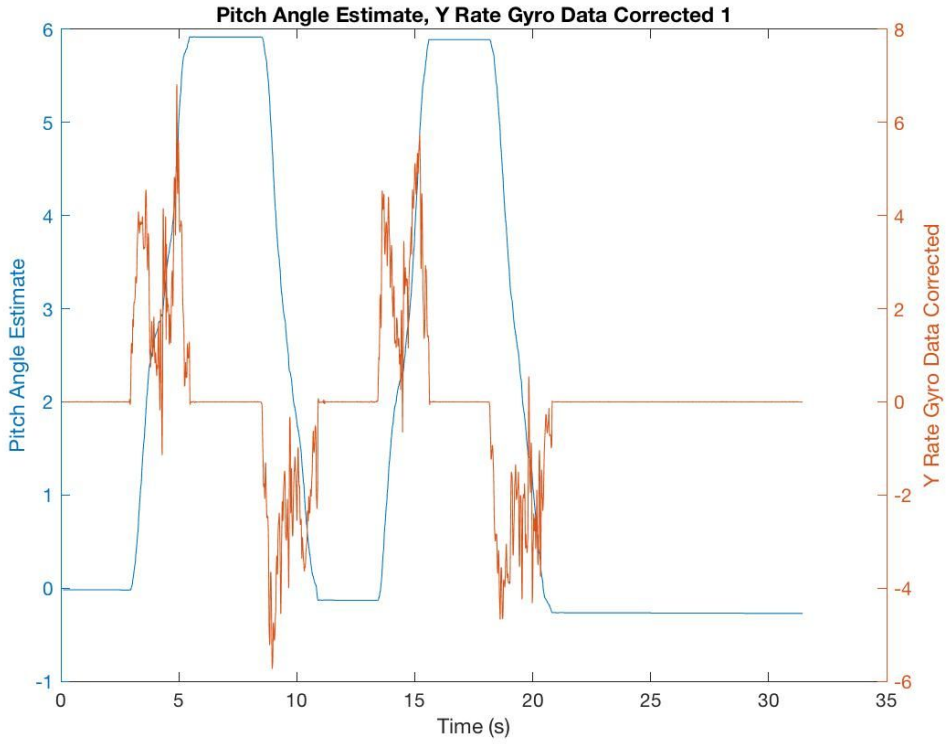
For part 4, after the full accelerometer/gyro estimator had been implemented, we placed the quadcopter in a known position on a secured piece of paper so that we could return it to the original orientation after manipulating it to see how the estimation would vary. We collected data about the sensor data and our estimators' output throughout the lab using `quad status` and the log files.

