Lab 3

Colin Boblet
Section: 01B

December 24, 2020

1 Introduction

In this lab we will use a MPU9250 to do attitude estimation. The final estimation will be closed loop using the gyros, accelerometer, and magnetometer. We will also implement a TRIAD algorithm to calculate the attitude using only the accelerometer and magnetometer.

2 Orthonormal DCM

The rotation matrix used to adjust the attitude estimate is shown in Equation 4. Using the MATLAB code shown in the appendix, this matrix was verified to be orthonormal. Each row and column has unit length and the transpose is the inverse.

$$[R] = \begin{bmatrix} cos\theta cos\psi & cos\theta cos\psi & -sin\theta \\ sin\phi sin\theta cos\psi - cos\phi sin\psi & sin\phi sin\theta sin\psi + cos\phi cos\psi & sin\phi cos\theta \\ cos\phi sin\theta cos\psi + sin\phi sin\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi & cos\phi cos\theta \end{bmatrix} \tag{1}$$

The angles ψ (yaw), θ (pitch), and ϕ (roll) can be extracted from the matrix using reverse trig functions. The first angle found is pitch using $asin(-sin\theta)$ from the first row. However, we are only able to get 90 to -90 degrees out of this value. This means that there are two possible values for pitch, but either one should result in valid Euler angles. Once we know pitch, it is easy to solve for yaw and roll by dividing out $cos(\theta)$. In the edge case where pitch is 90 or -90 degrees a more complex solution is required. When the pitch is 90 or -90 degrees the device is pointed up or down and $cos(\theta) = 0$, so we cannot divide out $cos(\theta)$. Additionally when the device is pointed up or down the roll component becomes irrelevant. Therefore we can set roll to zero and solve for yaw.

3 Open Loop

3.1 Forward Integration

Once we establish the rotation matrix and can extract the Euler angles, we need a way to update the attitude of the device. Gyros are used to measure the angular velocity, and integrating over time we can extract an angular change. The updated DCM is calculated by taking the cross product of the angular change and the previous DCM and adding this to the previous DCM. By adding is the calculated rotation matrix the DCM should be updated to the new attitude. In reality this does not work as it eventually moves the DCM out of orthonormality.

