Lab #3: 3D Sensor Calibration

ECE-167/L: Sensing and Sensor Technology

University of California Santa Cruz

Purpose

The purpose of this lab is to familiarize you with a modern 9DOF IMU sensor and the errors in the sensing elements. You will learn to communicate with the IMU, understand the error sources, and learn to apply modern linear algebra techniques to calibrate the errors out of the sensor. You will learn about bias drift, and the basics of gyro integration to get an angle from the raw rate sensor.

Hardware provided and intended use

Uno32, Sparkfun IMU Module, Earth's Gravity and Magnetic Field (free of charge)

1 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[1] restrictions.

In this lab, we will ignore all of these error sources and focus merely on scale factor and bias errors for the sensors, and methods to measure and calibrate out these biases. We will also take a look at gyro bias drift, and use naïve methods to remove it and validate the effectiveness of such a method. Note that there is every reason to believe that in these low-cost sensors the bias and scale factor errors are a function of temperature (and most likely other factors as well). The Sparkfun sensor board has an onboard temperature sensor for exactly this reason, however recalibrating the sensors over a temperature range is difficult, and requires extensive equipment unavailable in the lab.

[1]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

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 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 completely 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).

2 Hardware Setup, Materials Needed, and Configuration

To get started, there are a couple of things to physically change on the uno32 microcontroller. First, there are some jumpers (the blue-block connectors) under the IO-shield. The IO-shield must be carefully removed (the IO-shield is easy to break, ask the TAs if you haven't done this before). The jumpers for **JP6** and **JP8** on the micro (not the IO-shield) must be shifted such that jumpers are closest to the labels RG3 and RG2, respectively. This means that the pins for A4 and A5 will be exposed. Next, the IO-shield needs to be put back on. Female-to-male jumper wires should be connected to the I2C pins of port J11 on the IO-shield, one for GND, 3V3, SDA (data), and SCL (clock). Make sure the wires are all different colors so that you mitigate the possibility of connecting power from the micro to ground on the accelerometer. Finally, solder up the accelerometer and carefully connect the aforementioned I2C pins from the micro to the accelerometer's corresponding pins. To configure the accelerometer, use the MPU9250 library, I2C library, and their subsequent functions.

3 Ellipsoid Calibration using Simulated Data

In order to understand how we will be calibrating the 3-axis sensors, it is useful to understand the *geometry* of the problem. It is instructive to do this in 2D first. Begin with a hypothetical 2D sensor that measures the x and y components of some vector that is fixed. As you rotate the sensor, the sensing axes of the sensor measure the components of the fixed vector. If you plot the points from the sensor there will be a circle of radius equal to the length of the vector.

However, the sensor is imperfect. There is a null shift on each of its axes, and the scale factors are different on both axes as well. Now if you rotate the sensor and plot out the points, you will get an ellipse that is centered on the biases, and whose semi-major and semi-minor axes lengths correspond to the scale factors. Note that if there were cross-sensitivity in the axes, it would be rotated as well. And lastly, there is wideband noise on the measurements.[2]

The task (finally) is given the noisy points on the ellipse; find the scale factor, cross-coupling, and null shifts such that you can reconstruct the circle centered at the origin with radius corresponding to your vector. And, of course, you don't get the complete ellipse, but only part of it.

[2]For most sensors, the wideband noise is modelled as an additive Gaussian white-noise to the sensed signal. This is a simplification, as the noise will become correlated over a long enough time period. For those of you interested in this phenomenon, read up on the Allan Variance (https://en.wikipedia.org/wiki/Allan_variance).

Unfortunately, you cannot do this with simple least squares. This is because the problem is not linear in the coefficients that you want. It is, however, linear in algebraic combinations of the coefficients that you want. So you do this in two steps: (1) Solve the LS for the funny quantities, and (2) Extract the desired coefficients from the solution to (1) algebraically. It sounds worse than it is.