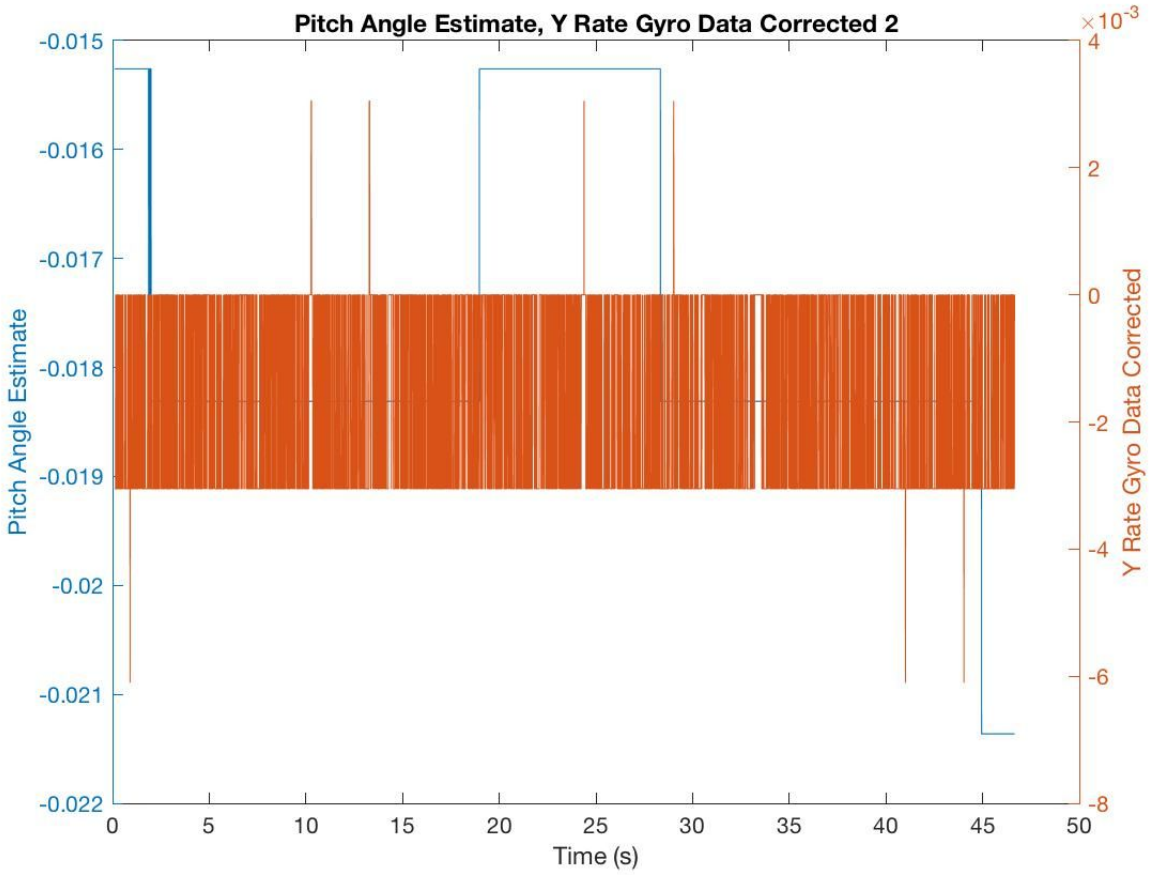


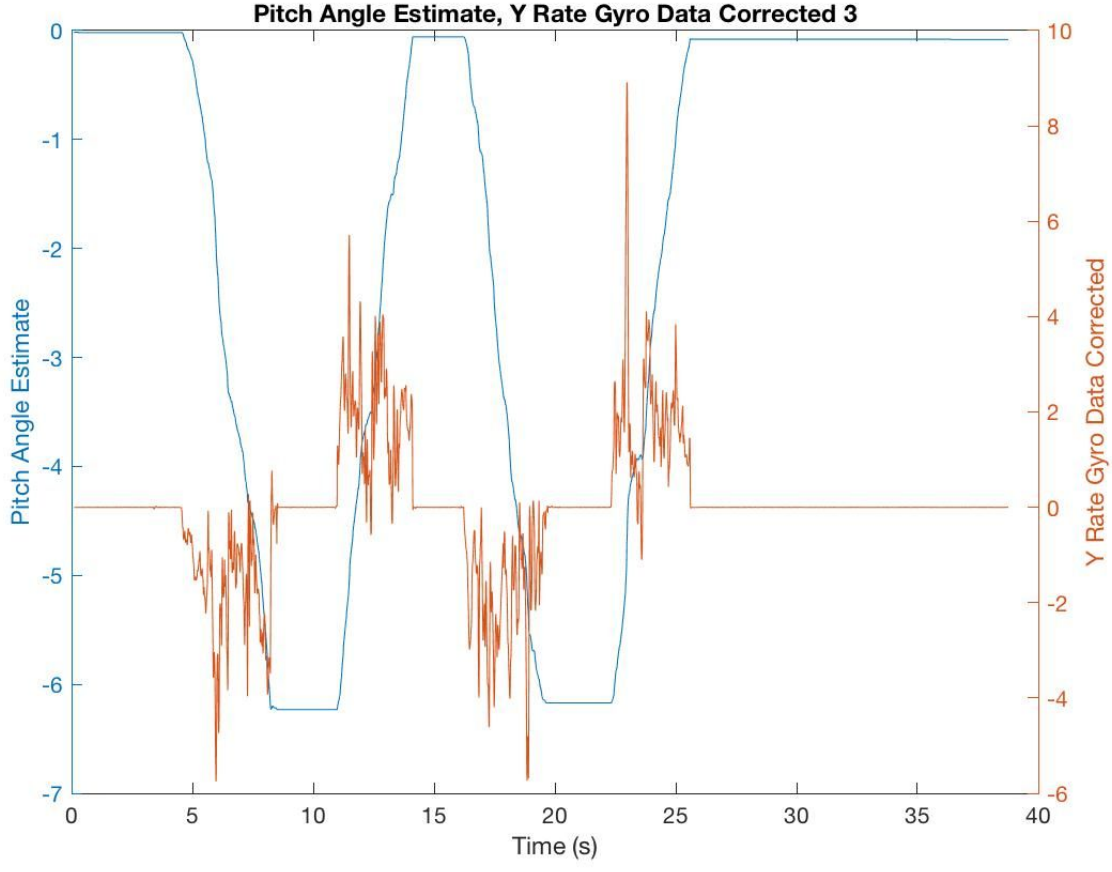


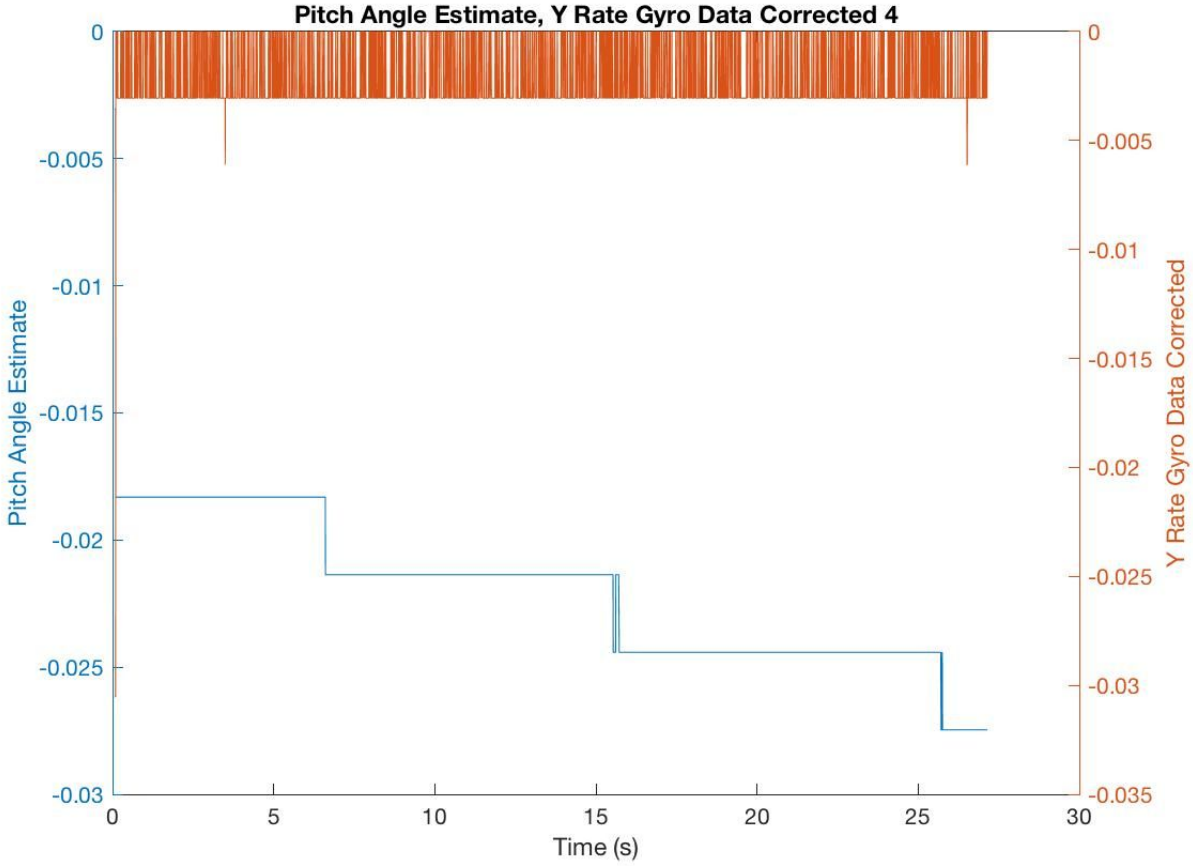
2.

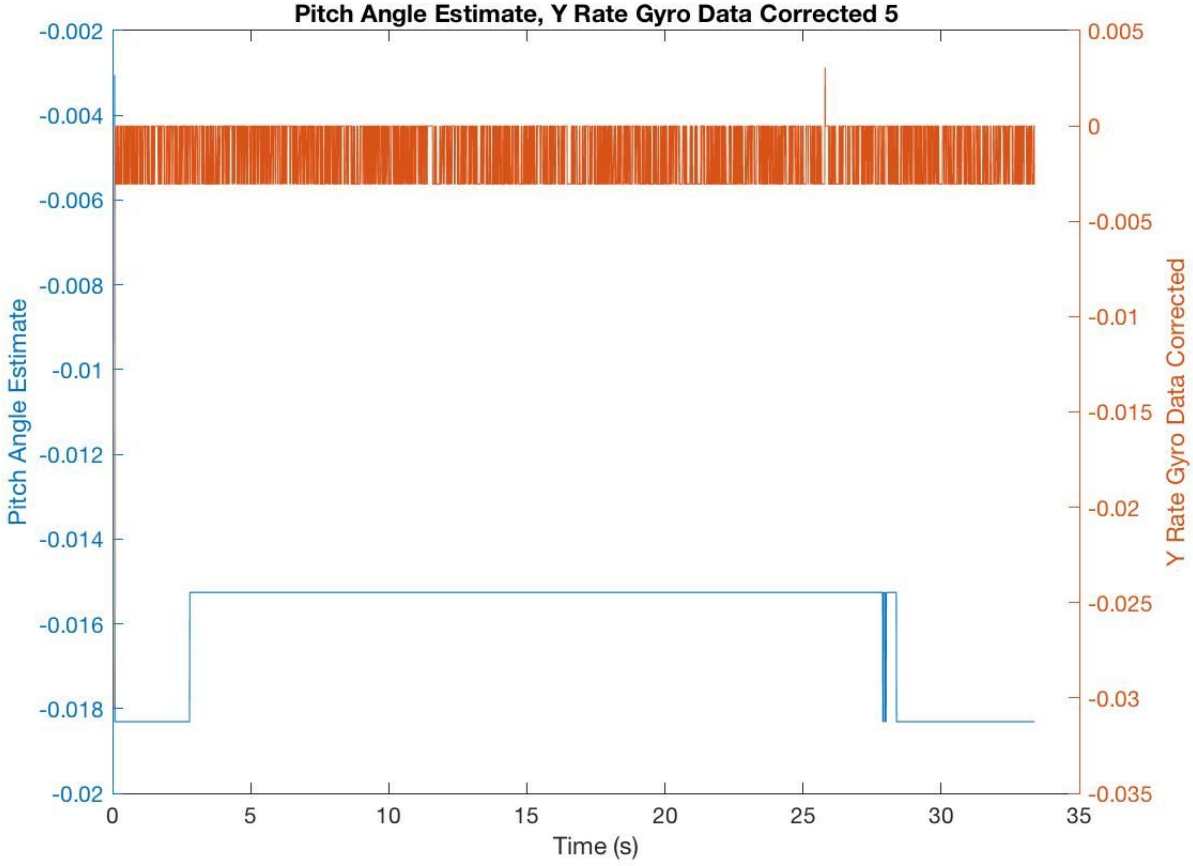
a) Pitch angle estimate based only on rate gyro, we move the vehicle through various rotations to test the performance of the estimator.




Pitch Angle Estimate, Y Rate Gyro Data Corrected 2
Pitch Angle Estimate
Y Rate Gyro Data Corrected
Time (s)
×10^-3
-0.015
-0.016
-0.017
-0.018
-0.019
-0.02
-0.021
-0.022
4
2
0
-2
-4
-6
-8
0
5
10
15
20
25
30
35
40
45
50


Pitch Angle Estimate, Y Rate Gyro Data Corrected 3
Pitch Angle Estimate
Y Rate Gyro Data Corrected
Time (s)
0
-1
-2
-3
-4
-5
-6
-7
10
8
6
4
2
0
-2
-4
-6
0
5
10
15
20
25
30
35
40


Pitch Angle Estimate, Y Rate Gyro Data Corrected 4
Pitch Angle Estimate
Y Rate Gyro Data Corrected
Time (s)
0
-0.005
-0.01
-0.015
-0.02
-0.025
-0.03
-0.035
0
5
10
15
20
25
30

