Leonid Shuster

ECE 167

3/20/20

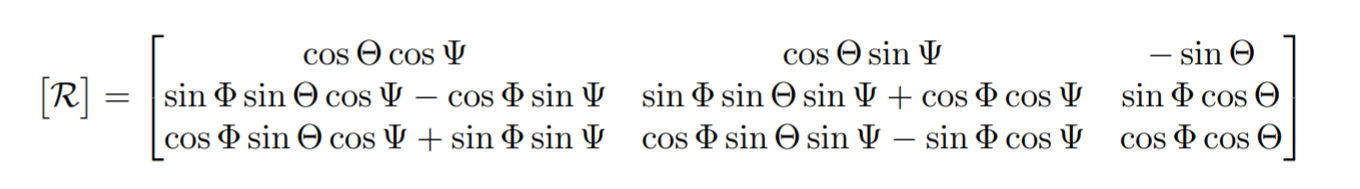
Lab Report 4

**Introduction**

In this lab, we were tasked with learning how to use sensor fusion and integration of three-axis sensors in order to get a more accurate attitude estimate. In the first part, we used the direction cosine matrix (DCM) to solve for the Euler Angles. In the second part, we used open loop integration of the gyroscope in order to track the attitude, while in the third part, we used closed loop integration of the gyroscope and accelerometer. In the fourth part, we aligned the accelerometer and magnetometer axes, and then used both along with the gyroscope in a closed loop integration to find the attitude. And finally, in the fifth part, we used the aligned accelerometer and magnetometer axes directly in the TRIAD algorithm to solve for the attitude.

**Part 1: DCM to Euler Angles**

In the first part of the lab, we were told to figure out the Euler Angles from a direction cosine matrix (DCM). The DCM is an orthonormal matrix that is used to go between a body frame reference and an inertial frame reference, and is represented by the Euler Angles, yaw (Ψ), pitch (Θ), and roll (Φ), in the following way:



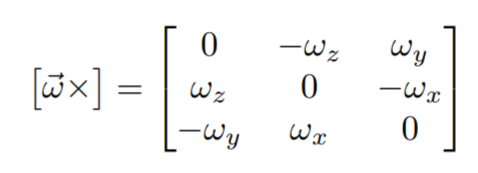
*DCM to Euler Angles (from Lab Manual)*

Therefore, by knowing the DCM and using inverse trigonometry, the Euler Angles could be solved with the following equations:

Knowing this, I then made a program in C that converted a fixed DCM into its Euler Angles which would later be used to determine Euler Angles from live data.

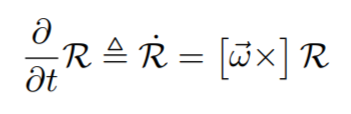
**Part 2: Open Loop Gyro Integration**

In the next part of the lab, we were told to use open loop integration of the gyro to update the DCM and keep track of the Euler Angles. The first equation we had to use was for the skew matrix which keeps track of the gyro rotation rate along each body axis:



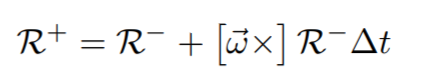
*Skew Matrix (from Lab Manual)*

Where wx is the rotation rate along the x-axis, wy is the rotation rate along the y-axis, and wz is the rotation rate along the z-axis. The other equation we had to use was the DCM differential equation which keeps track of the change in the DCM:



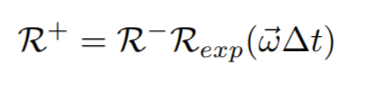
*DCM Differential Equation (from Lab Manual)*

Where R is the current DCM. Using those two equations, two methods could be used to solve for the updated DCM. The first utilized forward integration:



*Open Loop Forward Integration (from Lab Manual)*

Where R- is the current DCM, and is the time step. The problem with forward integration is that you’re adding rotations, so the DCM would slowly drift out of orthonormality. Therefore the second method, which used exponential integration, was much more efficient:



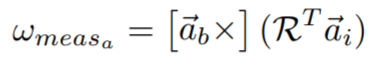
*DCM Open Loop Exponential Integration (from Lab Manual)*

After testing both methods out on MatLab with a fixed rotation rate along each axis, I verified that the forward integration method indeed drifted out of orthonormality slowly over time, while the exponential integration method stayed in orthonormality.

