In attempting to gain further insight into into calibration techniques for IMU decided to do a bit of a dive into the calibration techniques from accelerometers, magnetometers and gyroscopes. So here goes.

**Magnetometers**

From my readings there are 2 primary types of calibrations that can be performed for magnetometers as well as accelerometers but first lets talk about magnetometers:

1. Elipsoid to Sphere fitting and
2. 6-point tumble calibration.
3. Magnetic Declination Correction

Using Microsoft CoPilot:

Calibrating a magnetometer is essential to ensure accurate readings by correcting for various distortions. Here are some common calibration techniques:

1. **Hard-Iron Calibration**:
   * **Cause**: Hard-iron distortions are caused by permanent magnetic objects near the sensor, creating a constant offset in the magnetic field.
   * **Correction**: This involves finding the offset values for the X, Y, and Z axes by rotating the magnetometer in all directions and calculating the midpoint between the minimum and maximum values for each axis. [These offsets are then subtracted from the raw readings1](https://www.digikey.com/en/maker/projects/how-to-calibrate-a-magnetometer/50f6bc8f36454a03b664dca30cf33a8b).
2. **Soft-Iron Calibration**:
   * **Cause**: Soft-iron distortions are due to ferromagnetic materials near the sensor that distort the magnetic field, causing the data to form an ellipsoid instead of a sphere.
   * **Correction**: This involves fitting an ellipsoid to the data points and then transforming it into a sphere. [This process adjusts the scale and alignment of the data to correct for the distortions1](https://www.digikey.com/en/maker/projects/how-to-calibrate-a-magnetometer/50f6bc8f36454a03b664dca30cf33a8b).
3. **Magnetic Declination**:
   * **Cause**: The difference between magnetic north and true north varies depending on your location on Earth.
   * [**Correction**: This involves adjusting the magnetometer readings to account for the local magnetic declination, which can be obtained from geomagnetic models or online calculators1](https://www.digikey.com/en/maker/projects/how-to-calibrate-a-magnetometer/50f6bc8f36454a03b664dca30cf33a8b).
4. **Ellipsoid Fitting**:
   * **Cause**: Combined hard-iron and soft-iron distortions.
   * **Correction**: This technique involves fitting an ellipsoid to the collected data points and then transforming it into a sphere. [This method corrects for both types of distortions simultaneously2](https://resources.inertiallabs.com/resources/magnetometer-calibrations-and-the-inertial-labs-ins-p).
5. **Online Calibration**:
   * **Cause**: Dynamic changes in the environment or sensor placement.
   * [**Correction**: This involves continuously updating the calibration parameters in real-time as the sensor operates, ensuring accurate readings even in changing conditions3](https://www.mdpi.com/1424-8220/21/16/5288).

Ellipsoid Fitting addresses issues identified in Hard Iron and Soft Iron distortions.

Recommended readings:

References:

1. [Learn more about magnetometer models and HSI calibration · VectorNav](https://www.vectornav.com/resources/inertial-navigation-primer/specifications--and--error-budgets/specs-hsicalibration)
2. [Tutorial: How to calibrate a compass (and accelerometer) with Arduino](https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/): https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/
3. [Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference – NXP AN4246](https://www.nxp.com/docs/en/application-note/AN4246.pdf)

Other than Ellipsoid/Sphere fitting another method is available:

The 6-point tumble calibration is a method used to calibrate magnetometers (and other 3-axis sensors like accelerometers) by measuring the sensor’s output in six different orientations. This technique helps to determine and correct for offsets, gains, and cross-axis sensitivities. Here’s a step-by-step overview:

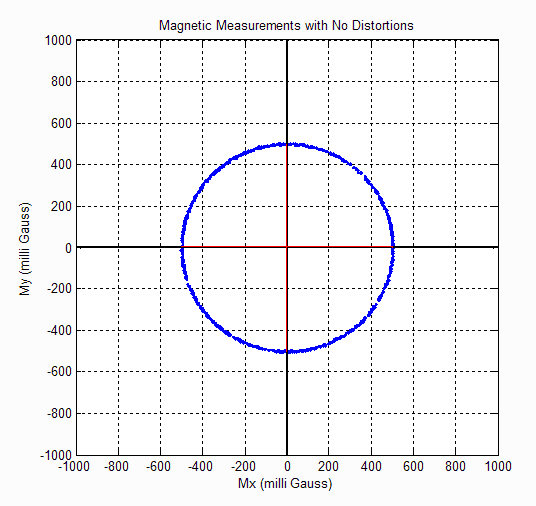
1. **Positioning**: The sensor is placed in six different orientations:
   * +X axis up
   * -X axis up
   * +Y axis up
   * -Y axis up
   * +Z axis up
   * -Z axis up
2. **Data Collection**: In each position, the sensor’s output is recorded. Ideally, these measurements should reflect the true magnetic field vector in each orientation.
3. **Calculations**:
   * **Offsets**: The average of the positive and negative measurements for each axis gives the offset. For example, the offset for the X-axis is calculated as ((X\_{+} + X\_{-}) / 2).
   * **Gains**: The difference between the positive and negative measurements for each axis gives the gain. For example, the gain for the X-axis is calculated as ((X\_{+} - X\_{-}) / 2).
   * **Cross-axis Sensitivities**: These are calculated to correct for any misalignment between the sensor axes.
4. **Correction**: The calculated offsets, gains, and cross-axis sensitivities are then used to correct the raw sensor data, ensuring accurate readings.

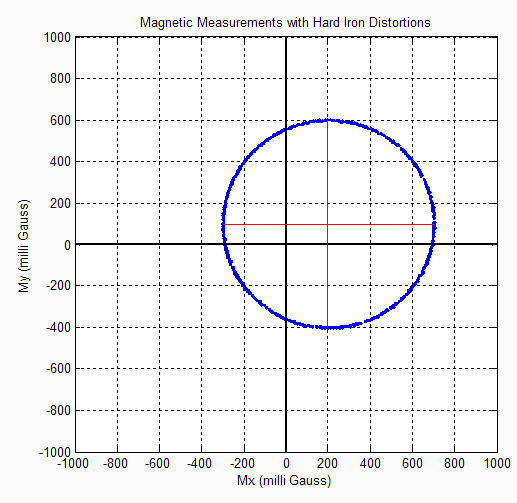
[This method is particularly useful because it doesn’t require any special equipment or reference fields, just the ability to accurately position the sensor in the six specified orientations1](https://community.st.com/t5/mems-sensors/6-point-calibration-of-magnetometer/td-p/274987)[2](https://www.st.com/resource/en/design_tip/dt0053-6point-tumble-sensor-calibration-stmicroelectronics.pdf).