Begin with the equation for an ellipse: $\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = R^2$. You know R (the length of the vector you are measuring), and you have lots of (noisy) x,y pairs. To get a perfect circle centered at the origin back, you want to remove the bias and scale factor from each x,y pair. If you have a $\hat{x} = \frac{1}{a^2}(x - x_0)$ and a corresponding $\hat{y}$, then $\hat{x}^2 + \hat{y}^2 = R^2$, and you have the circle.

Finally for the lab:

1. Expand the ellipse equation (ignoring the cross-coupling)
2. Write it out for a series of $(x, y)_i$ pairs
3. Gather the algebraic combinations of parameters into a vector, leave the x_i and y_i terms in a matrix.
4. Assume that you have the parameter vector estimate (from Least Squares) how would you get estimates of x0, y0, a, and b (you know R).
5. Download the EllipseXYData.m from the CANVAS and do the least squares, and solve for the parameters, and x0,y0, a and b (R is defined in the file)
6. Plot the 2-norm of the pre-calibrated and post-calibrated data.[3]

This is simply a simulation exercise to get you familiar with the problem. The specific technique you are doing is called a “two-step” estimation, and while it works well, it has some issues that we will address further on.

4 Naive Calibration of Magnetometer and Accelerometer

Now that you have a sense of how the ellipsoidal calibration works for 2D data, you can easily imagine extending the algorithm to 3D (and perhaps how messy the math will be). However, it would get more complicated if we did not restrict the off diagonal terms of the scaling matrix to zero (cross-sensitivity). Before we get to applying that to the sensors you have, we will first explore some more naïve alternatives to the ellipsoidal calibration techniques.

You have already done a simple two-point calibration before in the lab (generating a scale factor and offset for a single sensor). You are going to do the same for each axis individually. It is a bit easier to do this with the accelerometer, because gravity is straight down and most of our tables are level to the ground.

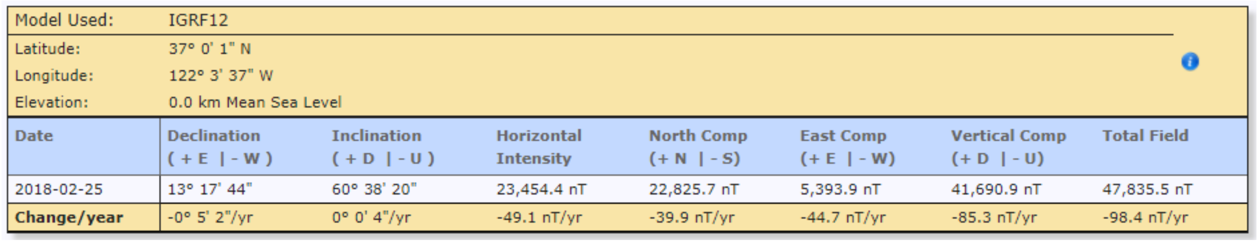
The accelerometer measures *specific force*, not acceleration. That is: $\vec{a}_m = \frac{\vec{f}}{m} = (\vec{a}_I - \vec{g})$. If you put an accelerometer on a table, it measures –g. If you measure it while it is falling, it will read zero until impact. You are going to use this to your advantage. First, calibrate the z-axis of the accelerometer by measuring the value with the sensor flat on the table, then flip the sensor upside down and measure again. These points should correspond to +/- 1g. Note, to minimize the effect of noise on your calibration, take at least 100-1000 samples at 50Hz in each orientation and average them to improve SNR. With those two points, you should be able to extract both scale factor and null shift for the z-axis of the accelerometer.

Repeat the experiment for the x- and y-axes. Note that this means you will have to hold the accelerometer at 90 degrees to your bench and flip it over. Make yourselves a jig from foamcore or anything else to help you maintain the alignment. Record your values for null shifts and offsets for each axis.

[3]The 2-norm is the Euclidean length of a vector, for a 2D vector it is $\sqrt{x^2 + y^2}$, for a nD vector it is the square root of the sum of the squares of the individual components; this can be denoted as $\sqrt{v^T v}$, where T is the transpose operator.

