ECE167L

Xingyu Zhou

Max Dunne

2020/03/18

Lab4 Report

Introduction:

I would like to see this lab as an extension to Lab3, because now we need to combine the gyro, accelerometer and the magnetometer to do attitude estimation. A large part of this lab is to figure out how the math of attitude estimation algorithm works, and another part is to rewrite the MATLAB script with C. There are 8 parts in total in this lab.

Part1: Conversion from DCM to Euler Angles

In this part, we are given the direction cosine matrix that contains the yaw, pitch and roll and we need to write a C code to extract all of them. The method is very straightforward.

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

By looking at the DCM above, if I need to extract pitch(θ), I'll simply take the arcsine of $DCM(0,2)$. For roll(Φ), I'll take the arctangent of $\frac{DCM(1,2)}{DCM(2,2)}$ and for the yaw(Ψ), I'll take the arctangent of $\frac{DCM(0,1)}{DMC(0,0)}$. Since they are all in radians when being extracted, I'll have to convert them to degrees. A simple way to do that is to multiply each result with $\frac{180}{\pi}$. Finally, the lab manual asks us to check if the matrix is orthonormal. To achieve this, I simply input the whole DCM into MATLAB, take its inverse and transpose, and compare the result. As you can see in result to the left, the inverse and transpose is identical, which means the DCM matrix is orthonormal.

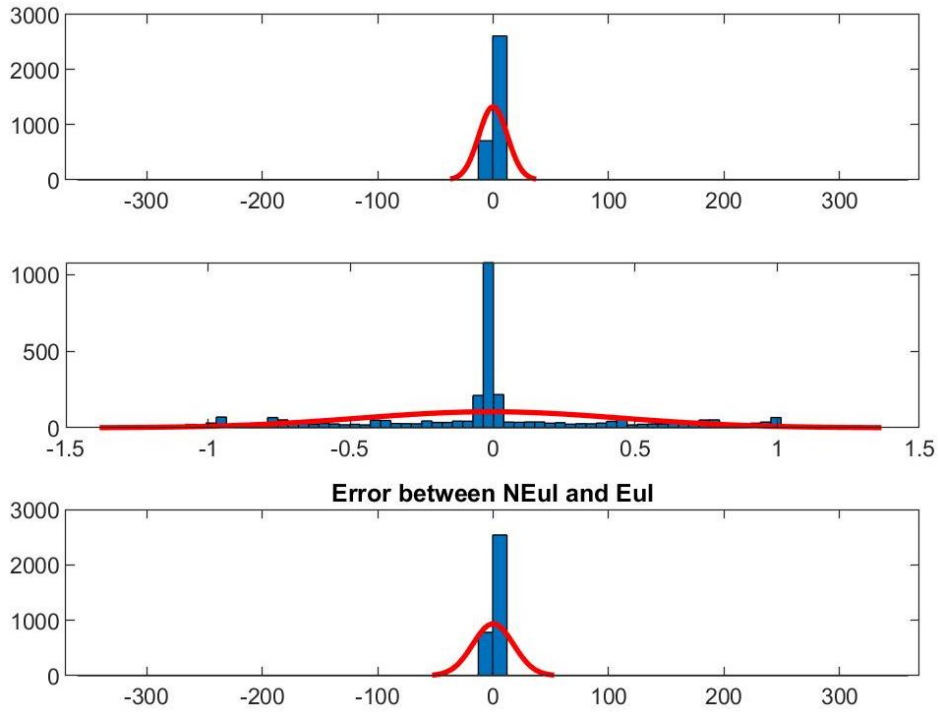
```
matrix =

   -0.2380    0.8046    0.5440
   -0.8288    0.1237   -0.5456
   -0.5063   -0.5808    0.6374


matrixT =

   -0.2380   -0.8288   -0.5063
    0.8046    0.1237   -0.5808
    0.5440   -0.5456    0.6374


matrixV =

   -0.2380   -0.8288   -0.5063
    0.8046    0.1237   -0.5808
    0.5440   -0.5456    0.6374
```

