University California Santa Cruz
Electrical and Computer Engineering Deparment
ECE 167: Signals and Systems

Lab 4: Attitude Estimation

Kyle Jeffrey

February 26, 2020

1 Introduction

From the lab manual,

"The purpose of this lab is to get a handle on sensor fusion, and learn to integrate the gyros and other three-axissensors in order to develop a useable attitude estimation algorithm driven by the actual sensors you have. Youwill learn how to use the accelerometer and magnetometer to estimate the gyro biases on the fly and to correctthe attitude estimate. Attitude estimation, while somewhat esoteric, is a crucial component of all autonomousnavigation and demonstrates sensor fusion in a readily accessible way. Having a functional attitude estimationsystem (both hardware and software) will definitely set you apart from most other students."

The calibration methods of the previous lab will be used here to fuse the accelerometer, magnetometer and gyro into one attitude sensor. This is known as sensor fusion. The specific implementation for sensor fusion being used is complimentary filtering. An attitude is otherwise known as an AHRS(Attitude Heading Reference System)

2 Familiarization with MATLAB Code

Before any programming or system assembly is done, it is important to become familiar with this complimentary filtering method of attitude estimation. The lab manual provides a quick explanation of the coordinate frames at work:

"Before you can imagine what is going on with the Direction Cosine Matrix(DCM), you need to get the concept of coordinate framesdown. Specifically, we imagine a BODY frame withxout the nose,yout the right wing, andzdown (tomake a right handed coordinate frame). The BODY frame is rigidly attached to the body, such that it movesand rotates with the body. The other coordinate frame we use is the INERTIAL frame, which is a coordinateframe that is typically defined asXpointed North,Ypointed East, andZpointed down (the so-called NED definition)"

The DCM contains all of the information about the inertial and body frames. The gyro sensor output is in deg/sec. In the previous lab, a discrete summation was used to get the rotation of the sensor, but as will be explored later, this is not a simple matter.

2.1 Open Loop Integration

Here is the output of a provided MATLAB script that uses the DCM integration to find the rotation of the gyro. Where $\hat{R}$ is considered the DCM and $\vec{\omega}$ is the gyro sensor data, the formula for integration is: $\hat{R} = \hat{R}R_{exp}(\vec{\omega})$.

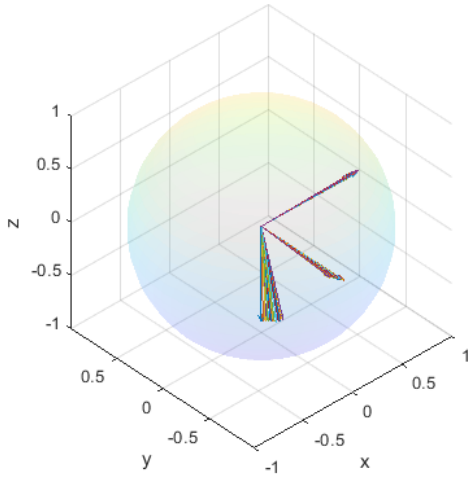


Figure 1: Integration of Static Rotation w. Bias

The $R_{exp}()$ operator is know as the "Rodriguez Parameter" and is beyond the scope of this paper. Just know that this Rodriguez Parameter maintains orthornormality for the DCM. The next section will discuss the importance of maintaining orthornormality when converting to Euler Angles.

3 DCM to Euler

The DCM contains all important information of the body frame and inertial frame, and is in fact the transformation between the two. Using the DCM, the roll, pitch, and yaw will be determined. Here is a quick visual reminder of this NED coordinate system:

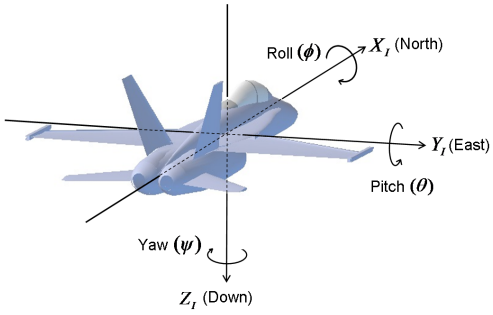


Figure 2: NED Coordinates

There are papers written on how the DCM parameters were mathematically determined, but essentially it is a consecutive application of yaw, pitch, and roll transformations of a 3D system in spatial dimensions. Shown here.

$$[\mathcal{R}] = \begin{bmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{bmatrix}$$

Figure 3: DCM Parameters

From the DCM, the following equations are derived and give the values for yaw(ψ), pitch(θ), roll(ϕ), though, extra analysis needs to be done to determine what quadrant the angles are in as $\arctan$ only has a range of -90, 90 degrees. Math.h standard C library has the atan2() function which handles this:

$$\theta = \arcsin(-R_{13})$$
$$\psi = \arctan(R_{23}/R_{33})$$
$$\phi = \arctan(R_{12}/R_{11})$$

IF the DCM Rotation Matrix begins to lose orthornormality as the vectors are no longer unit length, the equations for the Euler Angles become invalid. $\arcsin()Domain = [-1, 1]$. This is the primary advantage of the Rodriguez Parameter, as it maintains orthornormality. In the next section, regular forward integration will be tested to determine it's disadvantages.