$$DCM = DCM + [wx] * DCM \tag{2}$$

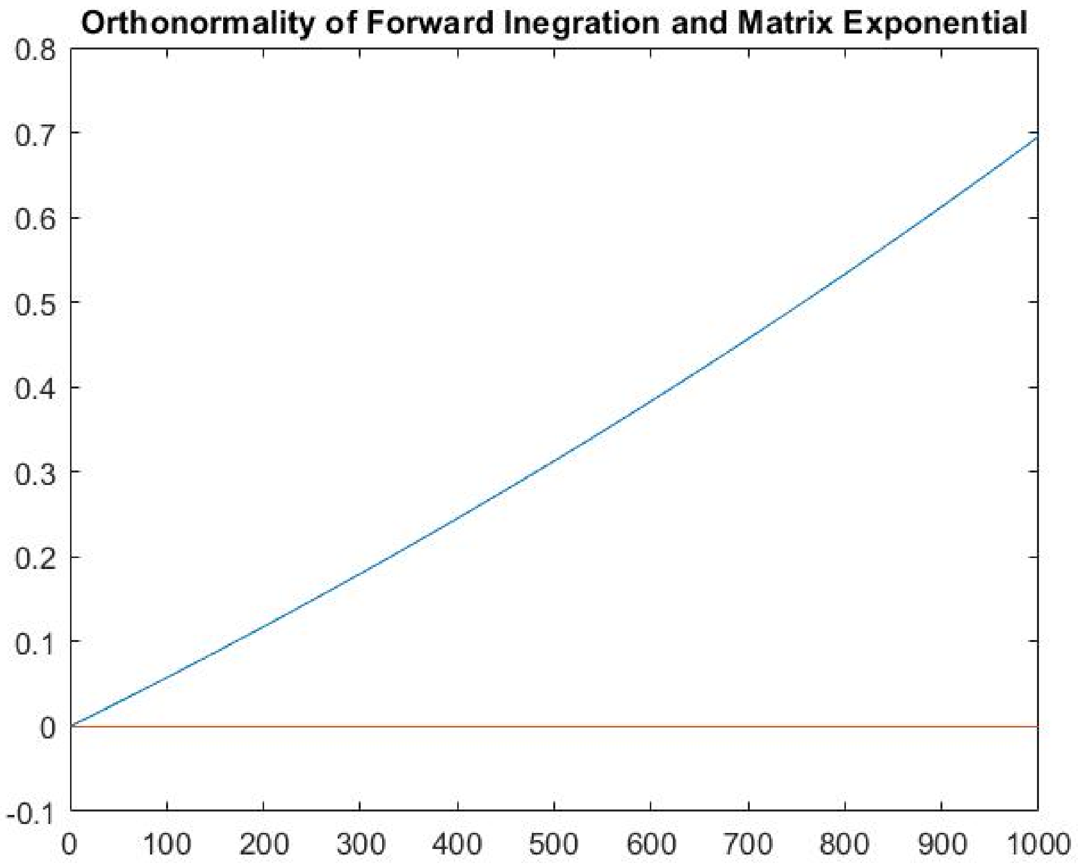

Figure 1: The blue line the is the orthonomality error of forward integration and the red line is the orthonormality error of the matrix exponential using CheckOrthonormality.m

3.2 Matrix Exponential

The issue of orthonomality with forward integration can be solved by using the matrix exponential to update the DCM. The matrix exponential is itself a rotation matrix. So, instead of adding to the DCM we are rotating the DCM into the new frame. The matrix exponential is calculated:

$$R_{exp} = I + sinc(wnorm)cos(wnorm)[wx] + sinc^2(wnorm)[wx]^2/2 \quad (3)$$

The sinc function is estimated in code as $sin(x)/x$, but in the case where x is less than 0.001, the first 3 term of the its Taylor expansion are used.

This matrix exponential is then left multiplied by the previous DCM to give an updated DCM. The orthonormality of the matrix exponential versus forward integration can be seen in Figure 1

4 Closed Loop

4.1 Proportional Gain Feedback

Gyros are not the only sensors on the MPU9250. The accelerometer and magnetometer can be used to implement feedback to correct for errors in the gyro measurements. Errors in the gyro measurements could come from moving too quickly for the gyros to measure or changing speed between time steps if the time steps are slow. Additionally the accelerometer and magnetometer can be used to determine attitude on start up and correct for gyro noise.

The attitude correction is calculated by rotating the inertial frame of a sensor into the body frame using the DCM from the previous time step. This rotation matrix is then compared against the values that the sensor is currently measuring to give a correction to the gyro data. The inertial frame of the accelerometer is based on gravity while the magnetometer inertial frame is based on the Earth's magnetic field.

The correction is then weighted depending on how accuracy of the sensor. In this lab the accelerometer is weighted at 10 while the magnetometer is weighted at 1. This is because the accelerometer is much less prone to noise or calibration issues. However, the accelerometer cannot estimate yaw as the inertial frame is perpendicular to that plane.

4.2 Proportional Integral Gain Feedback

Besides noise and start up attitude feed back can also correct gyro bias and bias drift. This is done similarly to Proportional Gain Feedback. The inertial frame is rotated into the body frame and the difference between the rotated inertial frame and the body frame is the bias error. This error is added to the current bias for correction.

5 Simulated Data

To give an idea of how impactful this feedback can be, several algorithms were used to estimate the attitude of simulated data. The pitch is shown in all figures as it is the easiest to calculate from the DCM. Figure 2 shows the open loop estimation without feedback. The following Figures 3 and 4 show closed loop estimation with only the accelerometer or magnetometer respectively. Figure 5 shows the attitude estimation with both the accelerometer and magnetometer.