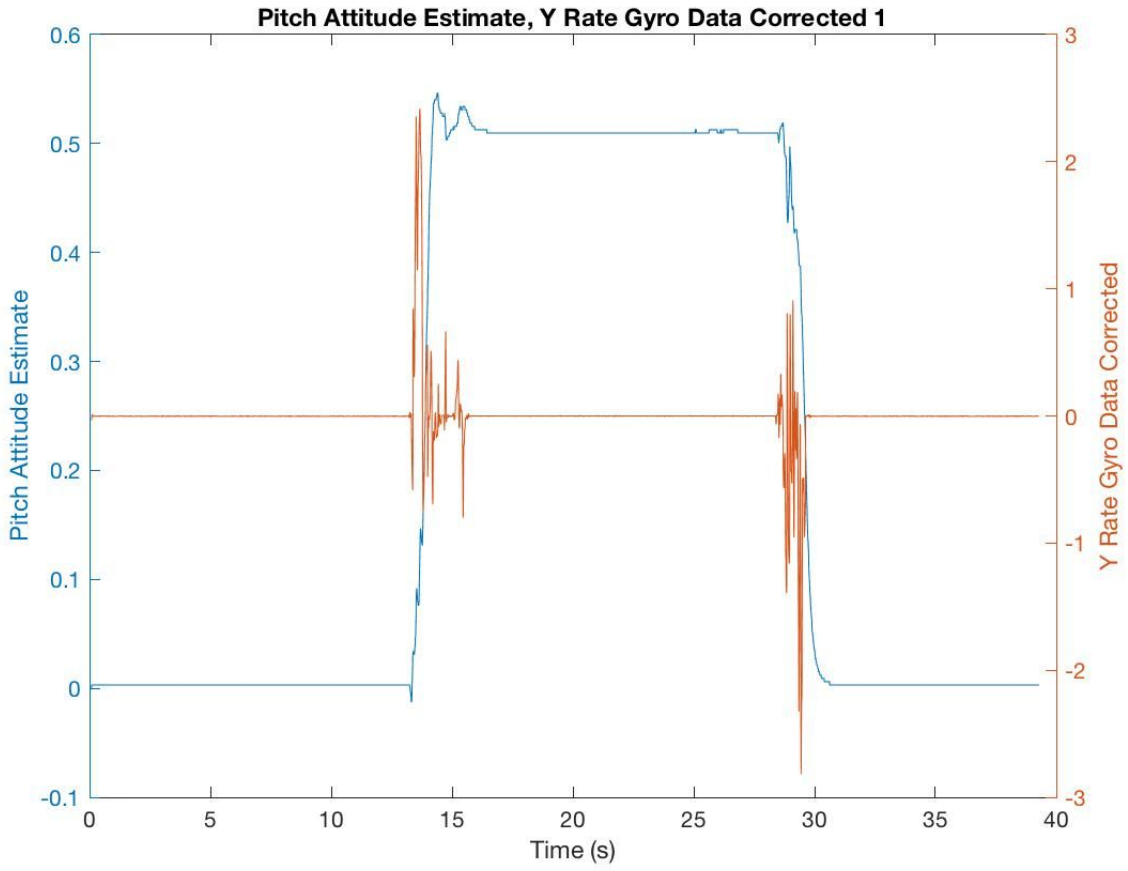

Pitch Angle Estimate, Y Rate Gyro Data Corrected 5
Pitch Angle Estimate
Y Rate Gyro Data Corrected
Time (s)
-0.002
-0.004
-0.006
-0.008
-0.01
-0.012
-0.014
-0.016
-0.018
-0.02
0.005
0
-0.005
-0.01
-0.015
-0.02
-0.025
-0.03
-0.035
0
5
10
15
20
25
30
35

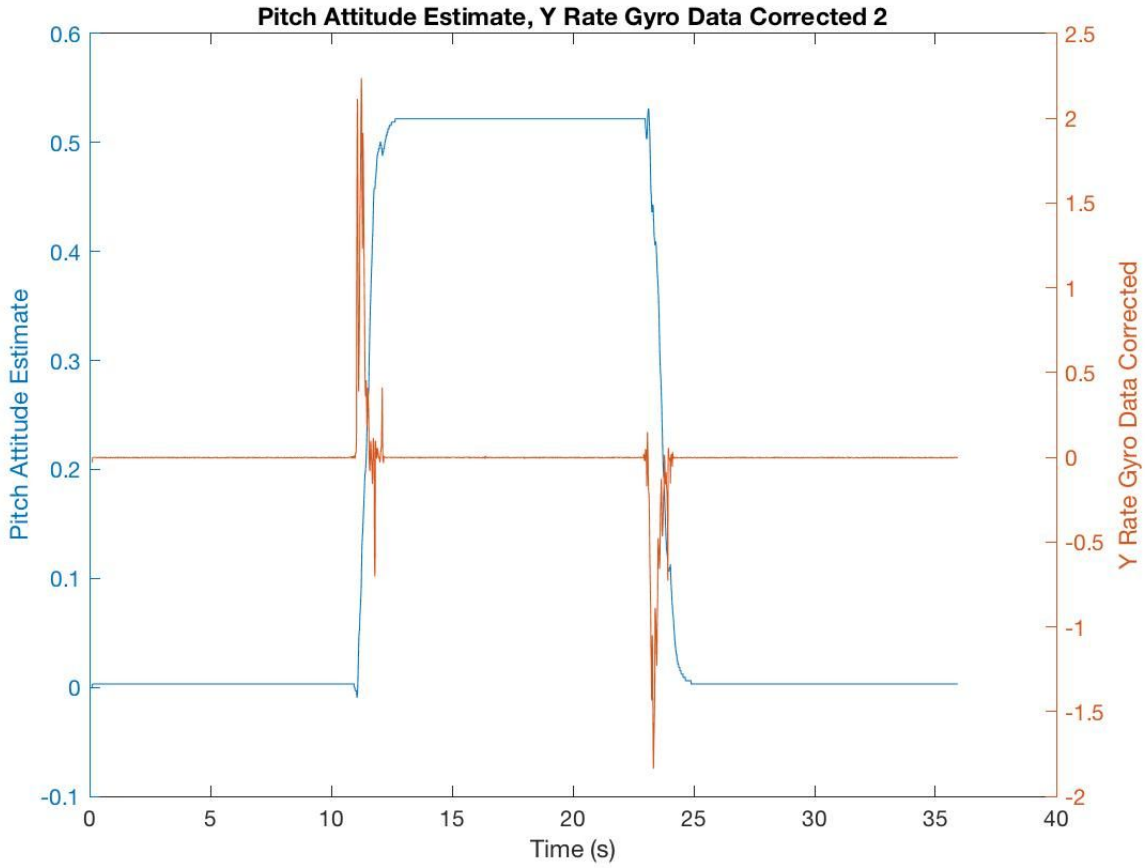
b) Given that the quadcopter is an unstable system (in that it will not naturally return to equilibrium when it is not controlled appropriately), and that our ability to control the quadcopter depends on our estimator being reliable, any small error in the estimator will compound quickly and result in a crash. We consider an “excessive error” to be 1 degree (=0.0174rad) trust this estimator for about 5 to 10 seconds, based on the graphs above.

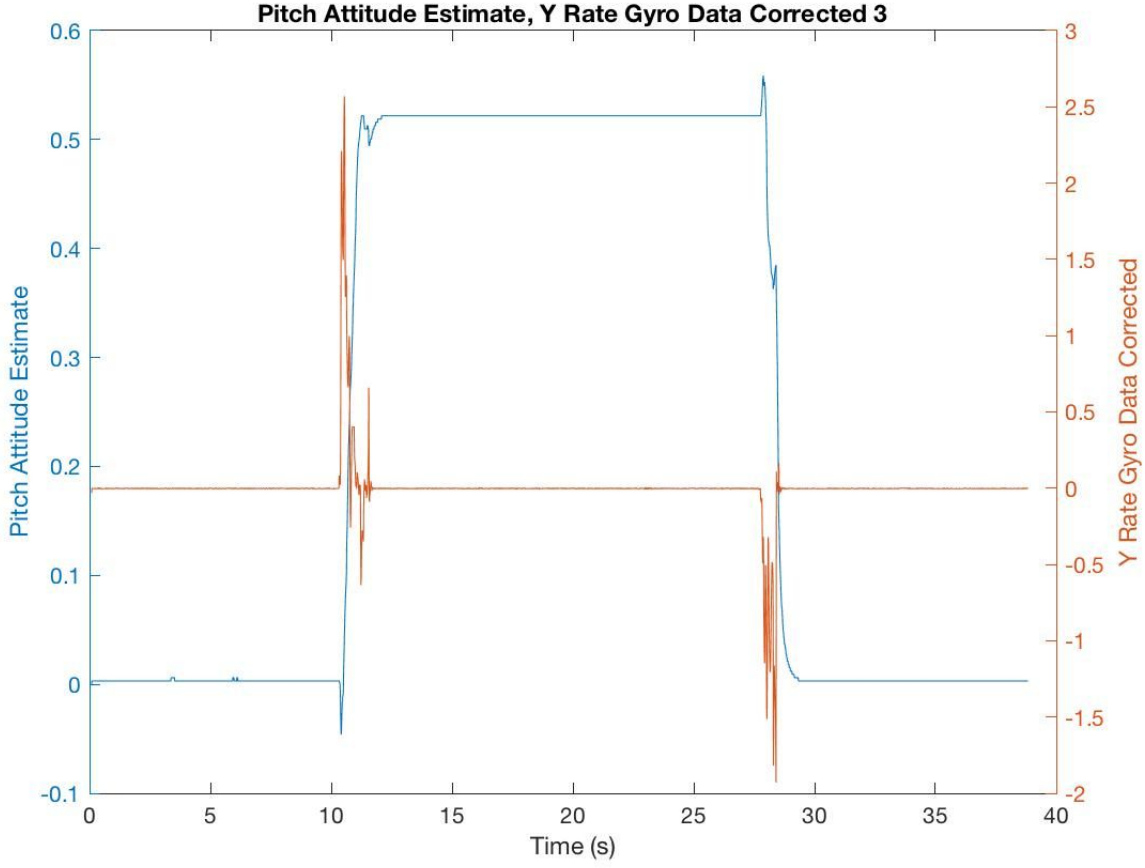
3.