Part2: Integration of constant body-fixed rate

This part is also very straightforward. All we need to do is to rewrite a bunch of MATLAB scripts that are provided to us in C. I have to use the Matrix math library that I wrote in CE13 since this part relies heavily on matrix calculations. There aren't much to talk about in this part except how I implement the sinc function in C. There's no sinc function in C like the one in the MATLAB, so what I did is to follow the instruction online. When the input is 0, just return 1 so I can avoid the divide by 0 error and return $\frac{\sin(x)}{x}$ otherwise. By looking at the SincTest.m file provided, I noticed that I'll also have to take care of the situation when the input is less than 1. In this situation, I'll return $1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!}$. I could use more terms of the Taylor expansion, but I feel like this is enough. The MATLAB scripts that we need to rewrite are Rexp.m , rcross.m and IntegrateOpenLoop.m.

Part3: Open Loop Gyro Integration

Part3 is divided into two parts. We need to run the simulated data using MATLAB for the first half of part3 and then run the same script again using the real data collected from our gyro. One thing to notice is that we need to convert the gyro reading back to deg/s since it's scaled to bits when outputted by the CreateTrajectoryData.m function. The initial DCM is the same as the one I provided in part1. I have tried both the matrix exponential form and the forward integration and apparently the former provides better result, so the results below are from the matrix exponential form method. The order from top to buttom is yaw, pitch roll.



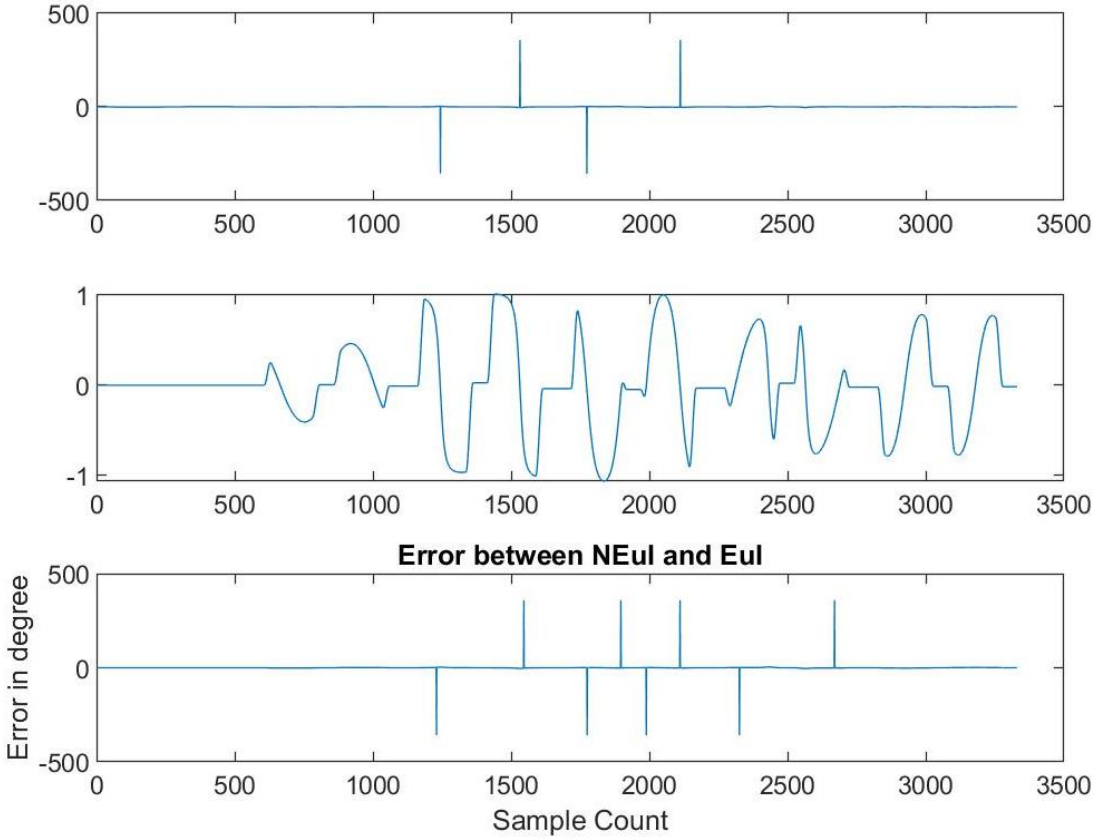
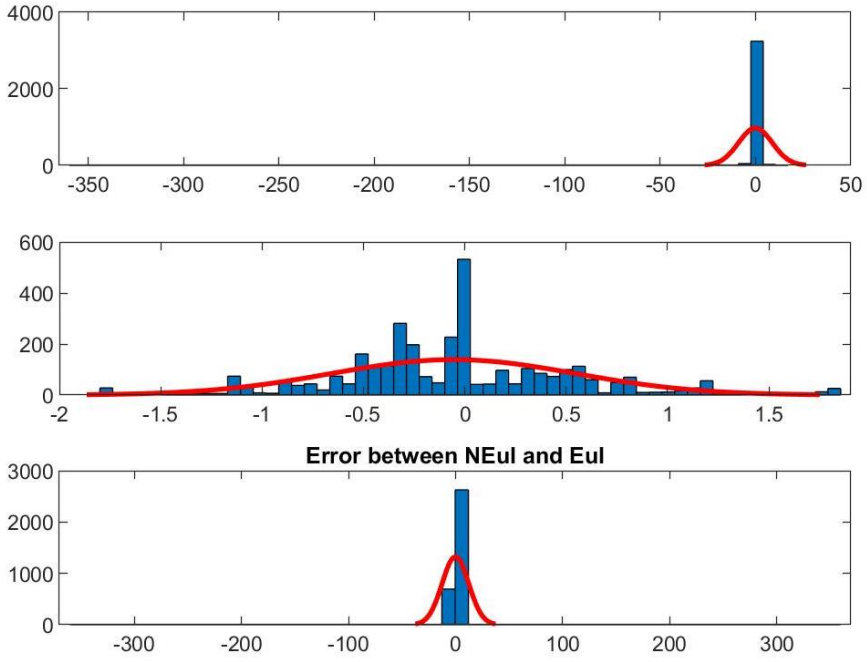