4 Integration of Constant Body-fixed Rates

To become familiar with the gyro sensor and DCM, some simulated analysis and experiments analysis was done. In a simulated approach, a static value for rotation rates of the gyro were set to gain some familiarity with finding the DCM using matrix math in C. After that, the Gyro was directly used to determine real-time DCM values. This section will also explore the two techniques for integrating Gyro velocity data into spatial values. There are two ways(there could be others) of converting the gyro deg/second velocity data into deg of rotation position data. Those are *"(1) using the simple forward integration ; and (2) using the matrix exponential form."* . Forward integration in matrix form is: $R^+ = R^- + [\vec{\omega}\times]R^-\Delta t$

4.1 Forward Integration

The first method of finding the rotation matrix is to do basic integration of the gyro rotation rates. In matrix form this looks somewhat cryptic, but essentially the skew symmetric "$\vec{\omega}$" matrix is used to find $\int_0^t V_k dx$ where V_k is replaced with a gyro reading for each axis.

4.1.1 Simulated Forward Integration

A matrix math library was used from a previous class to do all of the float matrix multiplication between the skew symmetricc matrix and the rotation matrix. With a rotation rate of 1 deg/sec set statically and a simulated polling rate of 1 Hz, below is a graph of the rotation of the body frame for 45 seconds, i.e. 45 degrees of rotation.

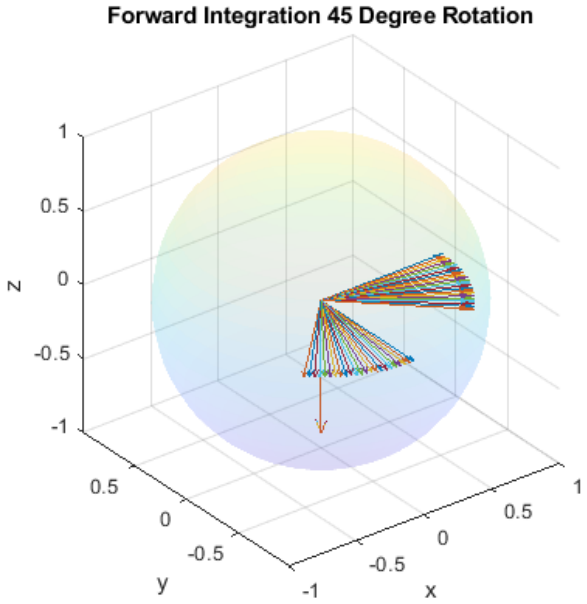


Figure 4: Simulated Gyro Rotation

The rotation appears fairly stable from the forward integration but when left to this forward integration, the DCM doesn't maintain orthornormality. Here is a graph of the body frame vectors after nine revolutions. Notice that x and y vectors are no longer unit vectors:

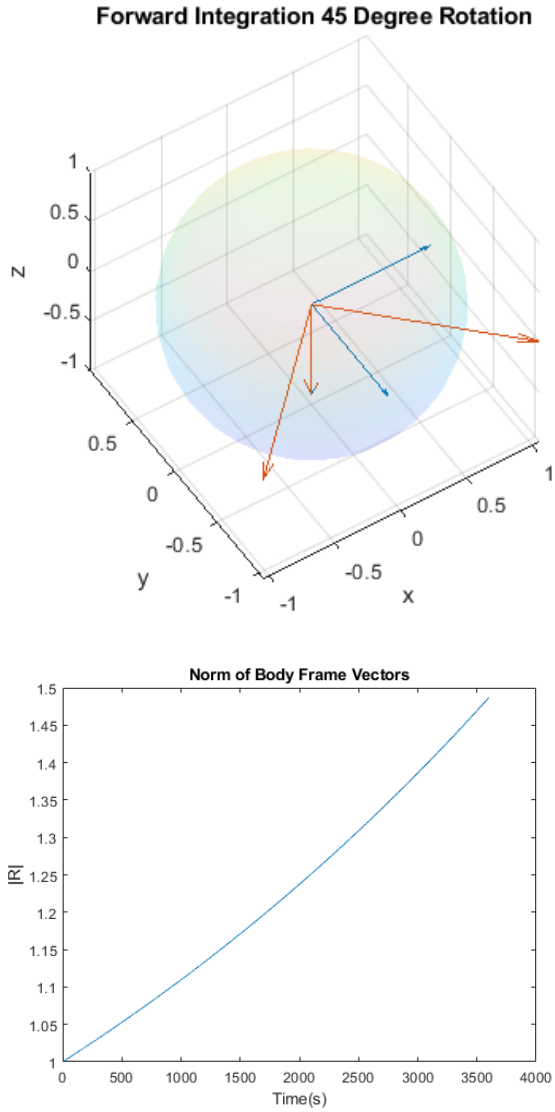


Figure 5: Simulated Gyro Rotation

4.1.2 Experimental Forward Integration

Using the same technique now, the Gyro data with previously determined biases, and scale factor was used to find the DCM and then extract Euler Angles to print to the OLED screen. Here is the drift of the DCM out of orthornormality after 6 rotations:

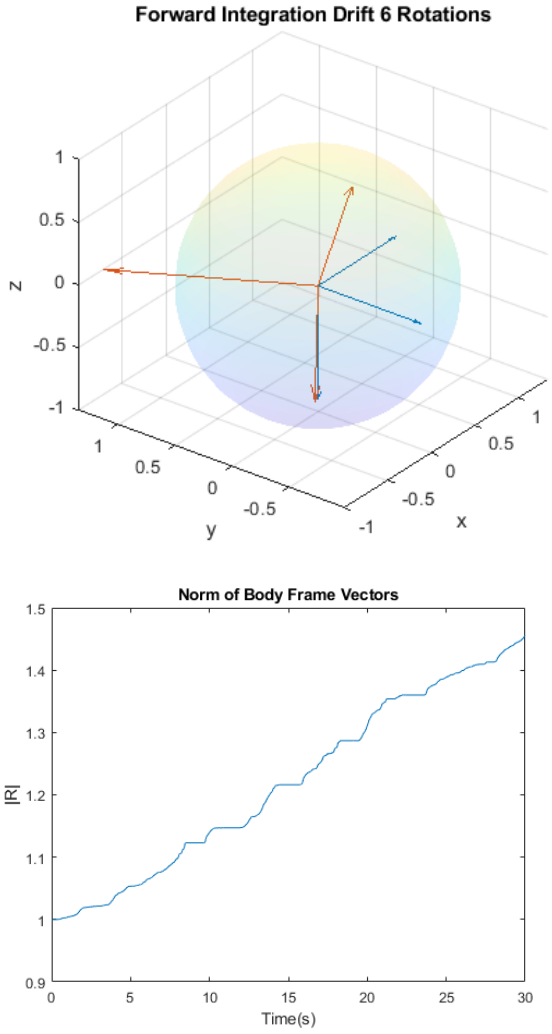


Figure 6: Experimental Gyro Rotation

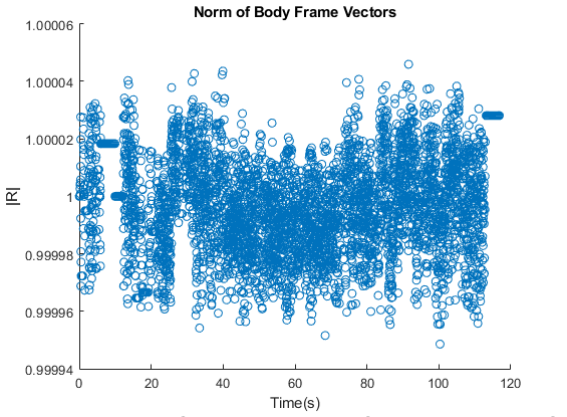
Notice how this drift only took 30 seconds. This is a huge issue to deal with and shows that simple forward integration can create huge drifting errors in a short period of time.

4.2 Rodrigues Rotation Matrix

Now, the matrix exponential form building the DCM will be tested with gyro data experimentally and it's drift will be analyzed. It should be such that the orthornormality of the DCM is maintained but, there will be errors in the DCM in regards to the gyro sensor in real space. The Rodrigues Exponential is $Rexp(\omega\Delta t) = I + sinc(\omega\Delta t_{norm}) * cos(\omega\Delta t_{norm}) * \omega\Delta t + sinc(\omega\Delta t_{norm})^2/2 * (\omega\Delta t)^2$

4.2.1 Experimental Data Matrix Exponential

Now using the Rodrigues Rotation Matrix with the gyro, the values will be analyzed, paying notice to any gyro drift, and how well the sensor actually tracks rotation.



