UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE167/L: INTRODUCTION TO MECHATRONICS

LAB 4: ATTITUDE ESTIMATION

PURPOSE:

The purpose of this lab is to get a handle on sensor fusion, and learn to integrate the gyros and other three-axis sensors in order to develop a useable attitude estimation algorithm driven by the actual sensors you have. You will learn how to use the accelerometer and magnetometer to estimate the gyro biases on the fly and to correct the attitude estimate.

HARDWARE PROVIDED AND INTENDED USE:

- I2C module
- Sparkfun IMU module
- Earth's Gravity and Magnetic Field (free of charge)

BACKGROUND:

With the advent of smart phones, the price of integrated low cost MEMs accelerometers, gyros, and magnetometers have dropped to an amazing degree. While these MEMs sensors are certainly not navigation (or even tactical) grade, they can be combined to give very good high bandwidth information about the orientation of the device in space (the attitude or pose of the device).

Like most MEMs sensor, however, these miniaturized devices have a plethora of error sources, including temperature drift, non-linearity, hysteresis, and some cross coupling between channels due to the manufacturing process. While more expensive sensors have much better spec's, these come at vastly higher cost ($100K+), and often with severe restrictions on use due to ITAR[†] restrictions.

In the previous lab you calibrated the scale factor and offset errors for the 3-axis sensors, and got a handle on the gyro bias and drift. In this lab, you will learn to solve the non-linear differential equation in order to track the attitude (in the form of the direction cosine matrix—DCM). Note that this is an excellent example of combining multiple flawed sensors in order to get a better answer. This is known as sensor fusion, and there are many techniques to do this. The specific one you will be implementing in this lab is known as complementary filtering. Other techniques such as Kalman Filtering (KF), Extended Kalman Filtering (EKF), and Muliplicative Update Extended Kalman Filtering (MEKF) are all beyond the scope of this class.

Also note that the 9DOF IMU moniker from both Invensense and Sparkfun is quite misleading. The 9DOF comes from the fact that there are 3 different 3-axis sensors on board (accelerometer, magnetometer, and gyro). Since each is measuring a different phenomenon, then there are 9 different measurements (10 with the temperature) available at any point in time, thus 9DOF. However, these sensors are combine through the process of sensor fusion and attitude estimation to estimate (not measure) the inertial displacement (x,y,z) and attitude (roll, pitch, yaw) of the device. Thus in any real sense, they can be used to get a 6DOF solution (which is all that exists in inertial space). Further note that these specific devices are

[†] ITAR – International Traffic in Arms Regulations, passed in 1976, has severe restrictions on re-exportation of guidance devices. See: https://en.wikipedia.org/wiki/International_Traffic_in_Arms_Regulations

incapable of resolving the attitude to the accuracy required to remove gravity from the accelerometer output. That is to say that the residual error in gravity orientation will swamp the inertial acceleration of the device itself, and thus the inertial position (double integration of the acceleration) will be unusable. Realistically, these devices are used to estimate attitude and not position. Technically, this is called an AHRS (Attitude Heading Reference System), not an IMU (Inertial Measurement Unit).

You are going to implement an AHRS system from the raw outputs of the Sparkfun unit, and you are going to do it in C![‡]

PART 0

There are no hardware changes from the previous lab. Refer to Lab 3 if you need to set it up again.

PART 1: FAMILIARIZATION WITH MATLAB CODE

The instructor went over several things that you will need to implement in MATLAB in order to demonstrate how the attitude direction cosine matrix (DCM) works, and what was needed to integrate the gyros, add the feedback, estimate the biases on the gyros and see how things worked in general.

One of the key things to remember is that the DCM transforms vectors in the *INERTIAL* coordinate frame into the *BODY* frame. That is, if you want to know where the "nose" of the object is pointing in the INERTIAL frame, then you would calculate that as $\frac{\hat{e}_x}{I}=[R]^T\begin{bmatrix}1\\0\\0\end{bmatrix}$. That is, take the *transpose* of the DCM, *[R]*^{T}, to get the *BODY* to *INERTIAL* frame transformation. It is useful to keep track of what coordinate frame you are going from and which one you are going to.[§]

Spend some quality time with the MATLAB scripts that the instructor has made available to you. Understand what is going on inside them. You will be implementing these yourself in C on the micro, so you should be intimately familiar with how they work.

PART 2: CONVERSION FROM DCM TO EULER ANGLES

Given that you cannot plot 3D coordinates on your micro very easily, you will need to be able to display something on the OLED to see what you are doing and if it makes sense. These are going to be the three Euler angles (from the 3-2-1 set) in degrees. These are defined to be: Ψ (yaw), Θ (pitch), Φ (roll). The conventions are that $+\Psi$ is a rotation from North towards East, $+\Theta$ is a rotation about the new body y-axis and is nose up, and $+\Phi$ is a rotation about the new (new) body x-axis and is right side down.

$$\begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi-\cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi+\cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}$$

You have been given in class the matrix form of the DCM from Euler Angles (above). You are going to write C code to extract the Euler angles from the DCM (by picking parts from the matrix and doing inverse trig on them). Note that you will need to get to degrees in order to make intuitive sense.

[‡] Welcome to the big leagues. While the math behind attitude estimation is usually not taught until graduate level classes, we are going to have you implement an algorithm to do it (which we provide you with). Many of you will go on to do capstone design classes or other projects where attitude estimation is required, this is going to be something useful to you in the future, so pay attention.

[§] The reason you want to keep switching coordinate frames is that some things are very easily measured in the body frame (e.g.: the actual sensors) but other things are more easily measured or known in the inertial frame (e.g.: gravity). Often you will go through several different frames to solve a problem.