Figure 1: Error comparison between the calculated Euler angle and the expected Eular angle (no noise)



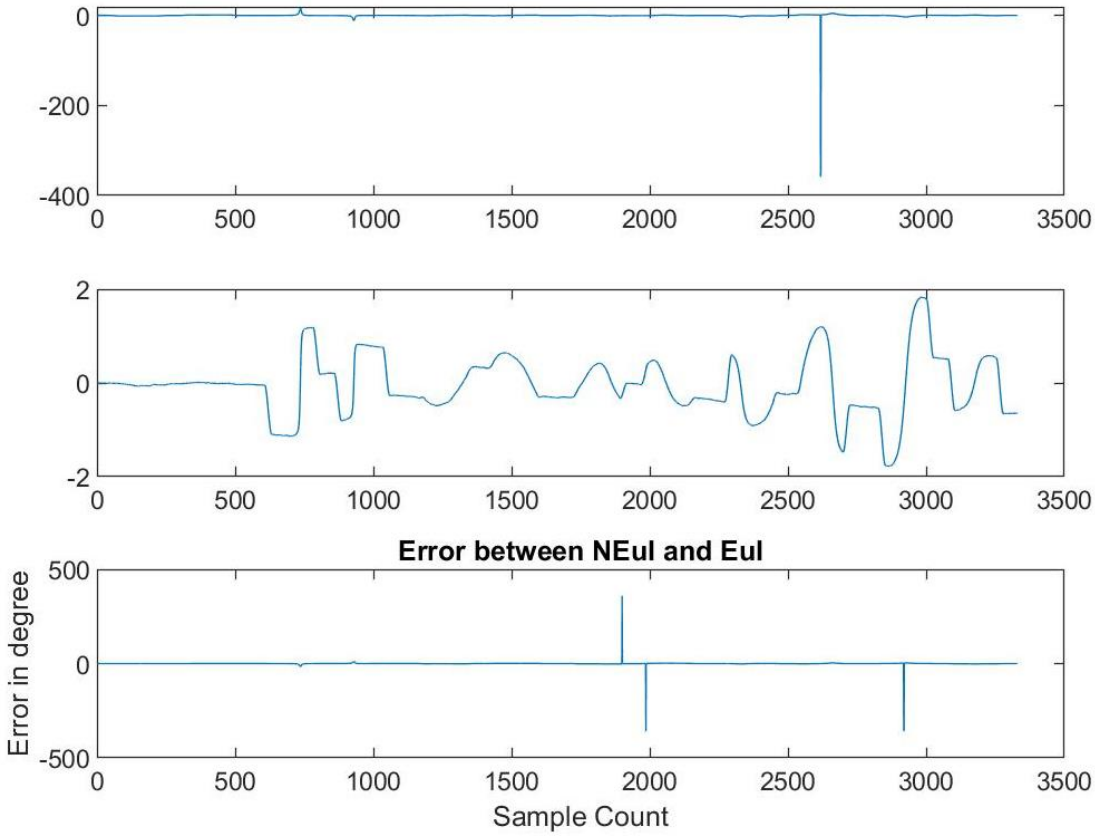


Figure 1: Error comparison between the calculated Euler angle and the expected Euler angle (with noise)

As you can see in above graphs, I was able to track both yaw and roll pretty well both with and without noise. There are some errors with pitch over the time, but they are still acceptable. Next, we need to run the algorithm again with the real data. Since I don't have access to the original Euler angle so I cannot put the plot in report, but when I run the animateAttitude.m, the movement is pretty close to how I move my sensor when collecting data. Then moving onto the C code. Again, there's nothing in particular that I need to mention since I'm just rewriting all my MATLAB scripts. When displaying the Euler angles on the Oled, I noticed that I have a small drift, but they still reflect my movement accurately.

Part4: Close loop Estimation

From what I understand, we will need to use both the accelerometer and magnetometer reading, as well as their inertial in this function. The most different part from open loop estimation in my opinion is the bias. In open loop estimation, the bias is not changed. However, in close loop estimation, the bias is updating constantly and will be used as an input to this function. Thus, in part4, I set the bias to 0 according to the lab manual and use the proportional integral gain feedback. I have both my Kp_a and Kp_m set to 40 with Ki_a and Ki_m set to 4, this is what I got from running the close loop estimation algorithm.

