Lab #3: 3D Sensor Calibration

ECE-167: Sensing and Sensor Technology
University of California Santa Cruz

The purpose of this lab is to familiarize you with a modern 9-DOF (degrees of freedom) 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 linear algebra techniques to calibrate the errors out of the sensor. You will learn about bias drift, and the basics of gyroscope integration to get an angle from the raw rate sensor.

With the advent of smartphones, the price of integrated low cost MEMs accelerometers, gyroscopes, and magnetometers have dropped to an amazing degree. Like most MEMs sensors, 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 specs, 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 most of these error sources and focus merely on scale factor and bias errors for the sensors, and methods to measure and calibrate out these sources of error. We will also take a look at gyroscope bias drift, and use naïve methods to remove it and validate the effectiveness of such a method. In these low-cost sensors the bias and scale factor errors are a function of temperature (among other things). The sensor board has an onboard temperature sensor for exactly this reason, however recalibrating the sensors over a broad range of temperatures is difficult, and requires equipment unavailable in the lab.

It is worth noting that the 9-DOF IMU moniker is quite misleading. The 9-DOF comes from the fact that there are 3 different 3-axis sensors on board (accelerometer, magnetometer, and gyroscope). Since each is measuring a different phenomenon, then there are 9 different measurements (10 with the temperature) available at any point in time, hence 9-DOF. However, these sensors are combined 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. These values are actually used to get a 6-DOF solution (which is all that exists in inertial space). Further note that 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).

Hardware, Materials and Configuration

You will be using the STM32 and the BNO055 IMU module. This is a cheap, consumer grade IMU. The main metric used in assessing the performance of an IMU is the gyro bias instability, or drift, which is typically measured in degrees per hour. The higher this value, the less useful the IMU is for complex tasks

[1] ITAR – International Traffic in Arms Regulations have severe restrictions on sale and export of guidance devices.

like mapping, localization, and navigation. Cheap MEMs devices like the BNO055 have very large drift values and therefore cannot be used in complex applications. But calibrating these cheap IMUs follows the same procedure.

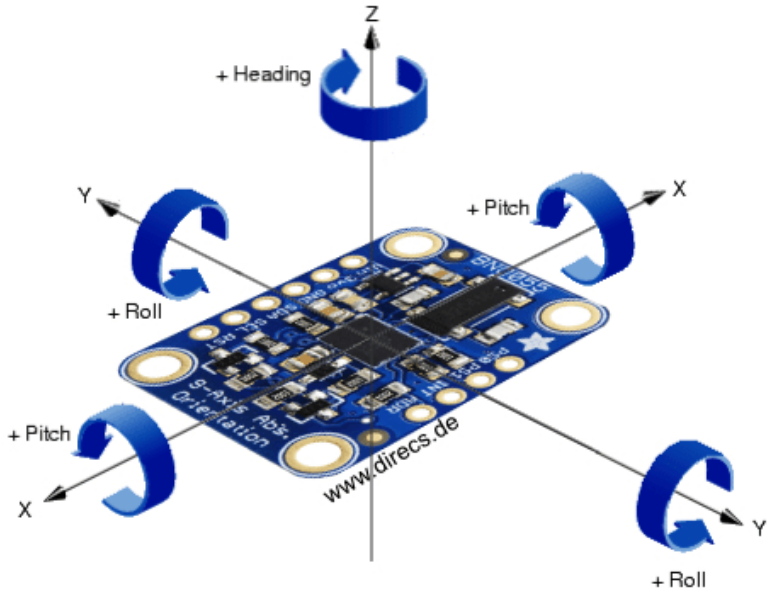

Figure 1: The axes of the BNO055 sensor

The BNO055 module we provide communicates over I2C. Your first task is to solder the header pins to the breakout board. Then consult the pinout available on canvas to connect the 3V3, GND, SDA, and SCL signals to the I2C port on the IO shield. We conveniently include an I2C client library called BNO055.c/h which provides simple direct function calls to access all the necessary data without having to deal with I2C manually.

Part 1: 2D 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 in space. As you rotate the sensor, the sensing axes of the sensor measure the axial 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 bias 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]

[2] For most sensors, the wideband noise is modeled 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.