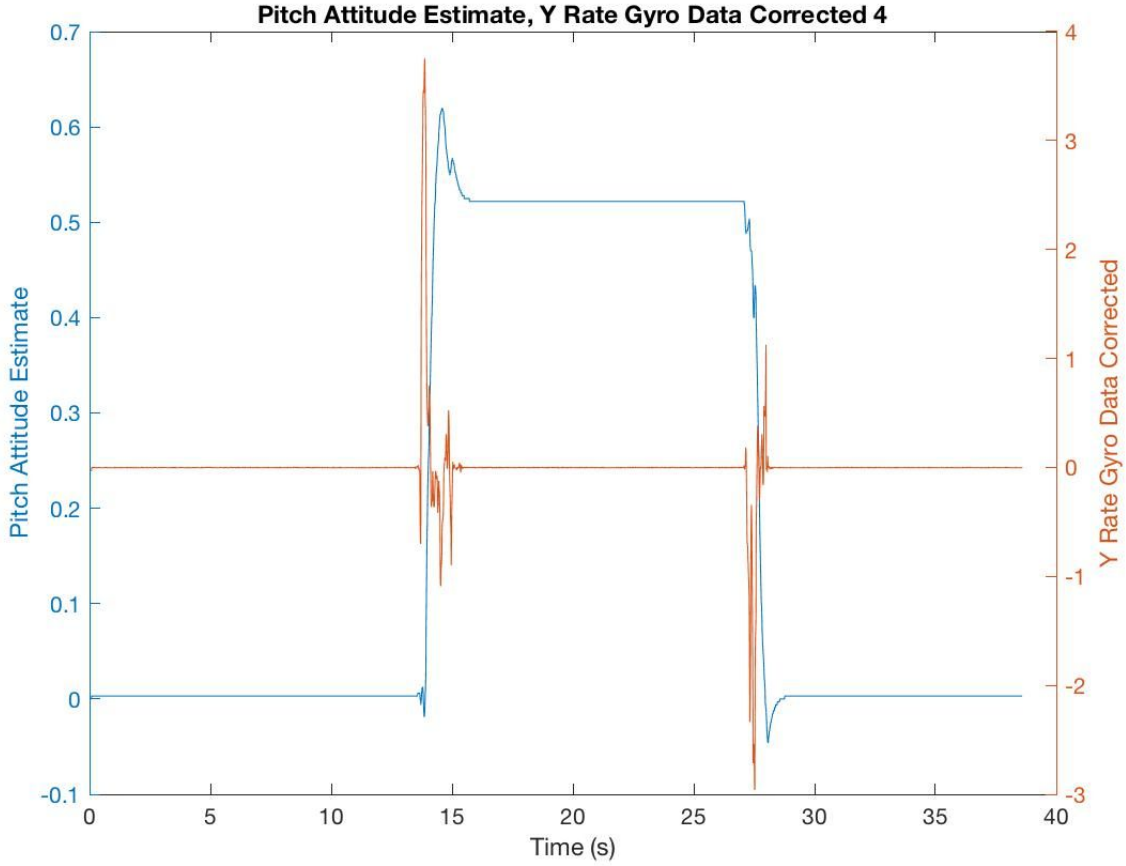
a) We didn’t have time during the lab to try out various choices of rho, we kept the given choices of rho =0.01 for our tests.

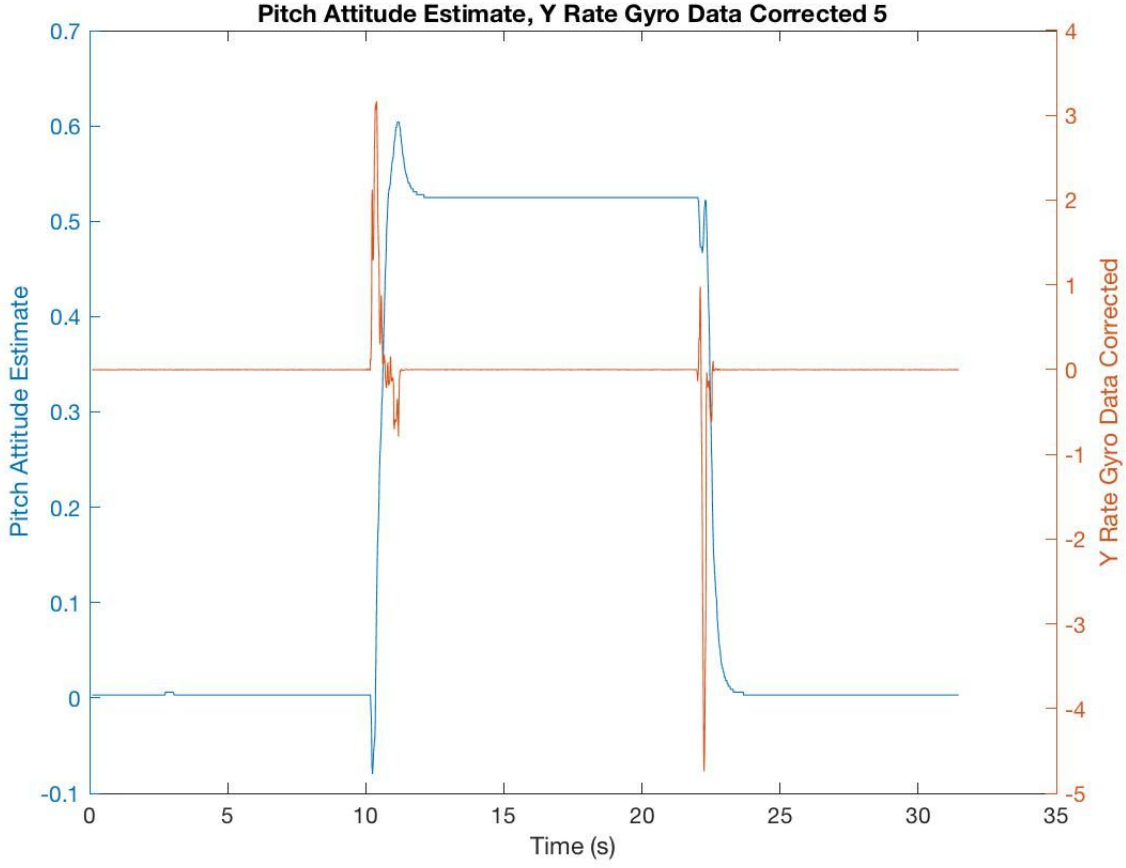
b) This estimator is much more robust to disturbances on the pitch and roll axis where the accelerometer can be incorporated. The drift that we see over time is much less noticeable by incorporating an actual orientation measurement in the estimator rather than naive integration.