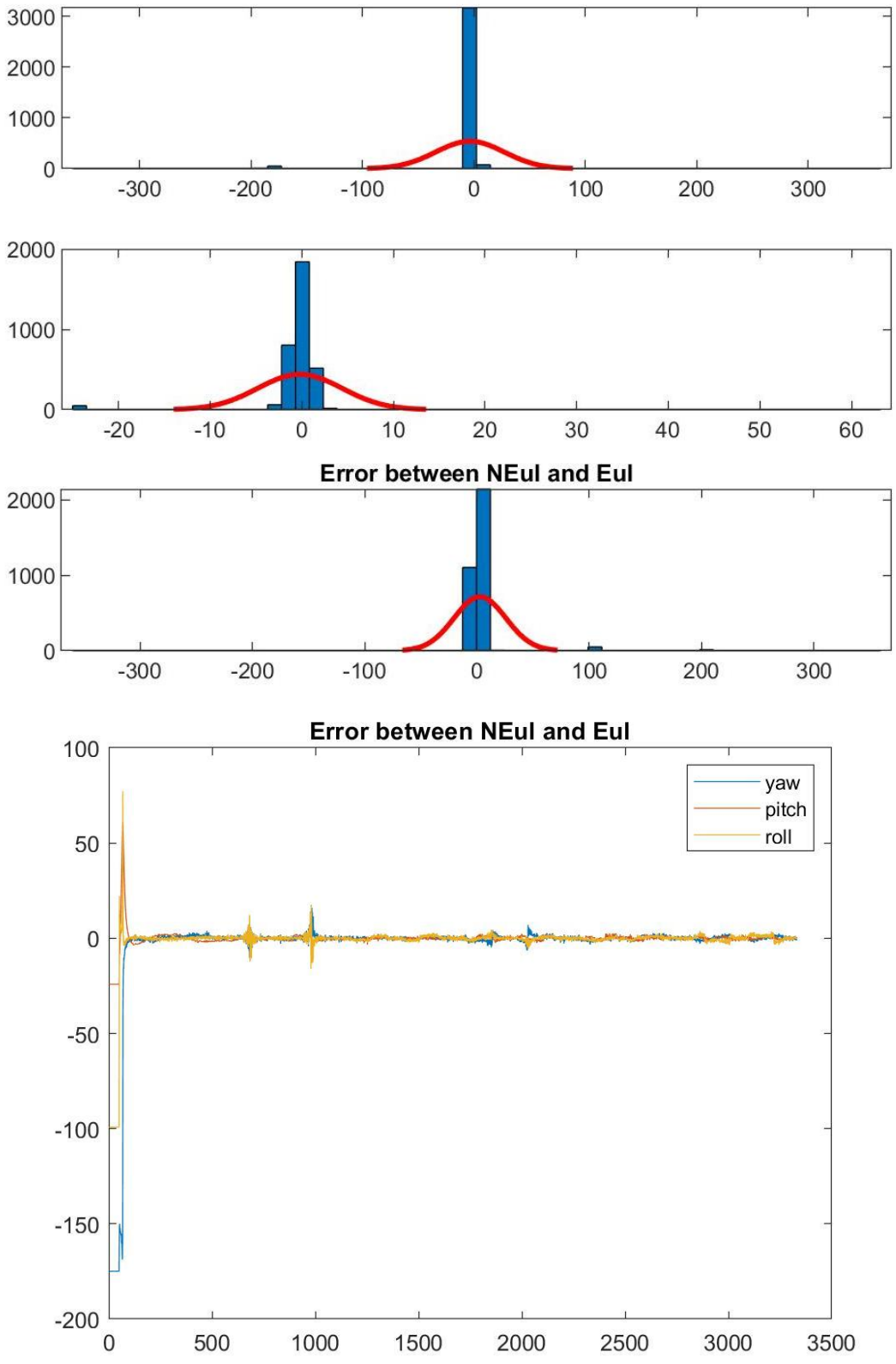


Figure 2: Error comparison between the calculated Euler angle and the expected Euler angle

As you can see in the graph above, the error goes to 0 fairly quick and there are only a few spikes along the way. I would say my part4 works fine.

Part 5: Feedback Using only the accelerometer

First, I need to implement the close loop estimation in C. This is easy since I just need to follow the MATLAB script. The most important part is how to scale the mag and accel reading before feeding them into my algorithm. First, I took the null shift and scale factor that I got (Atilde, Btilde) from Lab3. Then, I multiply my accel and mag reading by my Atilde, then the result is added by Btilde. After all these steps, the mag and accel readings can then be fed into my close loop integration function. Also, since we are only using the accelerometer, which means the Kp_m and Ki_m is set to 0, there's no way I can track my yaw. The algorithm seems to work just fine on my microcontroller.

Part 6 & 7: Misalignment and full complementary attitude estimation

I put these two parts together because part 6 is really short. All I did is to grab the corrected X,Y,Z reading for both accelerometer and magnetometer from Lab3 and feed them to the alignmasterslave.m function provided. This gives me a misalignment matrix that will be used to calibrate the mag reading again. The process goes as follows: first, take the transpose of the misalignment matrix. Then, after the first calibration of my accel and mag readings, multiply the transposed matrix with my calibrated mag reading. Finally, feed the result to my close loop estimation. Since this is full complementary estimation, I set my Kp_m to 0.01(have to keep it small because higher Kp will messed up my reading) and Ki_m to be 0.001. Other than that, everything else is exactly the same as part 5. Everything works just fine in this part except a small drift overtime.

Part 8: TRIAD algorithm

For this part we will only need the mag and accel reading for the attitude estimation. I'm not quite understand the concept of the TRIAD algorithm, but I still rewrite it in C following the MATLAB script. Part 8 is similar to part7, but gyro readings will not be needed, and the DCM will be directly calculated this time. The calibration process of mag and accel readings is the same as part 7 so I won't be going over it again. The result is that I get a better, more stable reading than part 9.

Conclusion:

Even though I got checked off at the end, I'm still not 100% understand everything in this lab. The lab manual is quite confusing at some point and I have to finish the lab based on my assumption. Most of the time I don't even know if I'm doing the right thing or not because the lab manual doesn't specify what the expect output should be. Other than that, it's fun to know

how (parts of) the navigation technique works in real life and I might need to spend some extra time figuring out lab4 completely.