**Elipsoid to sphere fitting**

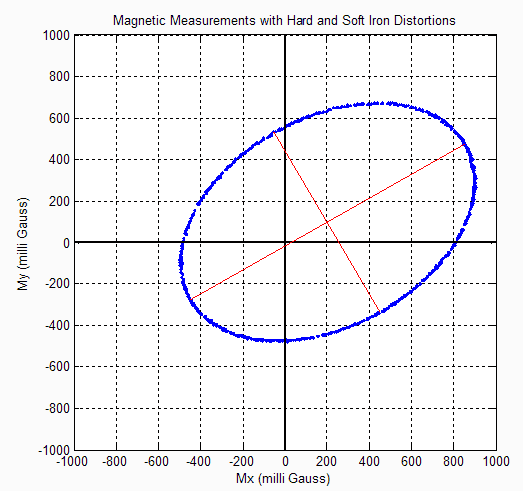
Reference 1 does a nice job of showing the effects of Hard and Soft Iron distortion and the effect of calibration using fitting.

In the case of a magnetometer with no distortions a plot of the My vs Mx shows a perfect sphere.

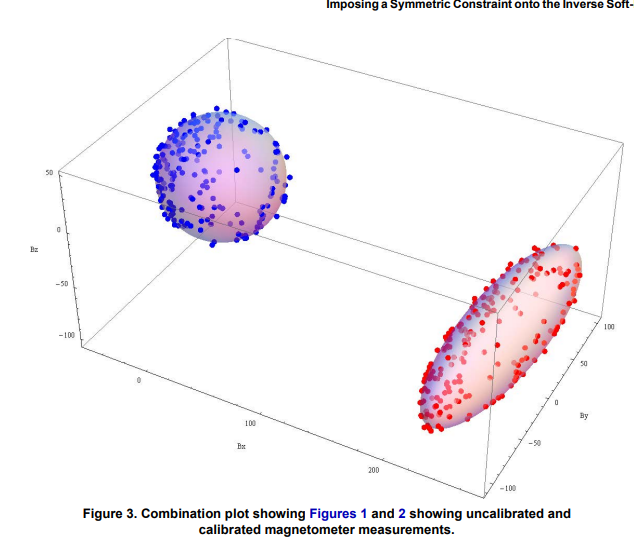


When the magnetometer is exposed to Hard Iron distortion such as a magnet or electronic sources it shifts the sphere in the plot:  


When exposed to Hard and Soft Iron distortions such as pieces of iron or other metals you will see a elongation of sphere as well as the shift in position:



From AN4246 the effect of magnetometer calibration is shown:



**Calibration Tools for Ellipsoid Fitting**

**FreeIMU GUI Toolset**

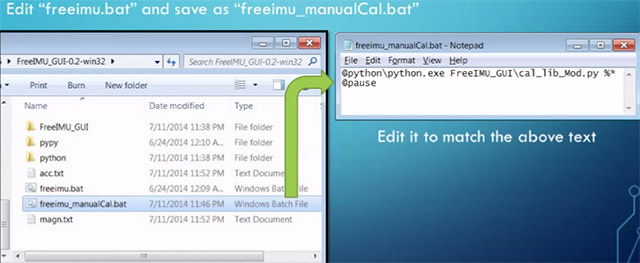
**From** [Adafruit post](https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/calibration) **on the calibration of the LSM303 accelerometer and magnetometer it states that:**

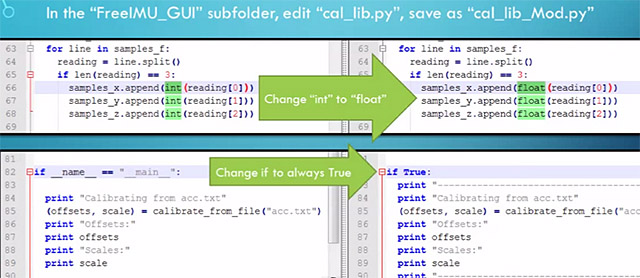
***For super-precise accelerometer calibration, you will want to check out the FreeIMU Magnetometer and Accelerometer GUI by the late Fabio Varesano. The image above (from Fabio's site) shows a graphical representation of the sensor readings and the resulting calibration offsets calculated from the raw data.***

***This comprehensive calibration suite is designed to run on a PC. It is much too large to run on a microcontroller, like the Arduino, but the resulting calibration offsets it generates can be incorporated into your Arduino sketch for better accuracy.***

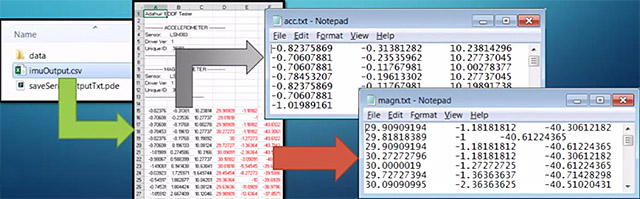
**Unfortunately with Fabio Versano’s untimely passing the original GUI hasn’t been updated from PYQT4. But fortunately some of us has either converted the GUI to PYQT5 or generated a Windows Executable from the original PYT4 so no need to even install Python.**

**You can obtain the various incarnations of**  [Varasano’s FreeIMU Calibration Application](https://github.com/mjs513/FreeIMU-Updates/wiki/04.-FreeIMU-Calibration" \t "_blank). In addition the link shows how to use the GUI.

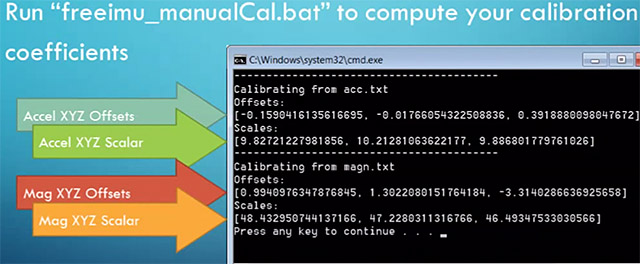
*In the video* [*Adafruit 10 DOF IMU Calibration with FreeIMU GUI Toolset*](https://www.youtube.com/watch?v=G6EtAeidDXk) *it also shows you how to use that shows how a couple of simple config file edits let you run the FreeIMU GUI Toolset in manual mode. Reference 2 includes the instructions on how to do this with Screenshots from the video:  
(These are screen shots from that video)[](https://thecavepearlproject.org/wp-content/uploads/2015/05/freeimu_videoscreencap1.jpg)*