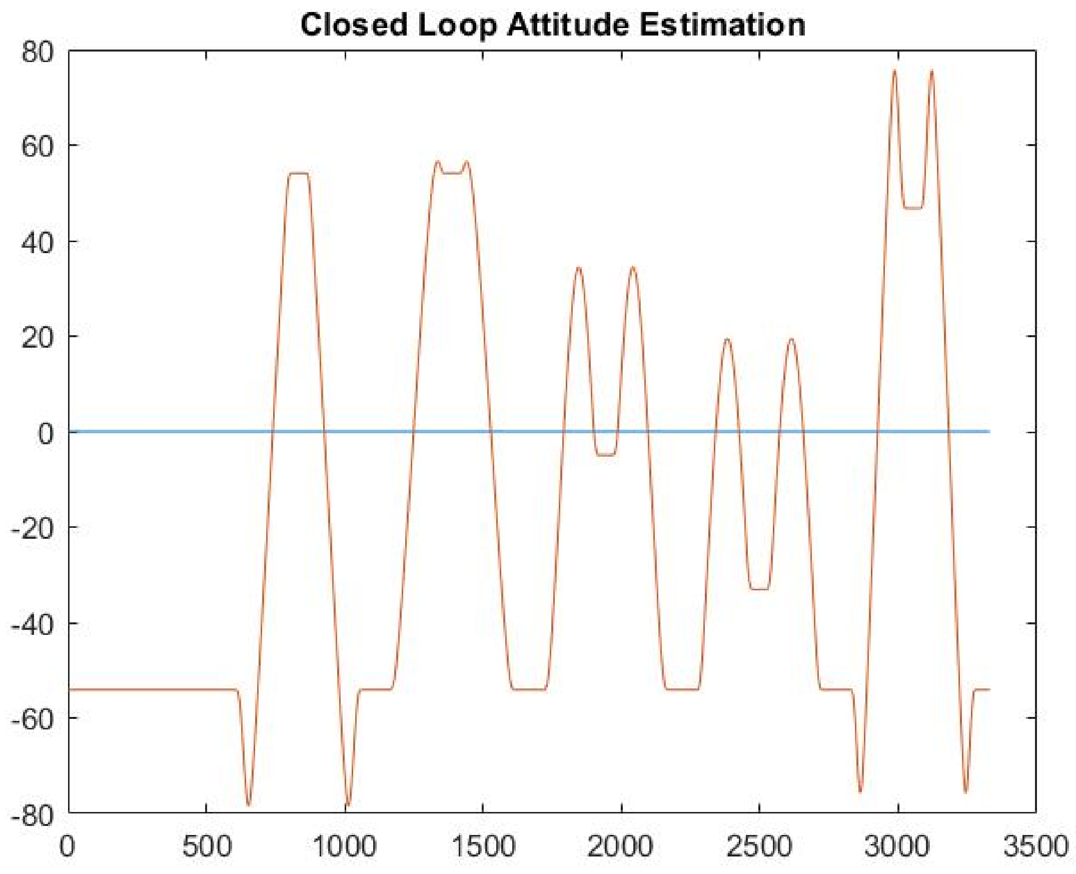

Figure 2: This shows open loop attitude estimation. Data was generated using CreateTrajectory-Data.m.

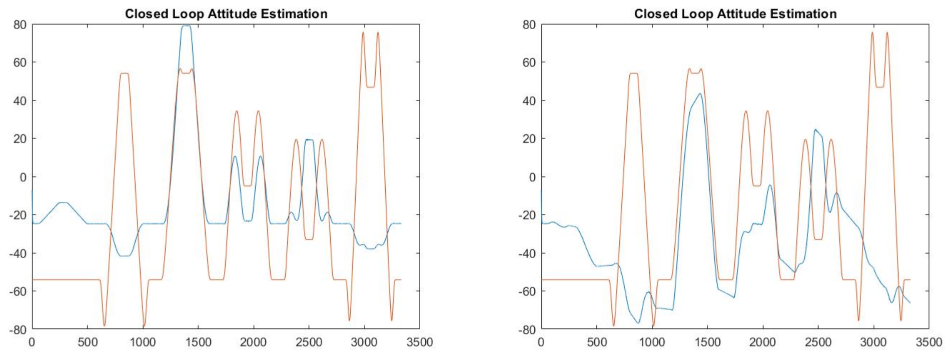

Figure 3: This shows closed loop attitude estimation with only the accelerometer. The figure on the left is without proportional integral gain feedback. Data was generated using CreateTrajectoryData.m.