1.0001	7.4000e-05	1.7150e-05
7.4000e-05	1.0001	3.0100e-06
1.7150e-05	3.0100e-06	1.0000

Figure 7: Rodrigues Drift and R'R

Multiplying the last value of the DCM by it's transpose will should give the identity matrix if the DCM is orthornormal. After 2 minutes, the DCM is still very close to orthornormality, off by some .0001 values. This is negligible and due to some floating point error, discussed more in the next section.

4.3 Float Point Error Issues

The Rodrigues Rotation Matrix is supposed to maintain orthornormality of the DCM. Interestingly though, in testing with simulated data, there was a noticeably amount of drift away from orthornormality. The simulated data here of the DCM vector lengths here shows this drift.

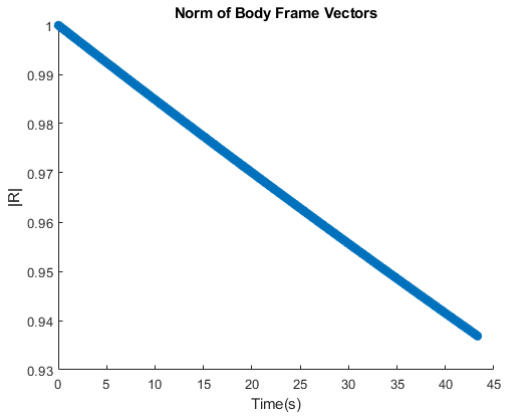


Figure 8: Simulated Rodriguez Rotation Drift

This should not to be the case, and must be from floating point error arithmetic, as the matrix library being used is based on a float 3x3 matrix. The nature of this error is now analyzed here. First, the lab manual noted issues with implementing the sinc() function in c. It suggested using a Taylor series expansion for values near 0, but in testing, setting sinc(x) to 0 when abs(x) ¡ .001 gave very accurate data. Shown here in a graph with what the actual values should be. They graphs are the exact same.

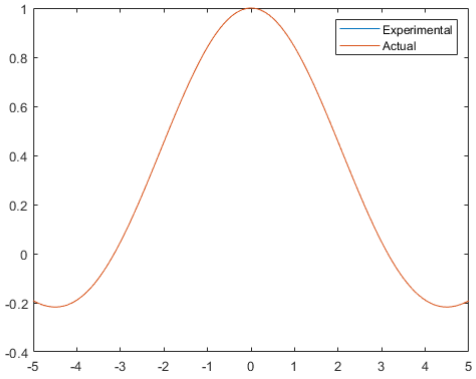


Figure 9: Sinc Data Comparison

With the sinc() function not being the issue, it was noticed that the ω when scaled by Δt crated a lot of floating point error. This is mostly exacerbated when calculating ω_{norm} which is calculated by $\omega_{norm} = \sqrt{p^2 + q^2 + r^2}$. To fix this, the ω wasn't scaled until after ω_{norm} was calculated.

5 Open Loop Gyro Integration

It's important to find whether the integration techniques used here are accurate in determining degrees of rotation. With a 50Hz polling rate default, is the resolution high enough to find the correct rotation? This will be explored with MATLAB code that creates trajectory data and actual Euler angle values that can be compared. The analysis will include how noise and bias affect the outcome. The orthornormality will not be examined in this section, as the previous section explored this in depth.

5.1 Open Loop Using Simulated Data

Using the provided matlab file createTrajectoryData.m, simulated data will be put through both forward integration and the matrix exponential forms.

5.1.1 Forward Integration

Using gyro data simulated by the provided MATLAB file, the two algorithms for integration were tested. Forward Integration is explored here. The real Euler angles were given and the error is shown in a graph over time here between the axes. Though, some of these values may seem worse than they are, as the angles wrap around when $|angle| > 180$ degrees.

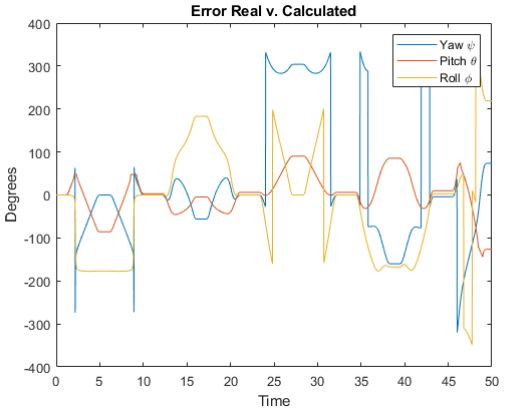


Figure 10: Forward Integration Error w/ Starting Angles

The error across time in degrees had a **standard deviation = 109.3** and a **mean error = 68.3.** This is extremely bad.