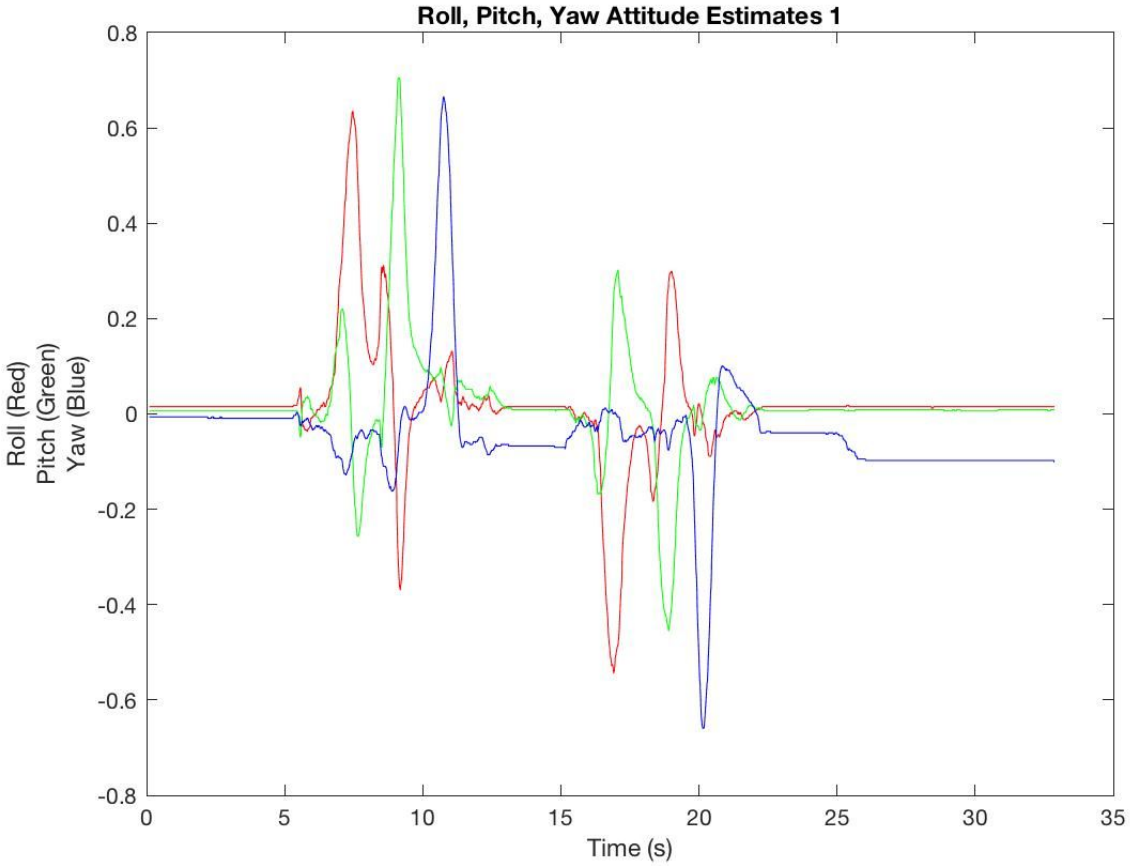

Pitch Attitude Estimate, Y Rate Gyro Data Corrected 2
Pitch Attitude Estimate
Y Rate Gyro Data Corrected
Time (s)
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
35
40

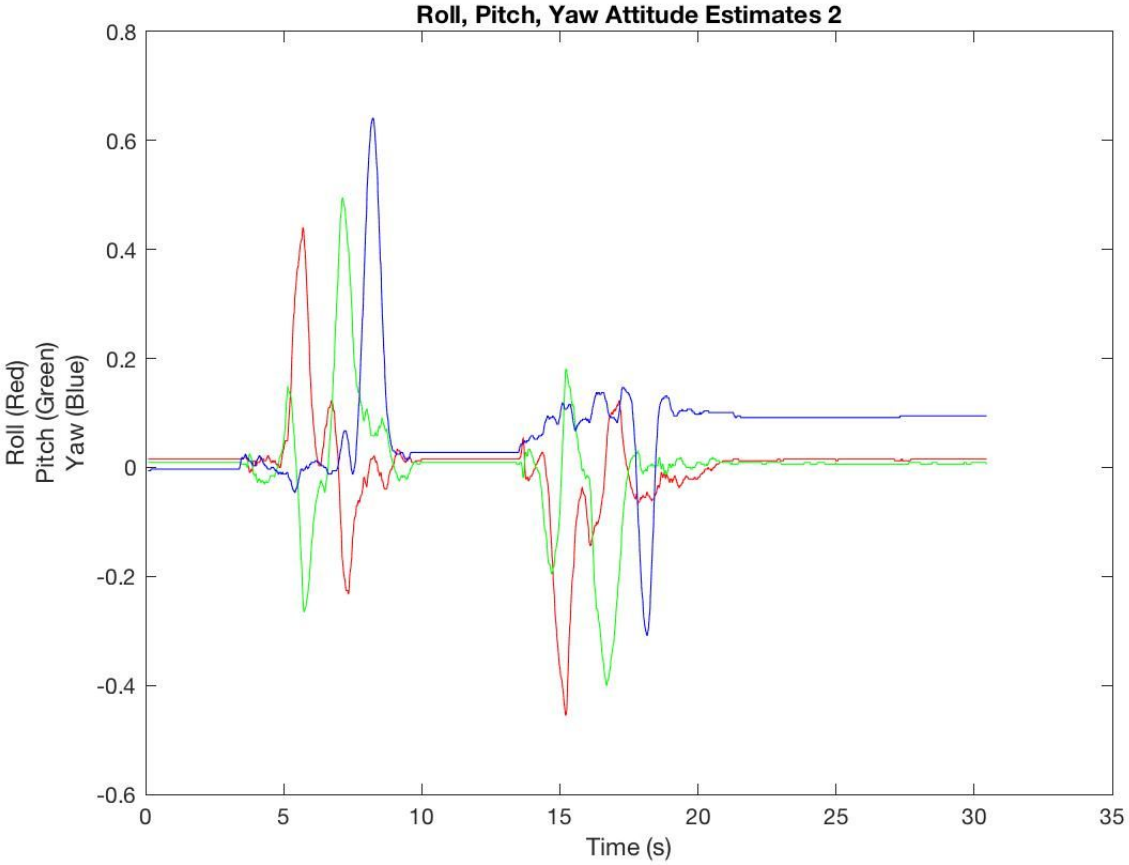

Pitch Attitude Estimate, Y Rate Gyro Data Corrected 3
Pitch Attitude Estimate
Y Rate Gyro Data Corrected
Time (s)
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
5
10
15
20
25
30
35
40

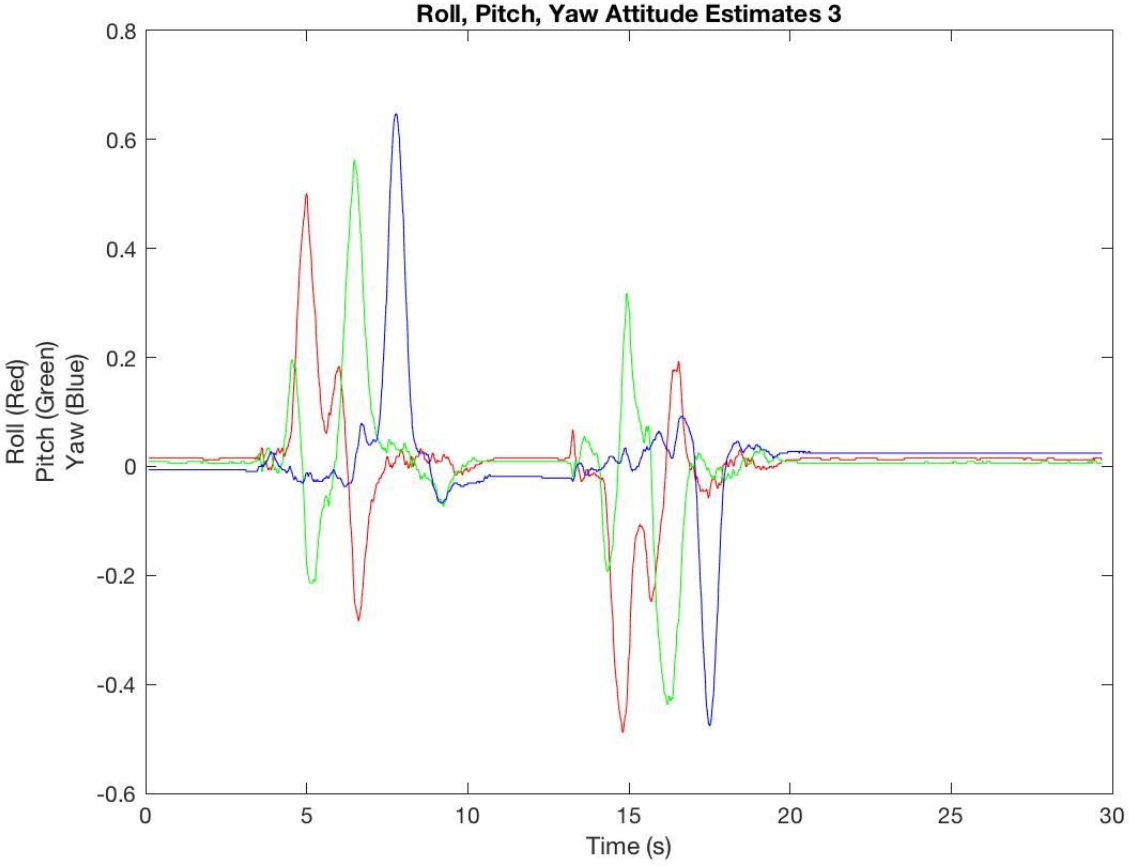

Pitch Attitude Estimate, Y Rate Gyro Data Corrected 4
Pitch Attitude Estimate
Y Rate Gyro Data Corrected
Time (s)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
4
3
2
1
0
-1
-2
-3
0
5
10
15
20
25
30
35
40