The magnetometer can also be calibrated in a similar fashion. The thing that makes this harder is that the magnetic field is not vertically down. Instead, in Santa Cruz, the magnetic field is 13 degrees east of north and 60 degrees down towards vertical (you can get the magnetic field at any point on the planet from the NOAA website: https://www.ngdc.noaa.gov/geomag-web/?model=igrf#igrfwmm). Note that the units are given in nT (10^{-9} Tesla). A more common unit is Gauss (1G = 100000nT). Thus the total field in Santa Cruz is 0.4784G.

There are two ways to do the naive calibration on the magnetometer axes: (1) Rotate the sensor in the horizontal plane until the x-axis peaks and the y-axis reads very close to zero (this should correspond to the point of "Horizontal Intensity" in the image below (or verify it yourself using simple vector math). Mark that direction on your table, and use that to set your value. This will allow you to do both horizontal axes. Analogously you can use the "Vertical Comp" to do the z-axis. (2) The second way is to try to match the magnetic field vector itself, and tilt and swing the magnetometer until you max out the x-axis while having zero on both the y- and z-axes. You will need to use jigs and ramps to ensure that you can consistently reach this point, and then proceed as you did with the accelerometer.



Model Used:	IGRF12
Latitude:	37° 0' 1" N
Longitude:	122° 3' 37" W
Elevation:	0.0 km Mean Sea Level

Date	Declination (+ E | - W)	Inclination (+ D | - U)	Horizontal Intensity	North Comp (+ N | - S)	East Comp (+ E | - W)	Vertical Comp (+ D | - U)	Total Field
2018-02-25	13° 17' 44"	60° 38' 20"	23,454.4 nT	22,825.7 nT	5,393.9 nT	41,690.9 nT	47,835.5 nT
Change/year	-0° 5' 2"/yr	0° 0' 4"/yr	-49.1 nT/yr	-39.9 nT/yr	-44.7 nT/yr	-85.3 nT/yr	-98.4 nT/yr

Figure 1: Output of IGRF model for Earth's Magnetic Field

5 Gyro Bias and Bias Drift

In order to generate a high bandwidth attitude estimate, you are going to need gyros as well as the accelerometers and magnetometers. We will (in the next lab) have you do an open-loop integration of the gyros and use the accelerometers and magnetometers as an aiding reference to feedback into the solution and keep it consistent.

Gyros (or gyroscopes) measure a *body-fixed* rotation rate. Very sensitive ones measure Earth's spin rate and can be used to determine your latitude (think about how for a minute). There are several technologies used to make gyros: spinning iron wheels, wine-glass ring oscillators, laser-ring, fiber optic, vibrating beam structures, and various MEMs technologies. The vary considerable in price ($5 - $5M per axis) and performance. All will have an upper rotation rate at which they saturate. Many have a lower limit below which they cannot detect a rotation rate.

One of the principle difficulties in using the gyros is their bias drift. That is, when you place a gyro on the table and turn it on, it does not read 0°/s. This null shift is the bias of the gyro (and since you will be integrating the angular rate to get to angle, the constant offset will cause a linear error term in angle). To complicate things further, the bias is a function of temperature and also time (that is, it is not stable), but can be assumed to be slowly varying. Cheaper gyros (i.e.: the ones we will be using) have larger biases, and the drift rate is faster. Very (very) expensive gyros have very small biases and very low drift rates.[4]

[4]This is one of those areas where the more you can pay, the better performance you get. The range of prices is breathtaking. When you get into the military or space grade gyros, you are often talking several million $ per axis.

Set up your sensor on the bench, and take 10 seconds of gyro data on each axis. Wait at least a few seconds after the power has come up to ensure that it is fully "warmed up" before you do this. Convert the raw reading to angular rate, and note that it is NOT zero.