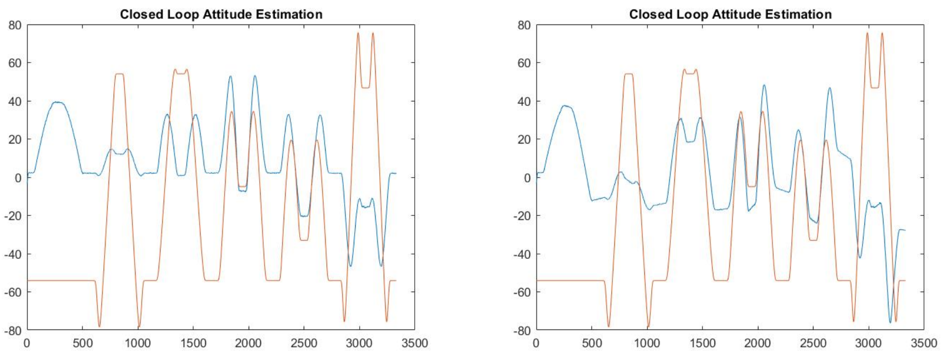

Figure 4: This shows closed loop attitude estimation with only the magnetometer. The figure on the left is without proportional integral gain feedback. Data was generated using CreateTrajectoryData.m.

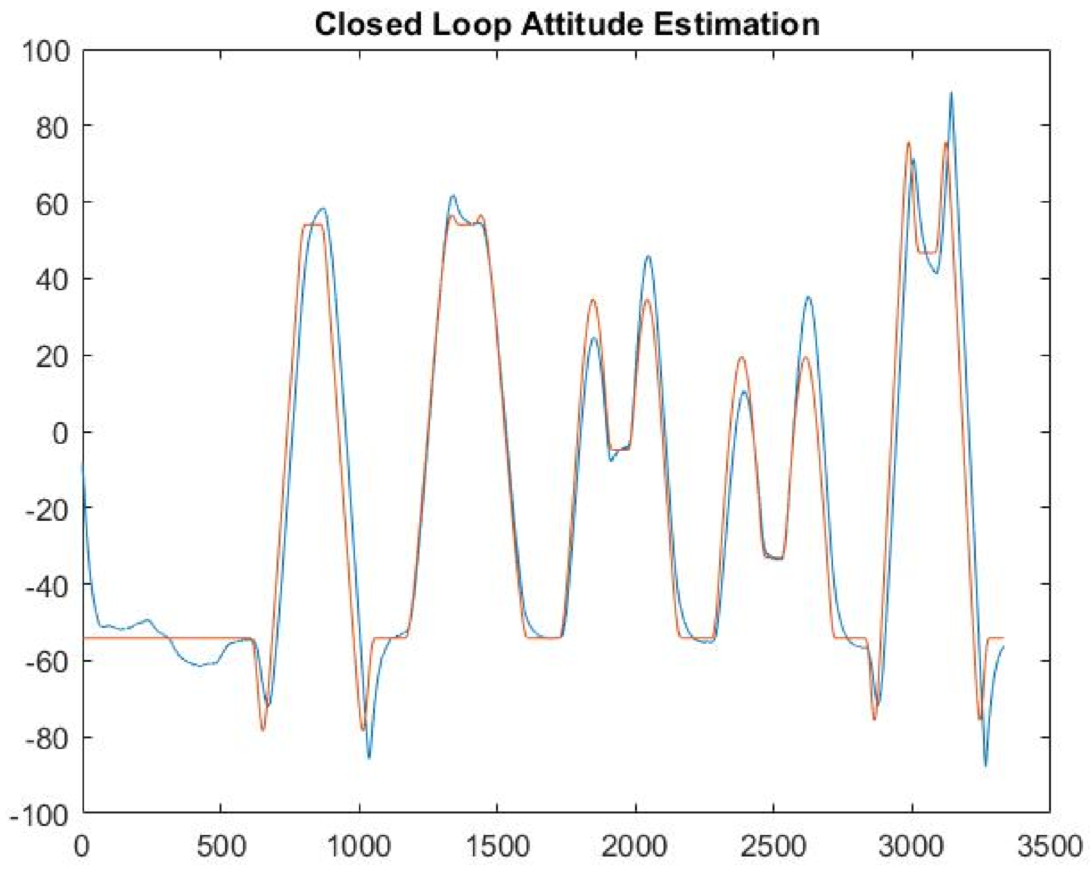

Figure 5: The blue line is the calculated Euler angle using simulated noisy data. The red line the the true Euler angle. The weights are $Kp_a = 15, Kp_m = 15, Ki_a = 1, and Ki_m = 1$.