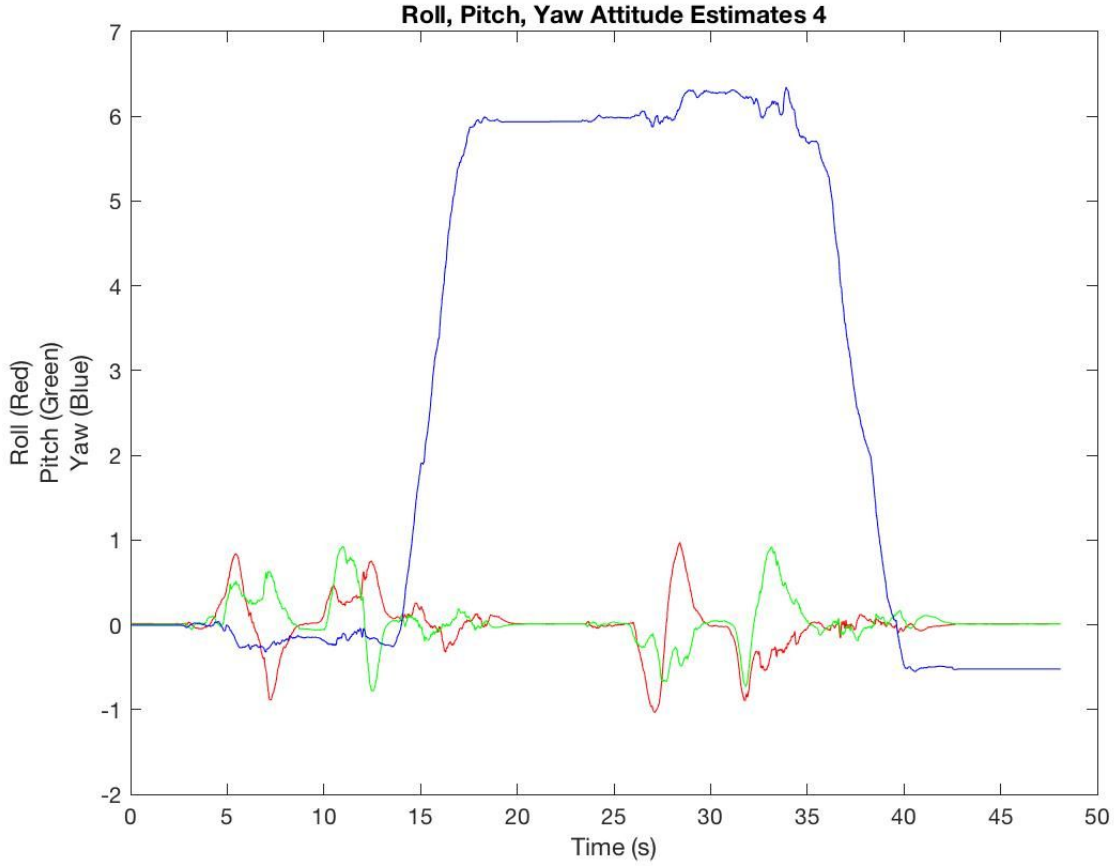

Pitch Attitude Estimate, Y Rate Gyro Data Corrected 5
Pitch Attitude Estimate
Y Rate Gyro Data Corrected
Time (s)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
4
3
2
1
0
-1
-2
-3
-4
-5
0
5
10
15
20
25
30
35

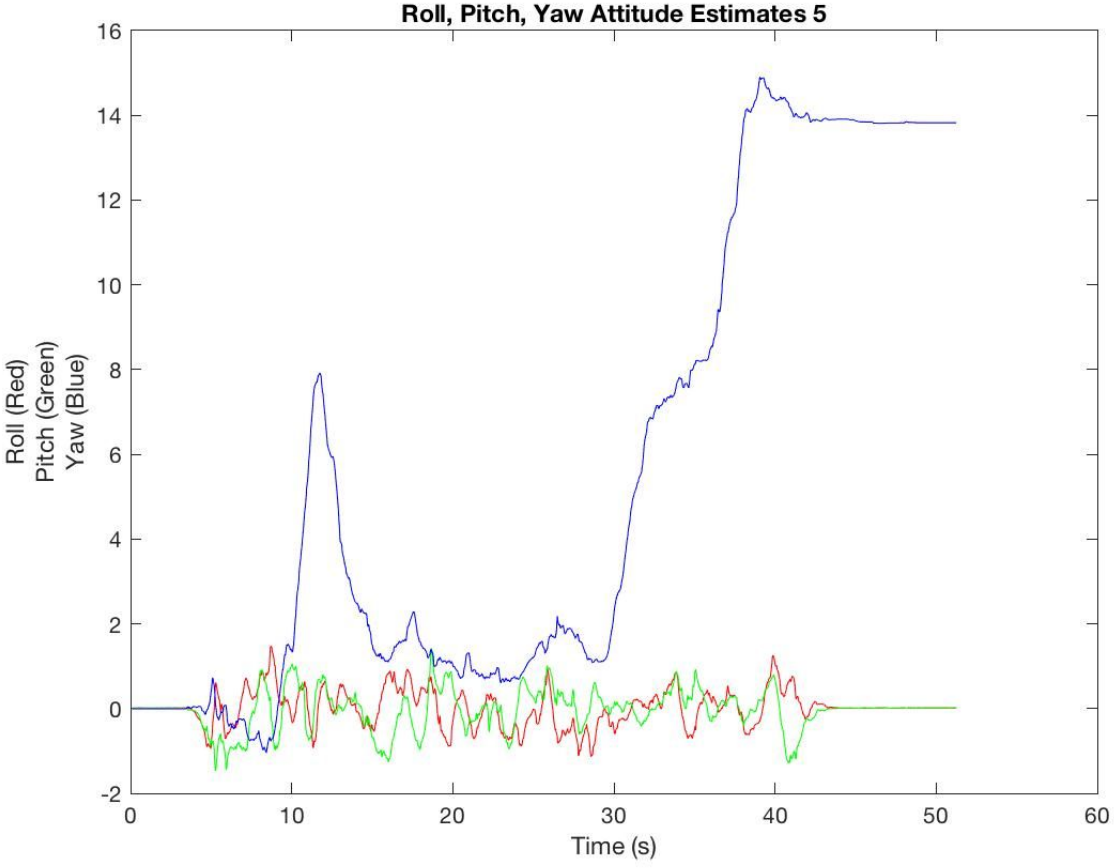
4. Listed below are the 5 trials of placing the quadcopter with its motors on a marked sheet of paper.




Roll, Pitch, Yaw Attitude Estimates 2
Roll (Red)
Pitch (Green)
Yaw (Blue)
Time (s)
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
5
10
15
20
25
30
35


Roll, Pitch, Yaw Attitude Estimates 3
Roll (Red)
Pitch (Green)
Yaw (Blue)
Time (s)
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
5
10
15
20
25
30


Roll, Pitch, Yaw Attitude Estimates 4
Roll (Red)
Pitch (Green)
Yaw (Blue)
Time (s)
7
6
5
4
3
2
1
0
-1
-2
0
5
10
15
20
25
30
35
40
45
50


Roll, Pitch, Yaw Attitude Estimates 5
Roll (Red)
Pitch (Green)
Yaw (Blue)
Time (s)
16
14
12
10
8
6
4
2
0
-2
0
10
20
30
40
50
60

	Initial measurement	Final measurement
Trial 1	[0.0152588, 0.00610352, -0.00610352]	[0.0152588, 0.00915527, -0.100708]
Trial 2	[0.0152588 0.00915527 -0.00610352]	[0.0152588 0.00610352 0.0946045]
Trial 3	[0.0152588 0.00915527 -0.00610352]	[0.012207 0.00610352 0.0244141]
Trial 4	[0.0152588 0.00610352 -0.00610352]	[0.012207 0.00915527 -0.521851]
Trial 5	[0.012207 0.00915527 -0.00610352]	[0.0152588 0.00610352 13.8245]