*[](https://thecavepearlproject.org/wp-content/uploads/2015/05/freeimu_videoscreencap2.jpg)*

*These changes allow you to run the application without the GUI, so long as you provide a couple of tab delimited text files of data.  The video goes into some detail showing how to use a processing sketch to save serial output from Adafruit 10 DOF IMU as a csv file, but all I did the first few times was copy and paste data directly from the serial window into a spreadsheet, and from there into notepad. (since my units are data loggers, I could use the csv files on the SD cards for the in-housing tests I did afterwards)*

*[](https://thecavepearlproject.org/wp-content/uploads/2015/05/freeimu_videoscreencap3.jpg)*

*Then you save “acc.txt” and magn.txt” in the FreeIMU GUI folder, right beside the freeimu\_manualCal.bat file that you modified earlier. Once you have your data files in place, run “Freeimu\_manualCal.bat”. On my machine the GUI still launches – displaying no data, but a command line window also opens:*

*[](https://thecavepearlproject.org/wp-content/uploads/2015/05/freeimu_videoscreencap5.jpg)*

If you use the Gui it does expect data in as raw counts from the sensor and the ability to respond to commands from the GUI. We will get into that in a bit. The nice thing with the GUI is its ability to do the calibration for both the accelerometer and magnetometer.

As for applying the calibration offsets and scale its:

**CalibratedData = ( unCalibratedData – Offset ) / Scaling Factor**

**After this you apply the accelerometer sensitivity, same goes for the magnetometer.**

**Example:**

As a common test bed I am using a Teensy 4.1 with a MPU-9250/MS5637 Pressure Sensor (unfortunately I do not believe it is available anymore) but any MPU-9250 should work. Access is via I2C but you can use SPI as well. A modified Bolderflight Invense-IMU library is being used for the basis of the test. The modified library not only supports the MPU-9250/MPU-6500 but also the Invense ICM-20649 and the ICM-20948 MPU’s. The MPU-9250 library was modified to add in 3 additional functions:

1. void getScales(float \*accScale, float \*gyroScale, float \*magScale) – which allows you to get the getting sensitivity values for the accelerometer, gyroscope and magnetometer. Note magScale for the MPU9250 is a 3 element array. When called will return:

Accelerometer Scale: 0.00049 G/LSB (assumes 16G config for accel)

Gyro Scale: 0.06104 degrees/sec/LSB (assumes 2000 dps for gyro config)

Magnetometer Scales:

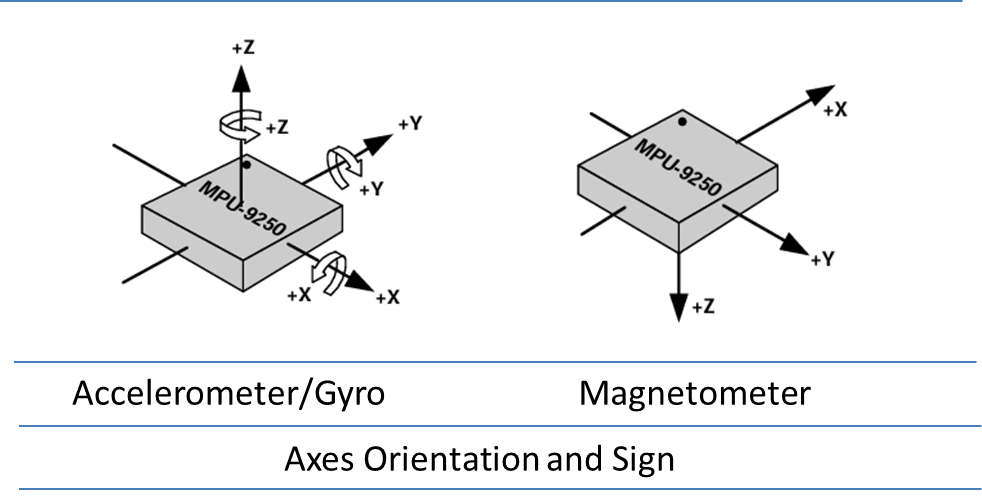
Magx: 0.17922 uT/LSB

Magx: 0.17922 uT/LSB

Magx: 0.17278 uT/LSB

1. bool Read\_raw(int16\_t \* values) – returns an array of 9 elements in raw counts from the MPU, 3 for accel, 3 for gyro and then 3 for magnetometer. Data is returned unaligned/
2. Mpu9250::Read(float \* values) – returns an array of 10 values, 3 accel (in g’s), 3 gyro (dps), 3 magnetometer (uT) and 1 for temperature. *Data is returned with accelerometer axes aligned with magnetometer axis, +Y swapped with +X and +Z Points down (az and gz multiplied by -1). We will talk about the necessity of aligned axis later.*

MPU-9250 Axes Orientation and Signs



**In the case of FreeIMU GUI data is feed with axes unaligned (raw values) and user has to swap axes prior to sending data to the AHRS algorithm.**

**Appendix 2 shows a basic Arduino Sketch to use with the FreeIMU GUI.**

**The output of the GUI is a calibration file in addition to showing the data on the screen:**

|  |
| --- |
| **#define CALIBRATION\_H**  **const int acc\_off\_x = -37;**  **const int acc\_off\_y = 30;**  **const int acc\_off\_z = 130;**  **const int acc\_scale\_x = 2047.000000;**  **const int acc\_scale\_y = 2060.000000;**  **const int acc\_scale\_z = 2065.000000;**  **const int magn\_off\_x = 99;**  **const int magn\_off\_y = 16;**  **const int magn\_off\_z = -186;**  **const int magn\_scale\_x = 239.000000;**  **const int magn\_scale\_y = 246.000000;**  **const int magn\_scale\_z = 263.000000;** |

**Calibration data is applied as follows:**

|  |
| --- |
| **values[1] = ((float)(raw\_values[0] - cal\_acc\_off[0]) / (float)cal\_acc\_scale[0]);**  **values[0] = ((float)(raw\_values[1] - cal\_acc\_off[1]) / (float)cal\_acc\_scale[1]) ;**  **values[2] = -((float)(raw\_values[2] - cal\_acc\_off[2]) / (float)cal\_acc\_scale[2]) ;**  **values[6] = ((float)(raw\_values[6] - cal\_magn\_off[0]) / (float)cal\_magn\_scale[0]);**  **values[7] = ((float)(raw\_values[7] - cal\_magn\_off[1]) / (float)cal\_magn\_scale[1]) ;**  **values[8] = ((float)(raw\_values[8] - cal\_magn\_off[2]) / (float)cal\_magn\_scale[2]) ;** |