The next step was to try open loop integration with live data. After translating the MatLab code provided into C, I took 10 seconds of gyro data from the sensor in order to remove the bias from the gyro. I then initialized the DCM and continuously read raw values from the gyro, converted them, and fed them into the open loop integration algorithm using exponential integration. I then checked the updated DCM after the integration, compared it with what I got using the same inputs in MatLab, and verified that my algorithm worked.

**Part 3: Closed Loop Integration With Feedback Using Only the Accelerometer**

In the next part of the lab, we were told to use closed loop integration of the gyro with feedback from the accelerometer to estimate attitude. This could be done by comparing what the measured values should be, and correct them by providing the proper feedback using the following equation:



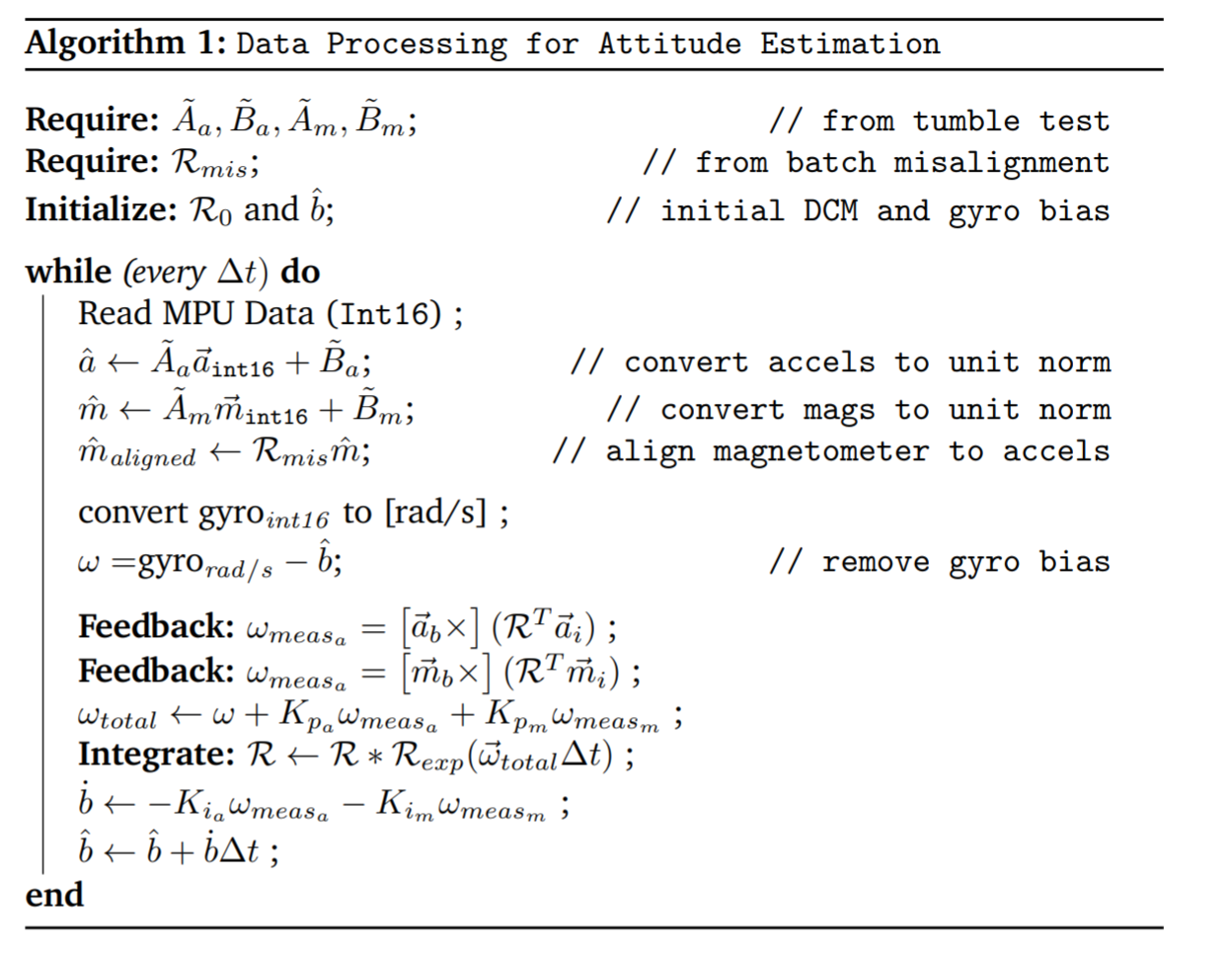
*Closed Loop Correction (from Lab Manual)*