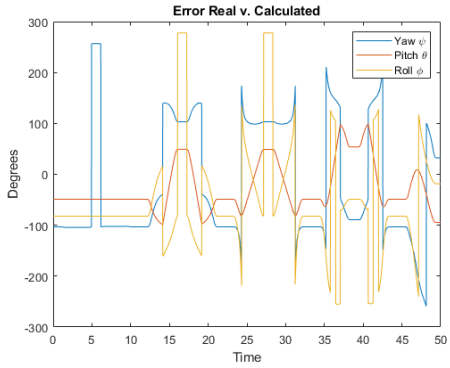


Figure 11: Forward Integration Error w/o Starting Angles

When the DCM is not set to the starting Euler Angles of the trajectory data, they will never converge back to the correct angles.

5.1.2 Matrix Exponential

The matrix exponential is then tested for error and between real Euler angles and calculated. This error set shows a big difference in accuracy between the two methods of integrating the gyro sensor data.

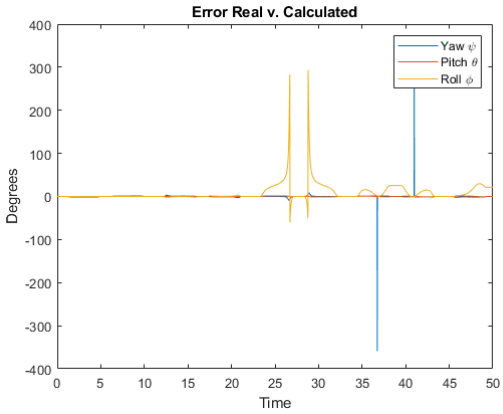


Figure 12: Matrix Exponential Integration Error

The error across time in degrees for had a **standard deviation = 13.78 degrees** and a **mean error =**

3.37 degrees. This method of integration is leagues ahead of the previous one. Another bonus is maintaining orthornormality as described in the previous section.

6 Closed Loop Integration

The gyro sensor based off the previous parts has no reference for the inertial frame. Ideally, it should know where geographic north is and where down is in respect to some absolute frame. As far as the current setup, the gyro only measures rotation away from it's starting position when turned on. So now, the magnetometer and accelerometer will be used to compare what down is, in respect to gravity measures from the accelerometer, and where north is, in respect to the magnetometers reading of magnetic north. The provided MATLAB file IntegratedClosedLoop.m will be investigated to build familiarity with this process.

The closed loop uses to corrections for the feedback loop. The first is the error correction, in this case with the z-axis, and the second is a bias estimate. The error correction is what corrects the absolute angle of rotation for the pitch and roll of the system. This correction is investigated here.

6.1 Z-Axis Error Correction

This error correction is calculated as $wmeas_a = \vec{a}_b \times R'\vec{a}_i$ where i stands for inertial frame and b stands for body frame. In essence, the difference between the DCM and the accelerometer are measured, as the accelerometer should always be pointing down towards earth. Before the gyro values are integrated the error is added as $\omega += Kp_a * wmeas_a$

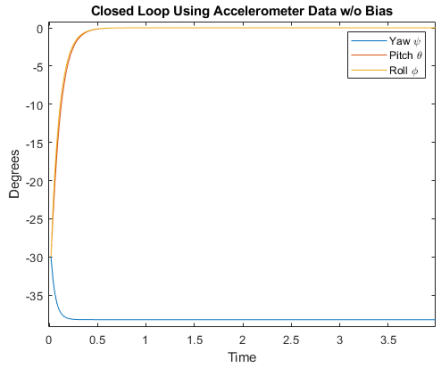


Figure 13: Closed Loop Accel Correction w/o Bias

Without the bias terms introduced yet, notice that the pitch and roll axis converge on to the correct absolute rotations which have been set to 0 here. Now here's an investigation with bias on the gyro terms.

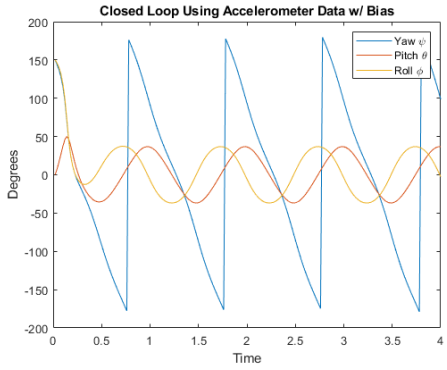


Figure 14: Closed Loop Accel Correction w/ Bias

When not using the bias estimate feedback correction here, the pitch and roll will attempt to be corrected, but they cannot account for the bias error. As the values get closer to the correct angle, the difference will decrease in the error correction and will be overpowered by the bias in the gyro values. This is why bias estimate is needed. Again, the accelerometer can do nothing to account for the absolute angle of the yaw, as it only contains information about the

down inertial vector.