6 Magnetometer Misalignment

A source of error that can occur in a real-world IMU is sensor misalignment.This occurs when the sensor are physically misaligned. This can be corrected by rotating on senor's data into the other's frame. In this way one sensor is designated as the master and the other is the slave. In this lab the accelerometer is the master and the magnetometer is the slave. Their frames are then calibrated using a tumble test and the provided MATLAB script: alignment.m.

The misalignment matrix was calculated to be:

$$M = \begin{bmatrix} 0.9763 & -0.1644 & 0.1406 \\ 0.2126 & 0.8483 & -0.4849 \\ -0.0396 & 0.5033 & 0.8632 \end{bmatrix} \tag{4}$$

7 TRIAD Algorithm

The TRIAD algorithm is another way of calculating the attitude using only the accelerometer and magnetometer. This algorithm compares the inertial frames of both sensors to the body frames of both sensors to determine an orientation. This algorithm does not need time to settle like the closed loop system. However, the noise of the magnetometer required re-calibration for the attitude to be more accurate.

8 Results

This lab achieved a mixed success. The C code was successfully implemented. The accelerometer and magnetometer feedback values were printed and the resulting DCM was compared against MATLAB calculations. The correction values and DCM were calculated correctly. The TRIAD code was tested similarly. The magnetometer and accelerometer values were printed and compared against the resulting attitude estimation.

However, this did not result in a succesful attitude estimation. I was able to complete a functioning open loop attitude estimator. Closed loop estimation with the accelerometer was able to estimate pitch and roll. However, I have a bug which prevents the yaw from staying below 90 degrees. The yaw would go to about 70 then jump back up to about 100. The closed loop with the magnetometer was able to estimate yaw, but there was significant cross coupling with pitch and roll. I did another tumble test calibration, and the calibration results were significantly different but did not improve attitude pitch and roll estimation. The full complementary attitude estimation was able to estimate yaw, but had slight cross coupling between pitch and roll proportional to the Kp_m and Ki_m values. Additionally, the full complementary estimation inherited the bug from the accelerometer and would not estimate a yaw less than 90. The TRIAD algorithm was able to estimate the full range of yaw accurately, but had significant cross coupling between pitch and roll. The errors in this lab seem to come from the magnetometer calibration

9 Conclusion

In this lab there are two different ways of calculating attitude: Full Complementary Attitude Estimation and TRIAD. The full complementary attitude estimation uses the gyros with feedback from the accelerometer and magnetometer. The TRIAD uses only the accelerometer and magnetometer. Which algorithm is more accurate would depend on the application of the sensor. TRIAD would work well for an application that requires fast attitude estimation in an area with little magnetic field noise. The full complementary attitude estimation is more robust, but would need to be sampled at a much lower frequency.

10 Acknowledgements

This site for rotation matrix angle extraction:

`https://www.gregslabaugh.net/publications/euler.pdf`

11 Appendix

This is MATLAB code for verifying the DCM is orthonormal.

```
% Define variables (y)aw (p)itch and (r)oll and the rotation matrix
syms y p r
R = [cos(p)*cos(y),cos(p)*sin(y),-sin(p);
    sin(r)*sin(p)*cos(y)-cos(r)*sin(y),sin(r)*sin(p)*sin(y) + cos(r)*cos(y), sin(r)*cos(p);
    cos(r)*sin(p)*cos(y) + sin(r)*sin(y), cos(r)*sin(p)*sin(y) - sin(r)*cos(y), cos(r)*cos(p)]

% Take the transpose and confirm that it is the inverse
Ri = transpose(R);
I = simplify(mtimes(R,Ri));

% Calculate the norm of each row and column
normals = zeros(1,6);
for i = 1:3
    normals(1,i) = simplify(R(i,1)^2 + R(i,2)^2 + R(i,3)^2);
    normals(1,i+3) = simplify(R(1,i)^2 + R(2,i)^2 + R(3,i)^2);
end
normals;
```