The task is: given the noisy points on the ellipse; find the scale factor, cross-coupling, and biases 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.

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.
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

$$\hat{x} = \frac{1}{a}(x - x_0) \quad \text{and} \quad \hat{y} = \frac{1}{b}(y - y_0)$$

Then, for the corresponding $\hat{y}$, you have a circle:

$$\hat{x}^2 + \hat{y}^2 = R^2$$

Then, the idea is to:

1. Expand the ellipse equation
2. Write it out for a series of (x_i, 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. Get a parameter vector estimate from least squares
5. Get estimates of x_0, y_0, a, and b (you know R).

To get familiar with the problem, use `EllipseXYData.m` from the matlab folder in your repo and do the least squares, and solve for the parameters, and `x0`, `y0`, `a` and `b` (`R` is defined in the file). Then, plot the 2-norm of the pre-calibrated and post-calibrated data.[3]

Part 2: Naive 3D Calibration of Accelerometer and Magnetometer

Now that you have a sense of how the 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.

[3] The 2-norm is the Euclidean length of a vector, for a 2D vector it is $\sqrt{(x^2 + y^2)}$, for a N-dimensional vector it is the square root of the sum of the squares of the individual components: $\sqrt{(v^T v)}$

You have already done a simple two-point calibration before in the lab (generating a scale factor and bias/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.

The accelerometer measures not acceleration, but specific force:

$$\vec{a}_m = \frac{\vec{f}}{m} = (\vec{a}_I - \vec{g})$$

If you put an accelerometer on a table, it measures $-g = 9.8 m/s^2$. 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 +/- 1*g*. 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 bias 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 use anything else to help you maintain the alignment. Record the scale factors and bias values for each axis.

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 approximately 13 degrees east of north and 60 degrees down towards vertical. Note that the units are given in nT (10^{-9} Tesla). A more common unit is Gauss (1 G = 100000 nT). Thus the total field in Santa Cruz is 0.4784 G. Also these values change slightly over time! See Table 1.

There are two ways to do the naive calibration on the magnetometer axes. The first option is to rotate the sensor in the horizontal plane until the x-axis value peaks and the y-axis reads very close to zero (this corresponds to the point of "Horizontal Intensity" in Table 1). Mark that direction on your surface, and use that to set your value. This will allow you to do both horizontal axes. Analogously you can instead use the "Vertical Comp" to do the z-axis.

The second way is to try to match the magnetic field vector itself, and tilt and swing the magnetometer until the output values are maximal on x-axis, but zero on both the y- and z-axes. Since this might be a weird angle, you may have to create some kind of support structure and marking system to ensure that you can consistently replicate these readings.

Regardless of which technique you chose, calculate the magnetometer scale factors and bias values for each axis and record them.

Model Used:	IGRF2020						
Latitude:	36° 59' 24" N						
Longitude:	122° 3' 32" W						
Elevation:	850.0 ft 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**
2023-04-24	12° 54' 48"	60° 36' 8"	23,233.9 nT	22,646.3 nT	5,192.3 nT	41,237.2 nT	47,332.0 nT
Change/year	-0° 4' 37"/yr	0° 0' 6"/yr	-46.5 nT/yr	-38.3 nT/yr	-40.7 nT/yr	-79.8 nT/yr	-92.3 nT/yr

Table 1: 2018 Output of IGRF model for Earth's Magnetic Field from www.ngdc.noaa.gov. **You should run this again for today**, as magnetic fields change over time. Use 1156 High St, Santa Cruz CA as the address, 850 ft as the elevation, and select IGRF2020 as the model.

Part 3: Gyroscope

3.1 Bias and Bias Drift

In order to generate a high bandwidth attitude estimate, you are going to need gyroscopes as well as the accelerometers and magnetometers[4].

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 gyroscopes: spinning iron wheels, wine-glass ring oscillators, laser-ring, fiber optic, vibrating beam structures, and various MEMs technologies. They vary considerably 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 principal difficulties in using gyroscopes is their bias drift. That is, when you place a gyroscope on the table and turn it on, it does not read 0°/s. This is bias…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 constant), but can be assumed to be slowly varying. Cheaper gyroscopes (i.e.: the ones we will be using) have larger biases, and the drift rate is faster. Very expensive gyroscopes have very small biases and very low drift rates.