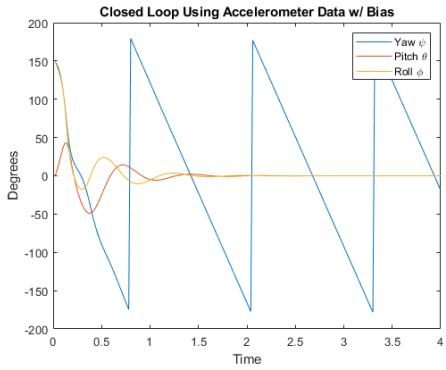


Figure 15: Closed Loop Accel Correction w/ Bias

When using the bias estimate which is $B_{estimate}+ = -Ki_a * wmeas_a$ the pitch and roll now level out, even with bias, and hold absolute positions.

6.2 Tuning the Gains(Simulated Data)

The gains are the coefficients for both the error correction and bias estimate correction of the gyro. Here the values of these gains are investigated with the accelerometer error measurement($wmeas_a$) and the accelerometer error measurement($wmeas_m$).

6.2.1 Accelerometer Feedback

Here are two graphs comparing the error correction of the gyro integration w/ bias terms with coeffecients set to $Kp_a = 1, Kp_a = .1$ respectively:

Figure 16

7 Feedback Using only Accelerometer

Now all of the analysis will be put into action on the microcontroller in c using the matrix math library. The code developed in 6.1 on MATLAB will be used to compare results for an expected value. Again, the coefficients will be fiddled with to determine the best noise to correction ratio.

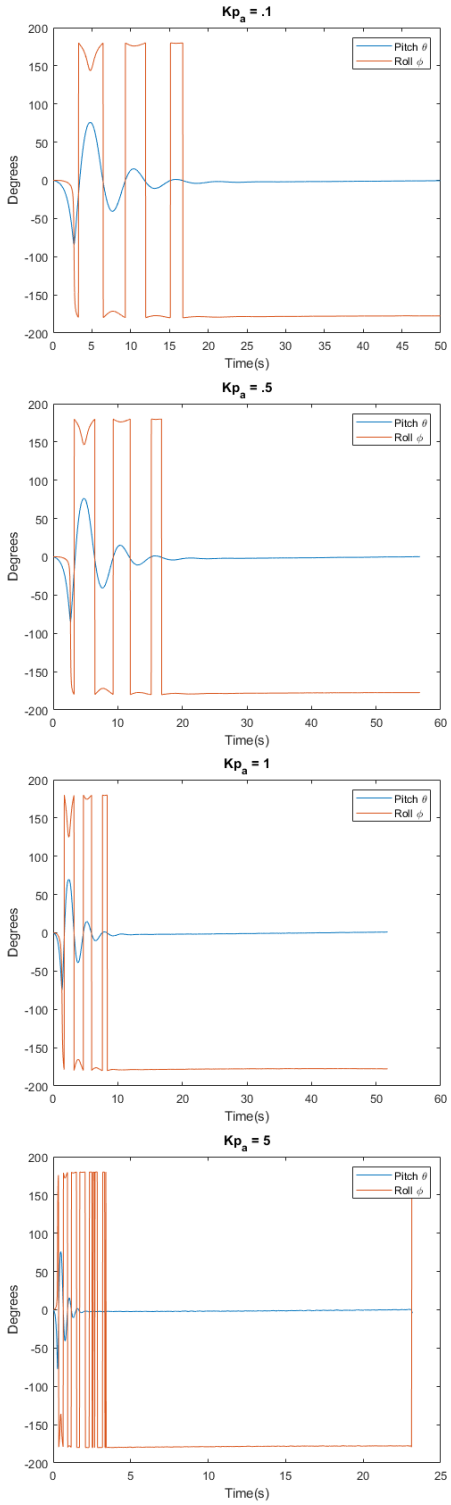


Figure 17: Kp = .1, .5, 1, 5

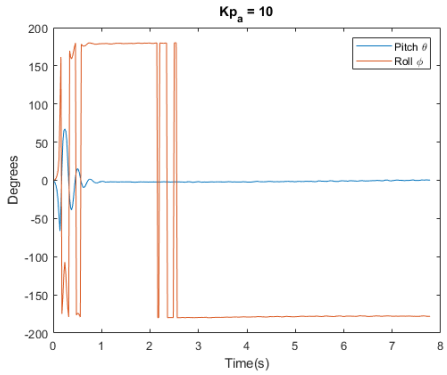


Figure 18: Kp = 10

The accelerometer appears to be giving very noise free signal so setting the coefficient Kp to 10 gives good feedback response, and notice that it reduces acquiring the initial position from about 15 seconds down to about 1 second.