**Magneto 1.2**

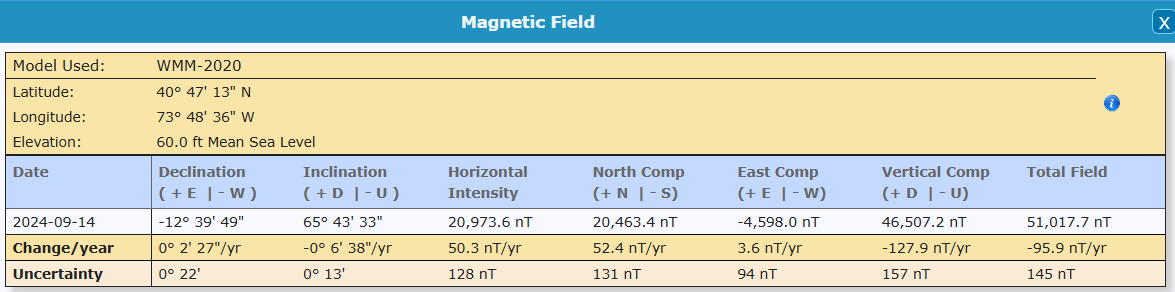
**From Copilot:**

*Magneto 1.2 is a tool used for calibrating 3-axis magnetometers, specifically addressing soft-iron and hard-iron distortions. Here’s a brief overview of the calibration process using Magneto 1.2:*

1. ***Collect Raw Data****: First, you need to gather raw magnetometer data. This involves rotating the sensor in various orientations to capture a wide range of magnetic field readings. The data should be saved in a tab-delimited text file.*
2. ***Determine Local Magnetic Field Strength****: Use a tool like NOAA’s World Magnetic Model to find the local magnetic field strength at your calibration location. This value is necessary for accurate calibration.*
3. ***Run Magneto****: Input the raw data file into Magneto 1.2. The software will process the data and provide calibration parameters, including bias (hard-iron distortion) and a transformation matrix (soft-iron distortion).*

[*For a detailed guide, you can refer to the GitHub repository by Michael Wrona1*](https://github.com/michaelwro/mag-cal-example)*. It includes Python scripts and an Arduino example to help you through the calibration process.*

**Magneto 1.2 is an ellipsoid fitting application that “ best fits the raw data points, using however different techniques. MagCal uses an “adaptive least square estimator, and Magneto uses the “Li's ellipsoid specific fitting algorithm”.**

**Magneto 1.2 can be downloaded from** [**Improved magnetometer calibration (Part 2)**](https://sailboatinstruments.blogspot.com/2011/09/improved-magnetometer-calibration-part.html)**. Basically Magneto 1.2 requires you to input the norm of earth’s magnetic field at your current location. To do this you can use** [NOAA Magnetic Field Calculator](https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml#igrfwmm)**. It allows you to either manually put in your latitude, longitude and elevation or put in your location which will fill in the information for you. Example for my location:  
**

**The value you are looking for is the Total Field value (last column). Note that it is in nT. Data is coming out of the MPU9250 as uT’s so you would divide the total field by 1000, or 51.0177uT.**

**Not going to reinvent the wheel here. The author of** [Tutorial: How to calibrate a compass (and accelerometer) with Arduino](https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/)

… according to the author on the [Sailboat Instruments](http://sailboatinstruments.blogspot.com/2011/09/improved-magnetometer-calibration-part.html) site you only need to match the total field “norm” values if you want the final output on an absolute scale:

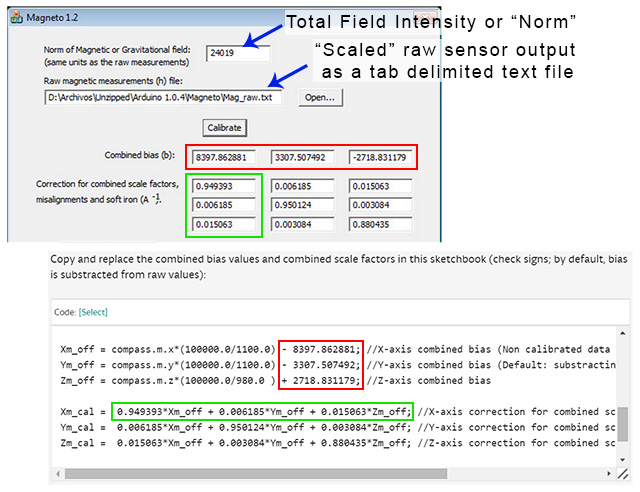
*“Magneto expects to receive raw data in +- format (a value of zero indicating a null field in the current axis), but not necessarily normalized to +-1.0.*

*If your sensors have SPI or I2C outputs, they will usually directly produce the required format. For example, the MicroMag3 magnetometer directly produces counts from -3411 to +3411, and the the SCA3000 accelerometer directly produces counts from -1333 to 1333, and Magneto can process directly these values, without the need to normalize them to +- 1.0. I understand that a normalization may be desirable to avoid machine precision problems, but this has not been the case with these sensors.*

*If your sensors produce voltage levels that you have to convert to counts with an ADC, you have indeed to subtract a zero field value from the ADC output before using Magneto. You would then normally choose the maximum positive value as input to the ‘Norm of Magnetic or Gravitational field’.*

*But this norm value is not critical if all you want to calculate later on is a heading (if it is a magnetometer) or a tilt angle (if it is an accelerometer). You can input any reasonable value for the norm, the correction matrix will be different by just a scaling factor, but the calculated heading (or tilt angle) will be the same, as it depends only on the relative value of the field components. The bias values will be unchanged, as they do not depend on the norm.”*

Once I had my raw readings at the same scale as the Total Intensity numbers, I could hit the calibrate button, taking care to put the generated correction factors in the right section of the matrix calculation code:

[](https://thecavepearlproject.org/wp-content/uploads/2015/05/using-magneto1.jpg)Rather than simply finding an offset and scale factor for each axis, Magneto creates twelve different calibration values that correct for a whole set of errors: bias, hard iron, scale factor, soft iron and misalignment. As you can see from the example above, this makes calculating the corrected data a bit more involved than with FreeIMU. I am not really sure I want to sandbag my loggers with all that floating point math (mistakes there have given me grief in the past) so I will probably offload these calculations to post processing with Excel.  To check that your calculations are working OK, keep in mind that in the absence of any strong local magnetic fields, the maximum readings should reflect the magnetic field of the earth which ranges between 20 and 60 micro-Teslas.