5. MainLoop full listing (with comments)

```
#include "UserCode.hpp"
#include "UtilityFunctions.hpp"
#include "Vec3f.hpp"

#include <stdio.h> //for printf

//An example of a variable that persists beyond the function call.
float exampleVariable_float = 0.0f;  //Note the trailing 'f' in the
number. This is to force single precision floating point.
Vec3f exampleVariable_Vec3f = Vec3f(0, 0, 0);
int exampleVariable_int = 0;

//We keep the last inputs and outputs around for debugging:
MainLoopInput lastMainLoopInputs;
MainLoopOutput lastMainLoopOutputs;

//Some constants that we may use:
const float mass = 30e-3f;  // mass of the quadcopter [kg]
const float gravity = 9.81f;  // acceleration of gravity [m/s^2]
const float inertia_xx = 16e-6f;  //MMOI about x axis [kg.m^2]
const float inertia_yy = inertia_xx;  //MMOI about y axis [kg.m^2]
const float inertia_zz = 29e-6f;  //MMOI about z axis [kg.m^2]

const float dt = 1.0f / 500.0f; //[s] period between successive calls
```

```
to MainLoop
Vec3f estGyroBias = Vec3f(0,0,0);
Vec3f rateGyro_corr = Vec3f(0,0,0);
float estRoll = 0;
float estPitch = 0;
float estYaw = 0;

float estRoll_g = 0;
float estPitch_g = 0;
float estYaw_g = 0;

float p = 0.01f; //rho, the gyro/accel trade-off scalar

MainLoopOutput MainLoop(MainLoopInput const &in) {
  //Gyro
  if(in.currentTime < 1.0f){
    estGyroBias = estGyroBias + (in.imuMeasurement.rateGyro /
500.0f);
  }
  rateGyro_corr = in.imuMeasurement.rateGyro - estGyroBias;

  // ***Attitude Estimators***
  // roll -> x
  // pitch -> y
  // yaw -> z

  //  ***Gyro only attitude estimator***
  //estRoll_g = estRoll_g + dt*rateGyro_corr.x;
  //estPitch_g = estPitch_g + dt*rateGyro_corr.y;
  estYaw_g = estYaw_g + dt*rateGyro_corr.z;

  // ***Gyro + accelerometer attitude estimator***
  estRoll = (1.0f-p)*(estRoll + dt*rateGyro_corr.y) +
p*(in.imuMeasurement.accelerometer.y / gravity);
  estPitch = (1.0f-p)*(estPitch + dt*rateGyro_corr.x) +
p*(in.imuMeasurement.accelerometer.x / -gravity);
  estYaw = estYaw + dt*rateGyro_corr.z;

  // The function input (named "in") is a struct of type
  // "MainLoopInput". You can understand what values it
  // contains by going to its definition (click on "MainLoopInput",
  // and then hit <F3> -- this should take you to the definition).
  // For example, "in.joystickInput.buttonBlue" is true if the
```

```
  // joystick's blue button is pushed, false otherwise.

  //Define the output numbers (in the struct outVals):
  MainLoopOutput outVals;
//  motorCommand1 -> located at body +x +y
//  motorCommand2 -> located at body +x -y
//  motorCommand3 -> located at body -x -y
//  motorCommand4 -> located at body -x +y
  outVals.motorCommand1 = 0;
  outVals.motorCommand2 = 0;
  outVals.motorCommand3 = 0;
  outVals.motorCommand4 = 0;

  //copy the inputs and outputs:
  lastMainLoopInputs = in;
  lastMainLoopOutputs = outVals;
  outVals.telemetryOutputs_plusMinus100[0] = estRoll;
  outVals.telemetryOutputs_plusMinus100[1] = estPitch;
  outVals.telemetryOutputs_plusMinus100[2] = estYaw;
  outVals.telemetryOutputs_plusMinus100[3] = rateGyro_corr.x;
  outVals.telemetryOutputs_plusMinus100[4] = rateGyro_corr.y;
  outVals.telemetryOutputs_plusMinus100[5] = rateGyro_corr.z;

//  gyro only estimator:
  outVals.telemetryOutputs_plusMinus100[6] = estRoll_g;
  outVals.telemetryOutputs_plusMinus100[7] = estPitch_g;
  outVals.telemetryOutputs_plusMinus100[8] = estYaw_g;
  return outVals;

}

void PrintStatus() {
  //For a quick reference on the printf function, see:
http://www.cplusplus.com/reference/cstdio/printf/
  // Note that \n is a "new line" character.
  // Also, note that to print a `float` variable, you have to
explicitly cast it to
  //  `double` in the printf function, and explicitly specify
precision using something
  //  like %6.3f (six significant digits, three after the period).
Example:
  //   printf("  exampleVariable_float = %6.3f\n",
double(exampleVariable_float));
```

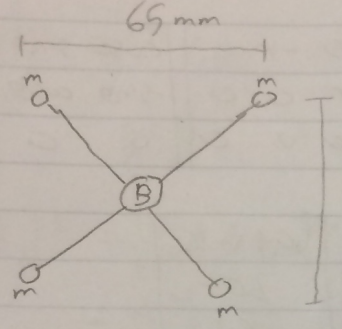
```
  //Accelerometer measurement
  printf("Acc: ");
  printf("x=%6.3f, ",
         double(lastMainLoopInputs.imuMeasurement.accelerometer.x));
  printf("y=%6.3f, ",
         double(lastMainLoopInputs.imuMeasurement.accelerometer.y));
  printf("z=%6.3f, ",
         double(lastMainLoopInputs.imuMeasurement.accelerometer.z));
  printf("\n");  //new line
  printf("Gyro bias:");
  printf("x_bias=%6.3f, ",
         double(estGyroBias.x));
  printf("y_bias=%6.3f, ",
         double(estGyroBias.y));
  printf("z_bias=%6.3f, ",
           double(estGyroBias.z));
  printf("\n");  //new line
  printf("Gyro Corrected:");
  printf("x_corrected=%6.3f, ",
         double(rateGyro_corr.x));
  printf("y_corrected=%6.3f, ",
         double(rateGyro_corr.y));
  printf("z_corrected=%6.3f, ",
         double(rateGyro_corr.z));
  printf("\n");  //new line
  printf("Raw gyro: ");
  printf("x=%6.3f, ",
double(lastMainLoopInputs.imuMeasurement.rateGyro.x));
  printf("y=%6.3f, ",
double(lastMainLoopInputs.imuMeasurement.rateGyro.y));
  printf("z=%6.3f, ",
double(lastMainLoopInputs.imuMeasurement.rateGyro.z));
  printf("\n");  //new line
  printf("Attitude: ");
  printf("estRoll_g=%6.3f, ",
         double(estRoll_g));
  printf("estPitch_g=%6.3f, ",
         double(estPitch_g));
  printf("estYaw_g=%6.3f, ",
         double(estYaw_g));

  printf("\n");
```

```
  printf("Attitude w/acc: ");
  printf("estRoll=%6.3f, ",
         double(estRoll));
  printf("estPitch=%6.3f, ",
         double(estPitch));
  printf("estYaw=%6.3f, ",
         double(estYaw));
  printf("\n");

//  printf("Example variable values:\n");
//  printf("  exampleVariable_int = %d\n", exampleVariable_int);
//  //Note that it is somewhat annoying to print float variables.
//  //  We need to cast the variable as double, and we need to
specify
//  //  the number of digits we want (if you used simply "%f", it
would
//  //  truncate to an integer.
//  //  Here, we print 6 digits, with three digits after the period.
//  printf("  exampleVariable_float = %6.3f\n",
double(exampleVariable_float));
//
//  //We print the Vec3f by printing it's three components
independently:
//  printf("  exampleVariable_Vec3f = (%6.3f, %6.3f, %6.3f)\n",
//         double(exampleVariable_Vec3f.x),
double(exampleVariable_Vec3f.y),
//         double(exampleVariable_Vec3f.z));

  //just an example of how we would inspect the last main loop inputs
and outputs:
  printf("Last main loop inputs:\n");
  printf("  batt voltage = %6.3f\n",
         double(lastMainLoopInputs.batteryVoltage.value));
  printf("  JS buttons: ");
  if (lastMainLoopInputs.joystickInput.buttonRed)
    printf("buttonRed ");
  if (lastMainLoopInputs.joystickInput.buttonGreen)
    printf("buttonGreen ");
  if (lastMainLoopInputs.joystickInput.buttonBlue)
    printf("buttonBlue ");
  if (lastMainLoopInputs.joystickInput.buttonYellow)
    printf("buttonYellow ");
  if (lastMainLoopInputs.joystickInput.buttonStart)
```

```
    printf("buttonStart ");
  if (lastMainLoopInputs.joystickInput.buttonSelect)
    printf("buttonSelect ");
  printf("\n");
  printf("Last main loop outputs:\n");
  printf("  motor command 1 = %6.3f\n",
         double(lastMainLoopOutputs.motorCommand1));
```