Set up your sensor on the bench, and take 10 seconds of gyroscope 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 mid-range fiber optic gyroscope, ~$3K, will be 1°- 5°/hr).

[4] In the next lab you will do an open-loop integration of the gyroscope and use the accelerometers and magnetometers as an aiding reference to feedback into the solution and keep it consistent.

10s reading
- mean of reading − BIAS]

10m reading
- drift = $\underline{3 \times \infty}$ − BIAS

3.2 Gyroscope Scale Factor via Angle Integration

The proper way to get scale factors for a gyroscope 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. Instead, we are going to calibrate the scale factor on the gyroscope by manually simulating a rate table (i.e. integrating the rotation rate through a fixed 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 gyroscope 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.

Part 4: Tumble Test

4.1 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. Find the following files in your matlab directory: `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 integers, it also generates the distortion matrix (`Adist`) and bias vector (`Bdist`). The function takes the number of datapoints you wish to create (e.g. `[A,M,Ad,Bd]= CreateTumbleData(1000)` creates 1000 points of data).

Now you are going to perform the full iterative least squares calibration using the simulated data. There are a number of steps required to do this. The first step is to map the raw output to units we care about. For the accelerometer, that would be g's, and for the magnetometer the units are nT. Look at the `CreateTumbleData.m` to find the dynamic ranges for the simulated data so you can do the necessary conversions to useful units.

Run the scaled data through the `CalibrateEllipsoidData3D.m` function. This will generate the $\tilde{A}$ matrix and B vector. If you recorded the distortion matrix (`Adist`) when you generated the data, then $A_{dist}\tilde{A}$ - $I_{3\times3}$ 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 B vector you just got out of the `CalibrateEllipsoidData3D.m` function. Plot the pre-calibration and post-calibration norm and calculate means and standard deviations for both. 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 iterative least squares calibration methods.

4.2 Real Tumble Test for Accelerometer and Magnetometer

NOTE: Due to supply issues, we changed accelerometers in 2022, so the mapping from sensor output to real-world useful units is ***not the same mapping*** *you did in the previous part for the simulated data. In this section, to map to that would be g's and nT units, the raw signals from the magnetometer need to be multiplied by 1000, and the raw accel data needs to be divided by 1000.*

In Part 1 we asked you to derive and apply an iterative least squares calibration for a hypothetical 2D sensor. In Part 4.1 we had you do the 3D 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. 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. You want two sets of tumbling data: a *training* set and a *validation*[5] set.

Once you have the data collected, plot the norm of both the accelerometer and magnetometer values 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 biases and scale factors you got earlier, and plot the norm again. Lastly, run the data you just collected through `CallibrateEllipsoid3D.m` and `CorrectEllipsoid3D.m`, which will output new 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 for both the naive and iterative 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?

Part 5: Check-Off and Lab Report

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 calibration derived from your tumble test. This will then give you a calibrated unit vector

[5] Data not that is not used in the calibration process and reserved for validation is known as a validation set. If the calibration works on the validation set, then you have some confidence that the parameters are not overtuned.

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.

Lab 3 Rubric

- 40% check off (10 point scale)
    - Accelerometer: Show uncalibrated vs calibrated point clouds (1 point)
        - Show that z-axis corresponds to 1g when flat, -1g when upside-down (2 points)
    - Magnetometer: Show uncalibrated vs calibrated point clouds (1 point)
        - Show us the offsets you found in your Matlab code (1 point)
        - Do a live tumble test for 10s and output average, max and min values for each axis (1 point)
    - Gyroscope Angle Output: Demonstrate that you can rotate your gyro 6 random points between 0-180 and get correct angle output. Show the tutor/TA at least 10 seconds of data for each data point, and show that the drift is less than 3 degrees (4 points)
- 15% code quality (comments, readability)
- 15% code compiles
- 30% lab report
    - General format followed; all sections of lab thoroughly addressed (15 points)
    - Writing detailed enough for replication (10 points)
    - Image quality and writing quality (grammar, spelling, appropriate tone) (5 points)

Appendix 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 auto generate names based on 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 later 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.