I will add one more note on the total field intensity:

*I am quoting a part of a previous answer as it is directly relevant to your question:  
"You would then normally choose the maximum positive value as input to the 'Norm of Magnetic or Gravitational field'.*

In the case of the MPU-9250 the values you would input are the raw values multiplied by the associated scale factors for the magnetometer or accelerator.

In our case for the MPU-9250 once you get the A\_inv and A-Offsets:

|  |
| --- |
| //Magneto 1.2 Calibration  float MagOffset1[3] = {17.913548, 3.018422, -32.074879};  float mCal1[3][3] =  {  {1.038134, 0.001051, -0.007501},  {0.001051, +1.039582, 0.007319},  {-0.007501, 0.007319, +1.008515}  };  float AccOffset1[3] = {-0.008955, 0.012518, 0.062580};  float aCal1[3][3] =  {  {-0.008955, 0.010505, 0.004729},  {0.010505, 1.012468, 0.002124},  {0.004729, 0.002124, 0.995591}  }; |

And to apply the calibration:

|  |
| --- |
| float x = (float)raw\_values[6] \* magScale[0];  float y = (float)raw\_values[7] \* magScale[1];  float z = (float)raw\_values[8] \* magScale[2];  float magxc = mCal1[0][0]\*(x-MagOffset1[0])+ mCal1[0][1]\*(y-MagOffset1[1]) + mCal1[0][2]\*(z-MagOffset1[2]);  float magyc = mCal1[1][0]\*(x-MagOffset1[0])+ mCal1[1][1]\*(y-MagOffset1[1]) + mCal1[1][2]\*(z-MagOffset1[2]);  float magzc = mCal1[2][0]\*(x-MagOffset1[0])+ mCal1[2][1]\*(y-MagOffset1[1]) + mCal1[2][2]\*(z-MagOffset1[2]);  float ax = (float)raw\_values[0] \* accelScale;  float ay = (float)raw\_values[1] \* accelScale;  float az = (float)raw\_values[2] \* accelScale;  float agxc = aCal1[0][0]\*(ax-AccOffset1[0])+ aCal1[0][1]\*(ay-AccOffset1[1]) + aCal1[0][2]\*(az-AccOffset1[2]);  float agyc = aCal1[1][0]\*(ax-AccOffset1[0])+ aCal1[1][1]\*(ay-AccOffset1[1]) + aCal1[1][2]\*(az-AccOffset1[2]);  float agzc = aCal1[2][0]\*(ax-AccOffset1[0])+ aCal1[2][1]\*(ay-AccOffset1[1]) + aCal1[2][2]\*(az-AccOffset1[2]); |

There are a couple of interesting applications along the same lines: Matlab ellipsoid fitting - <https://www.mathworks.com/matlabcentral/fileexchange/23377-ellipsoid-fitting>



**PJRC Motion Cal App for Propshield**

Adafruit has a nice write up on how to use it and is available at: [Magnetometer Calibration | AHRS for Adafruit's 9-DOF, 10-DOF, LSM9DS0 Breakouts | Adafruit Learning System](https://learn.adafruit.com/ahrs-for-adafruits-9-dof-10-dof-breakout/magnetometer-calibration)

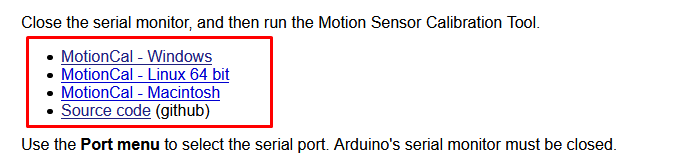
But to quote since I don’t like to hit links:

[Paul Stoffregen of PJRC](https://www.pjrc.com/) wrote a really awesome cross-platform calibration helper that is great for doing both soft and hard iron magnetometer calibration. What's nice about it is you get a 3D visualization of the magnetometer output and it also tosses outliers and tells you how much spherical coverage you got!

# Step 1 - Download MotionCal Software

MotionCal is available for Mac, Windows and Linux, [you can download it from clicking here](https://www.pjrc.com/store/prop_shield.html).

Look for this section in the website:

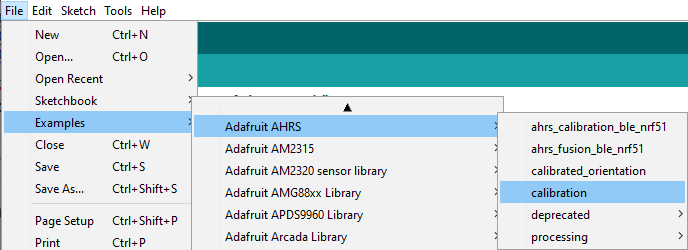
[](https://learn.adafruit.com/assets/87416)

And click the one that matches your computer the best.

# Step 2 - Configure & Upload the AHRS calibration Example

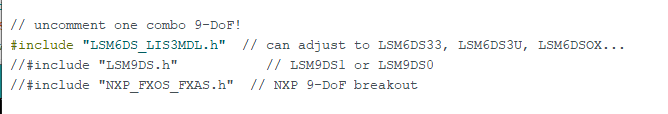
Next we have to tell the microcontroller board to send the magnetometer (and, if there is one, accelerometer and gyroscope) data out over serial in the right format.

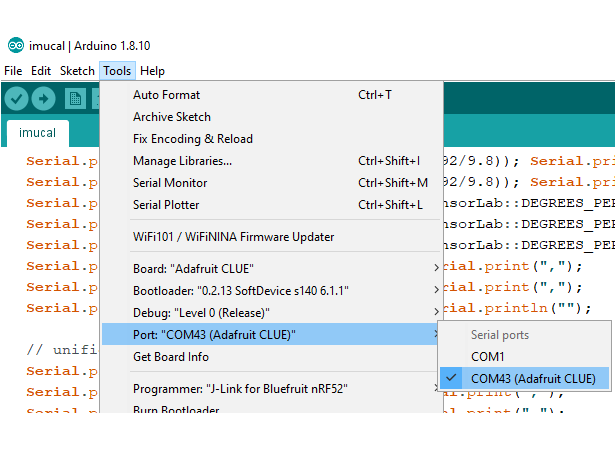
Open up the Adafruit\_AHRS->calibration example

[](https://learn.adafruit.com/assets/88464)

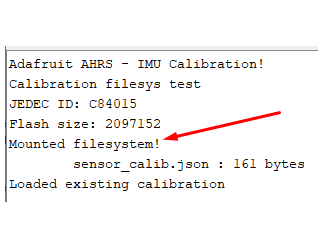
At the top of the sketch you'll see a section where you can #include different sensor sets. Not every sensor-set is defined, but our most popular ones are! (You'll need sensors that are Adafruit\_Sensor compatible.)