6. Homework Problems

(2)



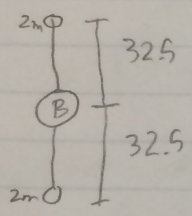
$B = 14.5 \times 10^{-3}\ kg$

$m = 3.3 \times 10^{-3}\ kg$

for a point mass

$I = mR^2$

For I_{xx}

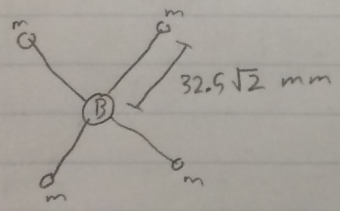


$I_{xx} = (14.5 \times 10^{-3} kg)(0\,m)^2 + 4(3.3 \times 10^{-3}\ kg)(0.0325\,m)^2$

$I_{xx} = 1.3943 \times 10^{-5}\ kg\ m^2$

since the quad is symmetric about the x and y axis $I_{xx} = I_{yy}$

For I_{zz}

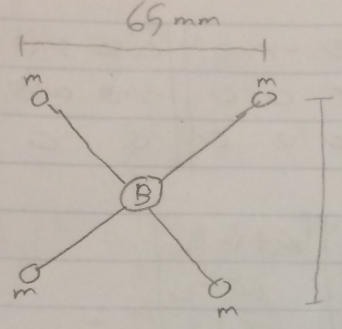


$I_{zz} = (14.5 \times 10^{-3} kg)(0\,m)^2 + 4(3.3 \times 10^{-3}\ kg)(0.0325\sqrt{2}\,m)^2$

$I_{zz} = 2.7885 \times 10^{-5}\ kg\ m^2$

$$\left[I_B^B\right]^B = \begin{bmatrix} 1.3943 \times 10^{-5} & 0 & 0 \\ 0 & 1.3943 \times 10^{-5} & 0 \\ 0 & 0 & 2.7885 \times 10^{-5} \end{bmatrix}$$

(2)



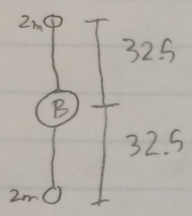
$B = 14.5 \times 10^{-3}\ kg$

$m = 3.3 \times 10^{-3}\ kg$

for a point mass

$I = mR^2$

For I_{xx}

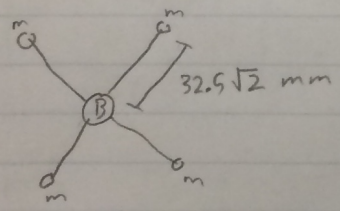


$I_{xx} = (14.5 \times 10^{-3} kg)(0\,m)^2 + 4(3.3 \times 10^{-3}\ kg)(0.0325\,m)^2$

$I_{xx} = 1.3943 \times 10^{-5}\ kg\ m^2$

since the quad is symmetric about the x and y axis $I_{xx} = I_{yy}$

For I_{zz}



$I_{zz} = (14.5 \times 10^{-3} kg)(0\,m)^2 + 4(3.3 \times 10^{-3}\ kg)(0.0325\sqrt{2}\,m)^2$

$I_{zz} = 2.7885 \times 10^{-5}\ kg\ m^2$

$$\left[I_B^B\right]^B = \begin{bmatrix} 1.3943 \times 10^{-5} & 0 & 0 \\ 0 & 1.3943 \times 10^{-5} & 0 \\ 0 & 0 & 2.7885 \times 10^{-5} \end{bmatrix}$$

3.

angular velocity: $\left(\underline{\omega}^{BE}\right)^B = \begin{bmatrix} 5 \\ 0 \\ 2 \end{bmatrix}$ rad/s

torque: $\left[\underline{n}_B\right]^B = ?$

M.M.I: $\left[\underline{I}_B{}^B\right]^B = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 0 & 5 \end{bmatrix}$ kg·m²

✱ Euler's Law ✱: $\underline{n}_B = D^E \underline{l}_B{}^{BE}$

euler's transform: $D^E \underline{l}_B{}^{BE} = D^B \underline{l}_B{}^{BE} + \underline{\Omega}^{BE} \underline{l}_B{}^{BE}$

$D^B \underline{l}_B{}^{BE} = D^B \left(\underline{I}_B{}^B \underline{\omega}^{BE}\right)$

product rule: $= \left(D^B \underline{I}_B{}^B\right)^{\to 0} \underline{\omega}^{BE} + \underline{I}_B{}^B \left(D^B \underline{\omega}^{BE}\right)$

✱ $\underline{l}_B{}^{BE} = \underline{I}_B{}^B \underline{\omega}^{BE}$ ✱

$D^B \underline{l}_B{}^{BE} = \underline{I}_B{}^B \cdot \left(D^B \underline{\omega}^{BE}\right)$

$\underline{n}_B = D^E \underline{l}_B{}^{BE} = D^B \underline{l}_B{}^{BE} + \underline{\Omega}^{BE} \underline{l}_B{}^{BE}$

$= \underline{I}_B{}^B \cdot D^B \underline{\omega}^{BE} + \underline{\Omega}^{BE} \left(\underline{I}_B{}^B \cdot \underline{\omega}^{BE}\right)$

$\left[\underline{n}_B\right]^B = \left[\underline{I}_B{}^B\right]^B \cdot \left[D^B \underline{\omega}^{BE}\right]^B + \left[\underline{\Omega}^{BE}\right]^B \left[\underline{I}_B{}^B\right]^B \left[\underline{\omega}^{BE}\right]^B$

↑ constant matrix

$\left[\underline{n}_0\right]^B - \left[\underline{\Omega}^{BE}\right]^B \left[\underline{I}_B{}^B\right]^B \left[\underline{\omega}^{BE}\right]^B = \left[\underline{I}_B{}^B\right]^B \left[\frac{d}{dt} \underline{\omega}^{BE}\right]^B$

$\left[\frac{d}{dt} \underline{\omega}^{BE}\right]^B = \left(\left[I_B{}^B\right]^B\right)^{-1} \left(\left[\underline{n}_B\right]^B - \left[\underline{\Omega}^{BE}\right]^B \left[\underline{I}_B{}^B\right]^B \left[\underline{\omega}^{BE}\right]^B \right)$

constant velocity → acc = 0

$\left[\underline{n}_0\right]^B = \left[\underline{I}_B{}^B\right]^B \left[\frac{d}{dt} \underline{\omega}^{BE}\right]^B + \left[\underline{\Omega}^{BE}\right]^B \left[\underline{I}_B{}^B\right]^B \left[\underline{\omega}^{BE}\right]^B$

✱ $\left[\underline{\Omega}^{BE}\right]^B = \begin{bmatrix} 0 & -\omega_3 & +\omega_2 \\ +\omega_3 & 0 & -\omega_1 \\ -\omega_2 & +\omega_1 & 0 \end{bmatrix} \Leftrightarrow \left[\underline{\omega}^{BE}\right]^B = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$

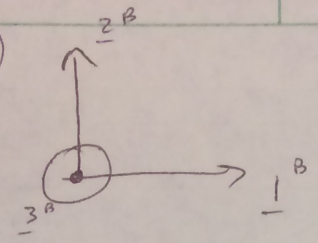
$\left[\underline{n}_B\right]^B = \begin{bmatrix} 0 & -2 & 0 \\ 2 & 0 & -5 \\ 0 & 5 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 0 & 5 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 2 \end{bmatrix}$

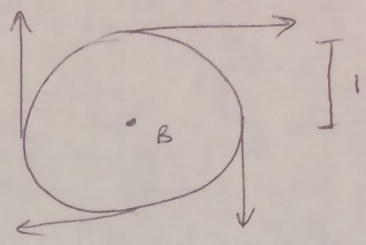
$\left[\frac{1}{s}\right] \left[\text{kg} \cdot \text{m}^2\right] \left[\frac{1}{s}\right] = \frac{\text{kg} \cdot \text{m}^2}{s^2}$

$$[\underline{n}_B]^B = \begin{bmatrix} -2 & -6 & 0 \\ -1 & 2 & -23 \\ 5 & 15 & 6 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} -10 \\ -51 \\ 25 \end{bmatrix} \frac{kg \cdot m^2}{s^2} = \boxed{\begin{bmatrix} -10 \\ -51 \\ 25 \end{bmatrix} N \cdot m = [\underline{n}_B]^B}$$

4.

(4)





thrust is rotating craft counter CW.

$n = r \times a$, $r = l$ for all thrusters

$\Rightarrow n_i = a_i$

so $n_{total} = \sum_{i=1}^{4} a_i$

$$\left[\underline{n_B}\right]^B = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} a_i \end{bmatrix}$$