Average this 10 second reading to get an initial estimate of the bias for each axis; since the sensor is not rotating, you are reading the bias term directly. Set up your sensor to log data for an hour, and then go back and subtract your biases from each axis. Plot the bias over time. Integrate the angles and plot the drift in °/hour to see how well you did (a medium performance Fiber Optic Gyro [FOG, $3K] will be $1^\circ - 5^\circ$/hour).

6 Gyro Scale Factor via Angle Integration

The proper way to get scale factors for a gyro is to use something called a "rate table" which will spin the sensor at a constant rate on each axis and still allow you to get the data out. They are expensive pieces of equipment and we don't have one.[5] Instead, we are going to calibrate the scale factor on the gyros by integrating the rotation rate through a constant set of angles.

Begin by setting up your sensor and taking 10 seconds of data to establish your biases on each axis. Next, set up a set of stops on your bench that you can rotate through a defined angle (180 degrees works well). Rotate the sensor so that each axis (sequentially, not simultaneously) rotates through the angle, and then back. Integrate the gyro rates (subtracting out your bias from each raw measurement) and see how close to the correct angle you got. Adjust your scale factors until you get it. Experiment with doing this at slower and faster speeds. See if your scale factors change with speed.

HINT: take a 10 second bias reading AFTER your rotation experiment and see if it matches. If it does not, average the before and after.

7 Tumble Test Simulation

In order to get a feel for what you are doing and what you expect to see (and it helps find the bugs in your code), we are going to have you run the "tumble test" algorithms using simulated data.

On the CANVAS webpage, download the MATLAB files: `CreateRandomAttitude.m` and `CreateTumbleData.m`. If you inspect the file, you can see that it not only creates the tumble data, but also adds noise, and distorts the measurements with scale factor, biases, and cross-coupling. You can edit the noise parameters if you wish to generate noise-free data. Each time it is run, it will generate a new distortion matrix. If you wish to segregate data into training and validation sets, do so by splitting the output data into two sections. Note that in addition to generating the accelerometer and magnetometer data *as you would see them, in ints*, it also generates the distortion matrix (Adist) and bias vector (Bdist). The function takes the number of points of tumble data you wish to create (e.g.: `[A,M,Ad,Bd]=CreateTumbleData(1000)` creates 1000 points of data).

Naively, you could plot the data, and by looking at estimate the scale factors and bias for each axis. Try it and see what you get for scale factor and bias doing this (think about what you will see looking at the X-Z data plot). For the accelerometer, it should scale to $\pm 1g$. For the magnetometer, you would also like to scale it to ± 1[Earth's Magnetic Field].

[5]Indeed, the best ones allow you to set up rotations on all three axes simultaneously, and do so in a temperature controlled environment. Thus you can simulate the full trajectory including the temperature variations from climbing and descending. Needless to say these are very very expensive, and mostly big defense contractors are the only ones who own them.

Once you have done this naive calibration, apply the scale factors and biases to the data, and plot the norm of magnetometer data and the calculate the standard deviation of the norm. You might find the MATLAB command `histfit` useful here.

Now, forgetting the naive calibration, you are going to perform the full LS ellipsoidal calibration using the simulated data. There are a number of steps required to do this. The first step is to re-scale the integer data to units of g's and units of Earth's Total Magnetic Field. That is, the range of the post-scaled data should be ± 1 for both the accelerometer and the magnetometer. You will need to look on the data sheets and in the code headers to find the appropriate scale factors. Note that the units of Earth's magnetic field that is on the datasheet is μT, not nT.

Run the scaled data through the `CalibrateEllipsoidData3D.m` function. This will generate the $\tilde{A}$ matrix and $\tilde{B}$ vector. If you recorded the distortion matrix (Adist) when you generated the data, then $A_{dist} * \tilde{A} - I_{3\times 3}$ should be small. It won't be entirely zero because of noise on the measurements, but it should have small numbers in every entry.