Uncomment whichever kit you are using, and comment out the rest

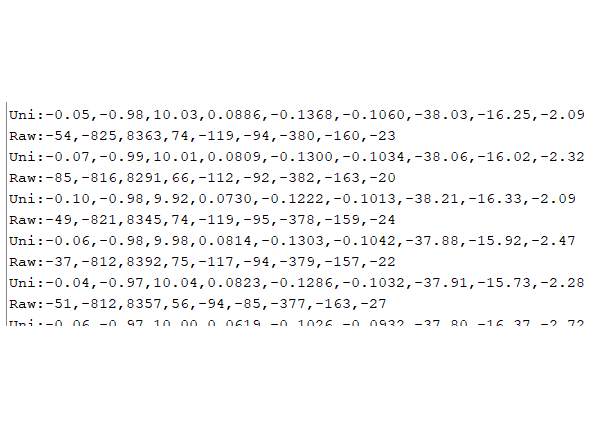
[](https://learn.adafruit.com/assets/88465)

[](https://learn.adafruit.com/assets/87420)

Select your desired board & port from the **Tools** menu then click **Upload**

[](https://learn.adafruit.com/assets/88466)

Open up the serial console and check that the EEPROM/Filesystem was found. There may already be an existing calibration from prior experiments

[](https://learn.adafruit.com/assets/87424)

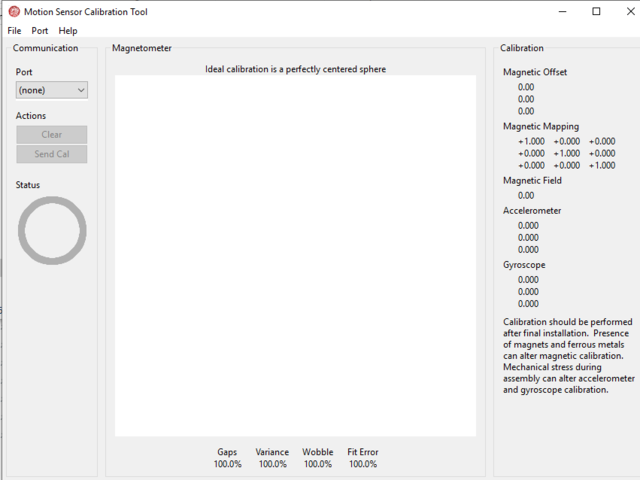
You'll then see a stream of data that looks like:  
Raw:-58,-815,8362,76,-121,-95,-375,-159,-24  
Uni:-0.07,-0.98,10.00,0.0832,-0.1327,-0.1046,-37.50,-15.93,-2.50

The first three numbers are accelerometer data - if you don't have an accelerometer, they will be 0

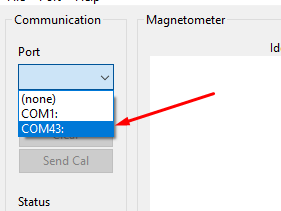
The middle three numbers are gyroscope data - if you don't have an gyroscope, they will be 0

The last three numbers are magnetometer, they should definitely not be zeros!

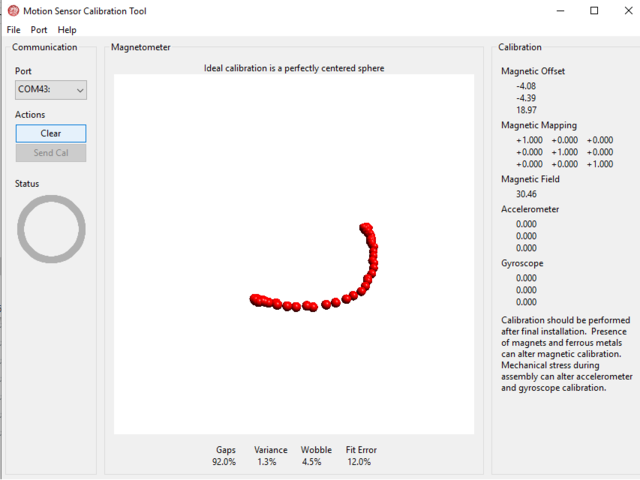
# Step 3 - Run MotionCal

[](https://learn.adafruit.com/assets/87425)

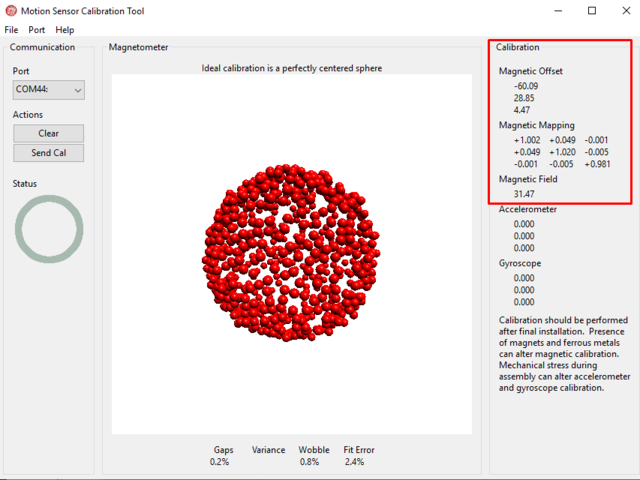
**Close the serial port**, and launch MotionCal

[](https://learn.adafruit.com/assets/87426)

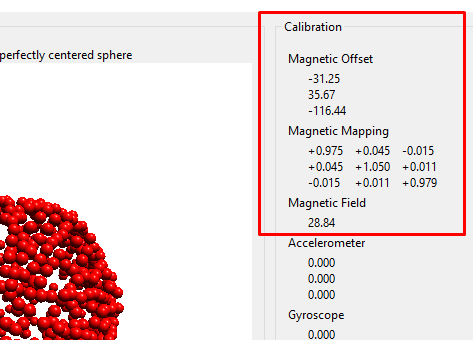
Select the same COM / Serial port you used in Arduino

[](https://learn.adafruit.com/assets/87427)

Twist the board/sensor around. Make sure its not near any strong magnets (unless that's part of the installation)

[](https://learn.adafruit.com/assets/87428)

Keep twisting until you get a complete 'sphere' of red dots. At this point you are calibrated!

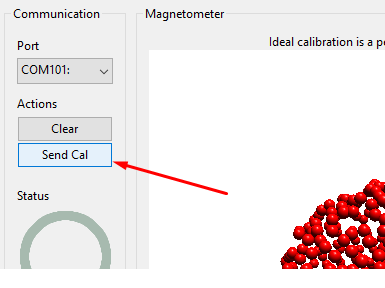
[](https://learn.adafruit.com/assets/88471)

In the top right you'll see the hard magnetic offsets at the top, the soft offsets in the middle and the field strength at the bottom.