After exploring the coefficients and how they effect the data, they are now compared here with the MATLAB script for closed loop integration to view expected values and actual values of the DCM and Euler angles. Here is a comparison between actual and experimental with Kp = 10 and Ki = .5. The sensor is placed upside down to see how the feedback converges to a 180 degree rotation.

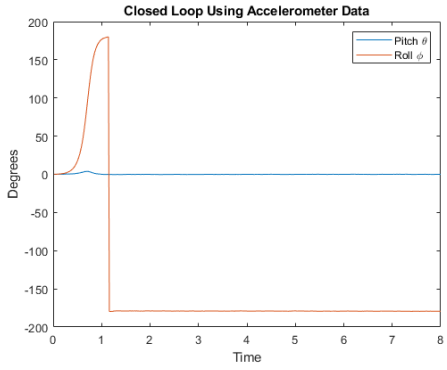


Figure 19: Expected Closed Loop Accelerometer

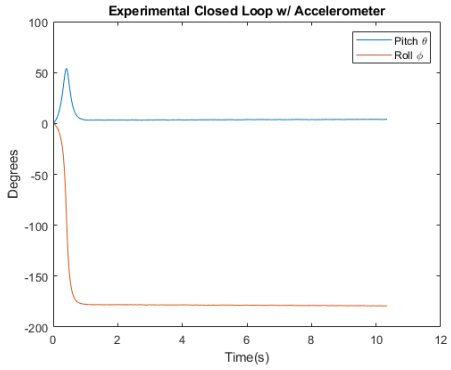


Figure 20: Actual Closed Loop Accelerometer Feedback

The responses appear to be comparable, though the axes approach 180 degree rotation from the other side.

8 Misalignment of Accelerometer and Magnetometer

After analysis of the accelerometer feedback and knowing that it works in correcting the inertial down frame of the IMU, the magnetometer should not be used to give the inertial frame correction for the north axis in the inertial frame. It has been assumed up to now that the accelerometer and magnetometer are compatible but, this is not actually the case. The Earths magnetic field does not point level cardinal north. It points at an angle downwards into the earth, but the accelerometer points directly down. If the magnetometer data is then used to attempt to correct the north inertial vector of the DCM, there will be error, as the two sensors are not orthogonal to each other. Here this will be accounted for. Using provided MATLAB code and tumble data from calibrated accelerometer and magnetometer data, a "misalignment matrix" will be calculated, which will now be reffered to as R_{mis}. The lab manual describes the process as such:

"This is going to be done using some fancy linear algebra techniques that you don't really need to understand (seeProf.Elkaim's misalignment paper if you want to). In short, you are going to use a set of n Wahba's problem solutions to figure out the individual rotations for each measurement pair, and then use those rotations to generate a single set of points to solve for them is alignment between master and slave axis sets. One of the sensors is set to "Master." In our case this is going to be the accelerometer. The magnetometer is going to be the "Slave" sensor. A misalignment matrix will be generated such that the magnetometer measurements can be put in a consistent axis set as the accelerometer"

Running the MATLAB script on some calibrated tumble data gave this R_{mis} :

0.9999	0.0101	-0.0054
-0.0101	0.9998	0.0153
0.0055	-0.0152	0.9999

Figure 21: Missalignment Matrix

Which means the misalignment between magnetometer and accelerometer in degrees are:

$$Pitch(\psi) = .3075$$
$$Roll(\theta) = .8760$$
$$Yaw(\phi) = .5814$$

This appears to be smaller misalingment then expected. Further analysis might be done in subsequent sections.

9 Full Complementary Attitude Estimation

Implement the ”full monty” fusion sensor with both magnetometer and accelerometer feedback

10 Confirmation of Gyro Integration using Wahba's Problem Solution

With the full implementation of the fusion sensor accomplished, the DCM will now be compared to a solution to the Wahba's problem called TRIAD. TRIAD takes the magnetometer and accelerometer calibrated data and produces a DCM. This DCM will be compared to the full fusion sensor DCM used on the board.

10.1 Simulated TRIAD on MATLAB

Using the MATLAB script for raw magnetometer and accelerometer to DCM, the simulated DCM from TRIAD was compared to the closed loop feedback system on the PIC32 here.