Correct the data by running the `CorrectEllipsoidData3D.m` function (using the $\tilde{A}$ matrix and $\tilde{B}$ vector you just got out of the `CalibrateEllipsoidData3D.m` function.

Once again, plot the pre-calibration and post-calibration norm and calculate means and standard deviations for both. Compare these to the naive calibration results on the simulated data you just did. In the lab report, include notes about what you observed, how closely you matched the actual distortion matrix, and a table of means and standard deviations for both sensors using the naive and ellipsoid calibration methods.

8 Tumble Test for Accelerometer and Magnetometer

In Part 3 we asked you to derive and apply an ellipsoidal calibration for a hypothetical 2D sensor. In Part 7 we had you do the 3D ellipsoidal calibration with simulated data. Now we are going to do it for real with our 3D sensors (but we won't make you derive the least squares solution). In order to do this, you are going to "tumble" the sensor in your hands and collect data. Remember, each data point is a point on the surface of the ellipsoid that you are going to be converting to a sphere. Try to get good coverage over all of the ellipsoid.

Do this by moving smoothly and rotating the device. Avoid sudden movements or shocks, and do it slowly. Collect data continuously throughout this process, and it is OK to revisit the same points. Think of this as painting the surface of a sphere with a pointer attached to your sensor. You want a lot of data, expect to spend a few minutes tumbling. Ideally, you want two sets of tumbling data (a training set and a validation set).

Once you have the data collected, scale the data to engineering units, and plot the norm of both the accels and mags vs. time. Take the mean and standard deviations of the norm to see how good the "raw" sensors are. Now calibrate your sensor reading using the null shifts and scale factors you got from Part 4, and plot the norm again.

Lastly, run your data through the MATLAB code as in Part 4, which will output the scale factors and biases for each sensor. There is also a function to output the calibrated sensor data with those parameters. Again, plot the pre- and post-calibration norm of each.

Make a table of means and standard deviations for the norm of the tumble test with the differing methods of calibration. If you have a second tumble set, redo the plots (but only use the parameters generated from the training set). How did the results change?[6]

Congratulations, you have calibrated your "IMU" sensor. In order to use the sensor in the future, you will take the data you collect from the sensor at each measurement, scale it to engineering units, and apply your ellipsoidal calibration derived from your tumble test. This will then give you a calibrated unit vector measurement for gravity and magnetic field. This will be used in the next lab when you will do complete attitude estimation.

We want a report of your methodology and results, such that any student who has taken this class would be able to reproduce your results (albeit with some effort). If there are any shortcomings in this lab manual, please bring it to our attention so that we may improve it for the next class.

[6]This is known as a validation set of data (that is not used in the calibration process). If the calibration works on the validation set, then you have confidence that the parameters are the correct ones and not overtuned.

A Logging of IMU data

Logging data can be achieved a few different ways, but one in particular is recommended for this lab. Using putty, under "Session" -> "logging" select Printable output, change the Log file name as desired (you can also have it autogenerate names based off time or date), and select a location on your computer to save the log file. Also, under "logging" select the serial button and change the com port as necessary. Make sure the Speed (Baud Rate) is 115200 as usual. Putty allows you to save these settings under "Saved Sessions" for late use. The basic procedure for logging data is to have your program reading the accelerometer data, then printing to serial, and finally using putty to log the serial output to a file. Make sure that the format in which you print to serial is acceptable for MATLAB to parse or for you to copy and paste the material of the file simply without having to delete non-data material. For instance, if you printed your data to serial with **printf("z accel data: %d\r\n", zAccel)** you would have to get rid of "z accel data:" for every data point measured, because that is useless information in MATLAB unless you want to go through the toil of parsing files in MATLAB (which you are welcome to do). Ideally, you would just do this: **printf("%d\r\n", zAccel)** just printing the number and a new line (carriage return as needed). Depending on how you're logging data, you can still print multiple data points at a time; just make sure you can parse it easily in MATLAB.

Acknowledgements

- Figure 1 GABE, where did this come from?