Also, demonstrate that the above matrix is, in fact, orthonormal (each row and column has unit length, is perpendicular to the others, and that the transpose is the inverse). Include this in your lab report.

Make yourself a C function that you can call that returns the angles for a given rotation matrix. Be careful to define how you are going to keep the 3x3 matrix.

PART 2: INTEGRATION OF CONSTANT BODY-FIXED RATES

In order to understand how the open loop integration of the DCM works, first play with the MATLAB code that does the open loop integration. Note that in the code that the instructor showed you in class, the time step is implicitly set to 1 second. You will need to include a Δt in your calculations.

What you are going to do is to set up the C code to do the integration given constant inputs on [p,q,r] and see that the DCM evolves in a way that makes sense to you. It is pretty key here to familiarize yourself with the MATLAB code to see how things are being updated and to really understand the visualizations. You are going to do this in two ways: (1) using the simple forward integration: $R^{+} = R^{-} + \dot{R}\Delta T$ and (2) using the matrix exponential form. See how fast your DCM drifts out of orthonormality using both methods.[**]

Note that you need a *sinc* function for the matrix exponential. Sin(x)/x is not going to cut it because it will give a divide by zero error when you have no rotation. You are going to have to do this on your own, an easy way to do this is the first few terms of a Taylor expansion near zero and switch to sin(x)/x farther away.[††]

PART 4: OPEN LOOP GYRO INTEGRATION

In order to track the attitude as you move the body around, you are going to integrate the gyros using the code that you developed in Part 3. You will also display the attitude (in the form of the Euler angles on the OLED) in real time. You should also save the gyro data (and the mag and accel data) so that you can run it through the integration on MATLAB in order to check your results.

Before beginning the integration, remember to set the sensor on the bench and take 10 seconds of data or so in order to remove the bias from the gyros.

Implement this using the matrix exponential form (and remember that you are taking the matrix exponential of [p,q,r]*ΔT).

PART 5: FEEDBACK USING ONLY THE ACCELEROMETER

The open loop integration works well (and is what you use for the high bandwidth attitude tracking). However, it relies on the gyros having no bias (and no bias drift) and having the initial attitude correct. Note that you could get that pitch and roll parts of the DCM from the accelerometer measurements to initialize the DCM, but that you can only get yaw with the magnetometer.

In order to drive the solution into the correct DCM even with biases on the gyros and a wrong DCM to begin with, you are going to implement the closed loop attitude estimation using the accelerometer feedback. Note that you will be estimating the gyro biases and the DCM. The feedback is going to come in as an alteration of the [p,q,r] that you send into the matrix exponential function. Play with the gains, Kp and Ki and see if you can find the bounds of stability and where it works well.

[**] Remember that orthonormality is defined that the DCM inverse is equal to its transpose, that is: $[R]^{-1} = [R]^{T}$. Another way of stating that is that $[R][R]^{T} = I_{3x3}$. You should test this and see how bad it gets as you proceed. There are techniques to re-orthonormalize a matrix, but they are beyond the scope of this lab.

[††] Yes, we expect you to derive this and to show your work in the lab report, as well as the thresholds where you switch from the approximation to sin(x)/x.

You should note that using the accelerometer feedback means that you cannot fix the yaw since you have no information about it.

Part 6: Misalignment of Accelerometer and Magnetometer

We assume that the individual sensing elements are aligned well at the time of manufacture and that they share a consistent sensor axis. This might not be the case. Using the same tumble data, we can extract the misalignment of one sensor relative to the other.[‡‡]

This is going to be done using some fancy linear algebra techniques that you don't really need to understand (see Prof. 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 the misalignment 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. You will need the true magnetometer reading in inertial coordinates (from NOAA site). Or you can get it by setting the magnetometer carefully level and pointed north and averaging the measurements over a decent time.

Run the misalignment code in MATLAB and generate the misalignment matrix. Use this to rotate your magnetometer measurements into the accelerometer sensor frame. Does this make sense to you?

Part 7: Full Complementary Attitude Estimation Using Mag and Accel Feedback

For the last part of the lab, you are now going to add in the magnetometer into the feedback. Thus you are going to have two Kp's and two Ki's, one for the accelerometer and one for the magnetometer.

This is using the "Full Monty" of calibration that you have available to you. The spherical calibration you did for your mags and accels, the bias removal of the gyro, and the misalignment calibration of the magnetometer to the accelerometer frame.

Output the results of your attitude estimate to the OLED in Euler Angles, and save them (along with the raw data) if you can. See if it all makes sense to you.

Congratulations: this is not an easy thing to do, and many many people who attempt this get the math completely wrong. Keep this one in the "I know how to do this" file for later use during your careers.

Part 8: Confirmation of Gyro Integration using Wahba's Problem Solution

Now that you have the calibrated and corrected magnetometer and accelerometer measurements (using the spherical calibration you did last lab) and the misalignment matrix you just computed above, it is possible to calculate the DCM directly as a solution to Wahba's problem, given these two vector measurements.

You can do this using the Markley SVD solution (in MATLAB) or you can do this much more simply using the TRIAD algorithm (which is computationally very easy). In order to get TRIAD to work, you will need to be working with unit vectors (length of 1).

[‡‡] Being pedantic, you are actually extracting the relative rotation of each sensors' *coordinate frame* relative to the other. The algorithms you are using work for both small and very large misalignments. This is useful if you have some ambiguity about the sensor axes from the datasheet.

See https://en.wikipedia.org/wiki/Triad_method for the TRIAD method (especially Eq. 1-9). Be a little careful in that the $\hat{A}$ that comes from equation 9 might be the transpose of the DCM or might be the DCM itself (the code we give you is correct).

Compare the Wahba's solution (either from the Markley SVD or from the TRIAD) to what you got from the gyro integration. You should be close. See if you can plot the errors and make some comments about what you see.