Where ab is in the body frame, and ai is in the inertial frame. This equation, along with updating the gyro bias, would factor into the feedback in the loop. Next, I looked at how the MatLab code for closed loop integration provided worked, and what happened as we changed the proportional and integral gains of the loop. After doing so, I also verified that the feedback from the accelerometer provided values for the pitch and roll, but without the magnetometer, the yaw could not be determined correctly.

Next, I translated the closed loop integration algorithm provided in MatLab into C. After doing so, I once again initialized the DCM, took 10 seconds of gyro data to remove the bias, and continuously measured gyro and accelerometer data and fed them into my closed loop integration algorithm to update the DCM. When I looked at the Euler Angles for the updated DCM, I saw that while my pitch and roll reacted in the right directions, my yaw reading was not correct. I also experimented with the accelerometer proportional and integral gains until I found values that resulted in the most accurate pitch and rolls.

**Part 4: Full Complementary Attitude Estimation**

In the next part of the lab, we had to add magnetometer feedback into our closed loop integration. However, since the accelerometer and magnetometer do not share the same sensor axis, we had to force the magnetometer to the accelerometer’s axis using a misalignment matrix. In order to do so, I took tumble data from the accelerometer and magnetometer. Then, I calibrated these measurements using the MatLab calibration function provided to get Atilde and Btilde for both the accelerometer and magnetometer, and used them in the MatLab misalignment function provided to produce a misalignment matrix that would force the magnetometer into the accelerometer’s frame. After all of this, I then followed the instructions provided at the bottom of the lab manual to align them:



*Aligning Accelerometer and Magnetometer (from Lab Manual)*

With this, the accelerometer and magnetometer values would be calibrated, and the magnetometer would be aligned with the accelerometer. When I ran the program, I measured accelerometer and magnetometer raw values, calibrated and aligned them, and fed them into the closed loop integration algorithm. After modifying the proportional and integral gains for both the accelerometer and magnetometer, I was able to see the yaw, pitch, and roll correspond correctly with the rotations along their proper axes.

**Part 5: TRIAD Algorithm**

In the last part of the lab, we were told to now use the TRIAD algorithm to estimate attitude. Looking at the TRIAD algorithm provided in MatLab, I translated it into C. I then took raw accelerometer and magnetometer measurements and, using the same technique as before of aligning the magnetometer to the accelerometer’s frame, I fed the aligned measurements into the TRIAD algorithm, which now only took in accelerometer and magnetometer values as well as their inertial frames. When I ran the program, I saw that the TRIAD algorithm was not as accurate as the closed loop integration with accelerometer and magnetometer feedback, which I think comes from the fact that there isn’t any feedback, as well as the measurements from the gyro to give a more accurate estimate.

**Conclusion**

By the end of the lab, I could successfully use either the Closed Loop Integration method or the TRIAD method to approximate attitude. I learned that in order to get the most accurate estimation of attitude, a combination of sensors had to be used. The gyro and accelerometer could give pitch and roll readings, but the yaw could not be determined without the magnetometer. I also learned that Closed Loop Integration and the TRIAD are two different methods to measure attitude, with the Closed Loop Integration giving more accurate results. If I were doing this lab again, I would utilize MatLab more for testing the different algorithms to more assuredly understand how they work.