In this case, the hard iron offsets are [-31.25, 35.67, -116.44]

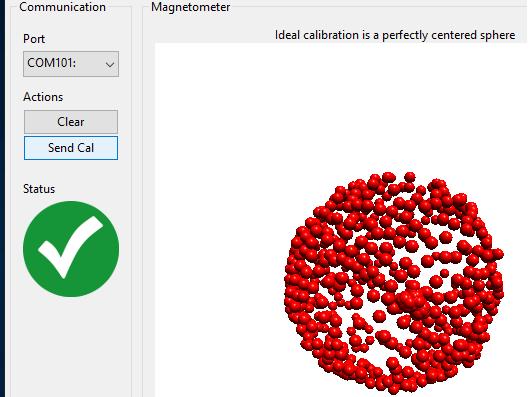
**Take a screenshot of this display, so you can refer to these numbers later!**

MotionCal does not calibrate the accelerometer or gyroscope (yet) - so those offsets will be zero

[](https://learn.adafruit.com/assets/88468)

Eventually you'll have enough datapoints that the **Send Cal**  button will activate (its grayed out by default).

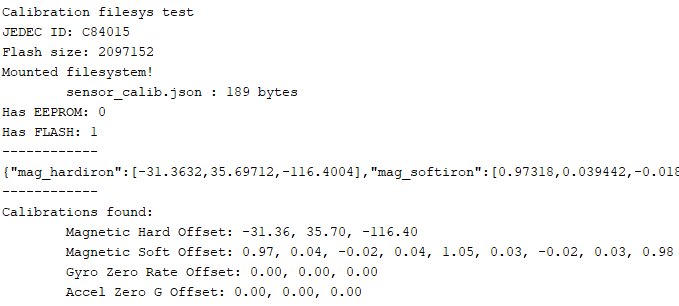
Once you can click the button, try clicking it (we had to try a few times?)

[](https://learn.adafruit.com/assets/88469)

You'll see a large green checkmark once the calibration is saved and verified!

# Step 4 - Verify Calibration

Re-load the **sensor\_calibration\_read** sketch to verify the calibration was saved!

[](https://learn.adafruit.com/assets/88472)

There is a trick to using MotionCal with other that the Propshield software. This has been identified in a issue generated against the app on github ([Data input range/scale documentation · Issue #12 · PaulStoffregen/MotionCal (github.com)](https://github.com/PaulStoffregen/MotionCal/issues/12)

### [dr-mrsthemonarch](https://github.com/dr-mrsthemonarch)commented [on Jul 26, 2021](https://github.com/PaulStoffregen/MotionCal/issues/12#issuecomment-886573144)

|  |
| --- |
| I have the same questions...Though according to the adafruit comments in their code, it seems the software looks for  Accel in 'raw' format 2^13 (8192) integers Gyroscope in Degrees/s rounded to integers Mag in microteslas \* 10  Though I get the motioncal software to work when using 2^14 raw accel format and the gyroscope in degrees/s, and the mag multiplied by 10 I actually don't believe the values are particularly correct, because I'm having quite a difficult time getting real angles from the sensors itself (i'm using MPU9250) |

Adafruit has adopted there calibration code to address issue. In our case of the MPU9250 when we raw values rescaled from the values output from the library:

|  |
| --- |
| float raw\_values[9];  // get and print uncalibrated data  if(imu.Read(values)){  Serial.print("Raw:");  Serial.print(int(values[0] \*8192/9.805));  Serial.print(',');  Serial.print(int(values[1] \*8192/9.805));  Serial.print(',');  Serial.print(int(values[2] \*8192/9.805));  Serial.print(',');  Serial.print(int (values[3]\*16 ));  Serial.print(',');  Serial.print(int (values[4]\*16 ));  Serial.print(',');  Serial.print(int (values[5]\*16 ));  Serial.print(',');  Serial.print(int (values[6]\*10));  Serial.print(',');  Serial.print(int (values[7]\*10));  Serial.print(',');  Serial.print(int (values[8]\*10));  Serial.println(""); |

The whole sketch is shown at appendix 3.

|  |
| --- |
| Once you have the A\_inv and A\_offset values: //MotionCal (PJRC) calibration  float MagOffset[3] = {17.791,3.680,-32.655};  float mCal[3][3] =  {  {1.0034,-0.0051,-0.0025},  {-0.0051,1.0065,0.0065},  {-0.0025,0.0065,0.9903}  }; |

They get applied as follows :

|  |
| --- |
| float x = (float)raw\_values[6] \* magScale[0];  float y = (float)raw\_values[7] \* magScale[1];  float z = (float)raw\_values[8] \* magScale[2];  float magxc = mCal[0][0]\*(x-MagOffset[0])+ mCal[0][1]\*(y-MagOffset[1]) + mCal[0][2]\*(z-MagOffset[2]);  float magyc = mCal[1][0]\*(x-MagOffset[0])+ mCal[1][1]\*(y-MagOffset[1]) + mCal[1][2]\*(z-MagOffset[2]);  float magzc = mCal[2][0]\*(x-MagOffset[0])+ mCal[2][1]\*(y-MagOffset[1]) + mCal[2][2]\*(z-MagOffset[2]); |

Other sources: [**How to Calibrate a Magnetometer | Digi-Key Electronics**](https://www.youtube.com/watch?v=cGI8mrIanpk)**.** A written guide associated with the video is [How to Use a Calibrated Magnetometer as a Compass (digikey.com)](https://www.digikey.com/en/maker/projects/how-to-calibrate-a-magnetometer/50f6bc8f36454a03b664dca30cf33a8b). It recommends the following resouces:

* [Adafruit Magnetometer Calibration Tutorial](https://learn.adafruit.com/adafruit-sensorlab-magnetometer-calibration)
* [How to Calibrate a Magnetometer?](https://www.appelsiini.net/2018/calibrate-magnetometer/)
* [Tutorial: How to calibrate a compass (and accelerometer) with Arduino](https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/)
* [Magnetometer Hard & Soft Iron Calibration](https://www.vectornav.com/resources/inertial-navigation-primer/specifications--and--error-budgets/specs-hsicalibration)
* [Calibrate an eCompass in the Presence of Hard- and Soft-Iron Interference](https://www.nxp.com/docs/en/application-note/AN4246.pdf)

**6-point tumble calibration**

As stated earlier:

The 6-point tumble calibration is a method used to calibrate magnetometers (and other 3-axis sensors like accelerometers) by measuring the sensor’s output in six different orientations. This technique helps to determine and correct for offsets, gains, and cross-axis sensitivities.