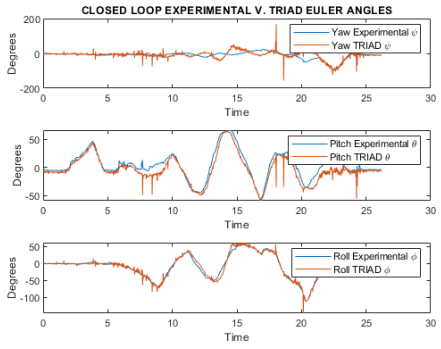


Figure 22: Simulated TRIAD vs. Closed Loop Integration

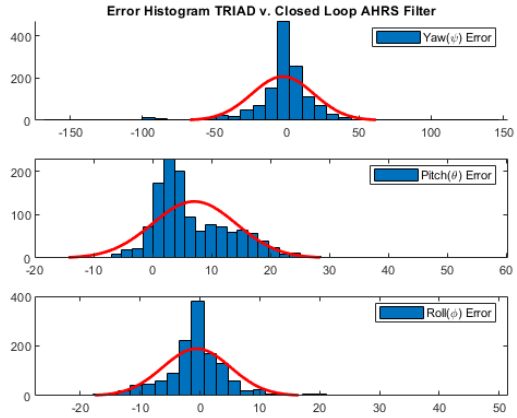


Figure 23: Simulated TRIAD v. Closed Loop Error Histogram

The corresponding standard deviations for the axes are:

$$std_\psi = 21.4$$
$$std_\theta = 7.1$$
$$std_\phi = 5.6$$

The error for pitch and roll appear to be within reason abstractly speaking, but the yaw has a significantly larger standard deviation. **In the next section it became apparent that the PIC32 implementation of the closed loop AHRS filter wasn't actually using the correct magInertial frame. The vector used in this part was $[1; 0; 0]$ which is not the correct inertial vector to use if the objective is for the attitude sensor to have its north vector set to cardinal north. My assumption is that this yaw pitch and roll error is then due to the magnetomer inertial vector used. Applying the inertial vector as $[1; 0; 0]$ sets the inertial north vector to the magnetometer's max value, which IS NOT GEOGRAPHIC NORTH. The magnetometer points**

down and angled away from geographic North because of Earth's magnetic field. In the experimental TRIAD vs. Closed Loop Integration this will be accounted for, and hopefully different results arise.

10.2 PIC32 TRIAD vs. Closed Loop

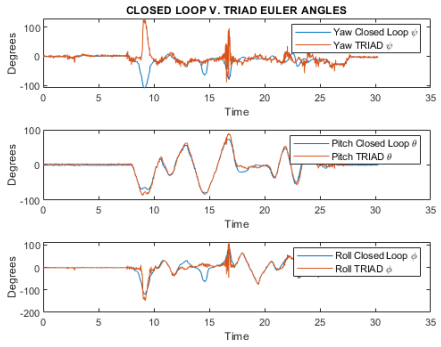


Figure 24: TRIAD vs. Closed Loop Integration

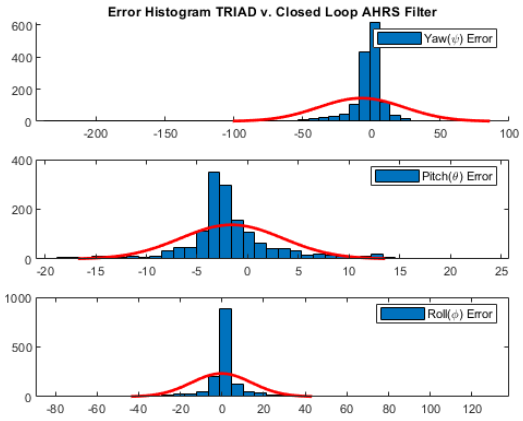


Figure 25: TRIAD v. Closed Loop Error Histogram

The standard deviatons for the board implementation are:

$$std_\psi = 31.1$$
$$std_\theta = 5.04$$
$$std_\phi = 14.4$$

The standard deviations don't appear to be any better on the board BUT, there is no longer much of a skew as before with the Pitch. This must be because of the new magnetometer inertial vector used.

11 Conclusion

The previous lab introduced calibration techniques for raw sensor data, and this lab drove the importance of good calibration. Many times throughout the implementation of the attitude sensor weird behavior was an effect of poor calibration. With the calibration techniques and Familiarization with DCM rotation matrices, attitude sensors are now well understood. There are several ways to accomplish this including using the TRIAD algorithm vs. the closed loop feedback systems.