Other sources:

[(105) How To - Calibrate 6 Point IMU Calibration 3 Axis Gimbal Stabilizer - YouTube](https://www.youtube.com/watch?v=MypiJUt3ptc)

[Easy Hard and Soft Iron Magnetometer Calibration:](https://www.instructables.com/Easy-hard-and-soft-iron-magnetometer-calibration/)



Source is for the MPU-9250 is attached as a zip file – a bit big at present to include as sttachment

The Six point calibration method that we use for the MPU-9250, which can be adapted for any IMU (combo of accelerometer, gyroscope and magnetometer is attached. It provides several option to get calibration data for the MPU.

On startup the sketch presents you with a menu of options:

Beginning IMU Initialization...

IMU Initialization Complete

Teensy Accel and Gyro EEPROM Biases:

Current Accel Biases (mss) and Scale Factors:

0.000000, 1.000000

0.000000, 1.000000

0.000000, 1.000000

Current Mag Biases (uT) and Scale Factors:

0.000000, 1.000000

0.000000, 1.000000

0.000000, 1.000000

IMU Biases:

Accel X Bias: 0.000000

Accel X Scale Factor: 1.000000

Accel Y Bias: 0.000000

Accel Y Scale Factor: 1.000000

Accel Z Bias: 0.000000

Accel Z Scale Factor: 1.000000

Gyro X Bias: 0.000000

Gyro Y Bias: 0.000000

Gyro Z Bias: 0.000000

Mag X Bias: 0.000000

Mag X Scale Factor: 1.000000

Mag Y Bias: 0.000000

Mag Y Scale Factor: 1.000000

Mag Z Bias: 0.000000

Mag Z Scale Factor: 1.000000

Enter one of the following commands using the serial terminal:

Enter 'a' to preform MPU library accel calibrations

Enter 'm' to perform MPU library mag calibrations

Enter 'g' to perform MPU library gyro calibrations

Enter 's' to perform static IMU bias calibrations

Enter 'd' to display all calibration values

Enter 'z' to reset all calibration values to zero

Enter 'p' to print corrected IMU readings to serial

Enter 'e' to load static cal values to EEPROM

Enter 'i' to load static cal values to IMU

Enter 'l' to load MPU Library cal values to EEPROM

Enter 'r' to calculate IMU sensor noise sigmas

**The options you are interested in is ‘a’cceleration, ‘m’agnetometer calibration. ‘s’tatic calibration and ‘r’ to calculate IMU sensor noise. When you run mag and accel calibration it will prompt you when to rotate the sensor axis. Take a look at appendix 4 for a good way to do it. After cal you can do a ‘d’ to display the values and you will see:**

**Printing IMU:**

**IMU Biases:**

**Accel X Bias: 0.093902**

**Accel X Scale Factor: 1.001197**

**Accel Y Bias: 0.093902**

**Accel Y Scale Factor: 0.999423**

**Accel Z Bias: -0.310150**

**Accel Z Scale Factor: 1.029498**

**Gyro X Bias: -0.069605**

**Gyro Y Bias: 0.038339**

**Gyro Z Bias: -0.018238**

**Mag X Bias: 18.221668**

**Mag X Scale Factor: 1.013987**

**Mag Y Bias: 2.135885**

**Mag Y Scale Factor: 1.013153**

**Mag Z Bias: -31.471630**

**Mag Z Scale Factor: 0.973922**

Which you can now use for your calibration. In this case it will take the form:

|  |
| --- |
| float magxc = (((float)raw\_values[6] \* magScale[0]) - magn\_off[0]) \* magn\_scale[0];  float magyc = (((float)raw\_values[7] \* magScale[1]) - magn\_off[1]) \* magn\_scale[1];  float magzc = (((float)raw\_values[8] \* magScale[2]) - magn\_off[2]) \* magn\_scale[2];  float ax = (((float)raw\_values[1] \* accelScale) - acc\_off[0]/G) \* acc\_scale[0] ;  float ay = (((float)raw\_values[0] \* accelScale) - acc\_off[1]/G) \* acc\_scale[1] ;  float az = ((-(float)raw\_values[2] \* accelScale) - acc\_off[2]/G) \* acc\_scale[2]; |

**where :**

|  |
| --- |
| **//Min - max**  **float acc\_off[] = {0.093902, 0.093902, -0.310150};**  **float acc\_scale[] = {1.001197, 0.999423, 1.029498};**  **float magn\_off[] = {18.221668, 2.135885, -31.471630};**  **float magn\_scale[] = {1.013987, 1.013153, 0.973922};** |

**ACCELROMETERS**

**Calibration of accelerometers as similar to that of the magnetometers as mentioned earlier.**

**FreeIMU\_GUI provides accelerometer calibration using ellipsoid fitting just like for magnetometers.**

**Magneto 1.2 supposedly can do the same provided:**

*For Magneto you might again need to pre-process your specific raw accelerometer output, taking into account the bit depth and G sensitivity, to convert the data into milliGalileo.*Taken from <https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/>

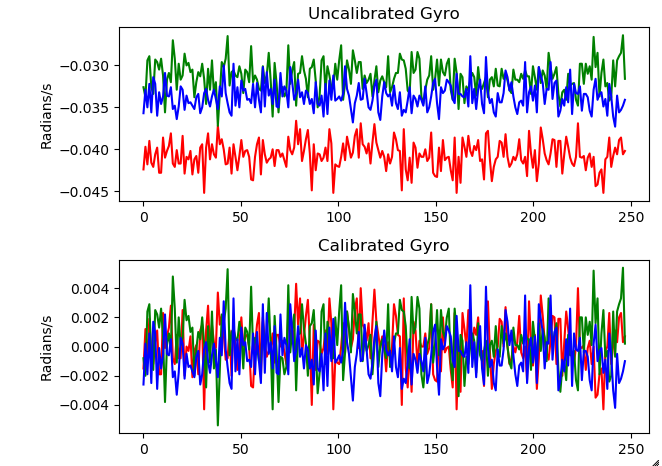
**6-point Calibration can also be used for accelerometers and the sketch attached above includes calibration option for the accelerometer.**

**GYROSCOPE**

**Adafruit is a great source of info on many topics and they have great topics. For instance in the case of gyroscope calibration** [Adafruit SensorLab - Gyroscope Calibration by lady ada](https://learn.adafruit.com/adafruit-sensorlab-gyroscope-calibration/gyroscope-calibration) **they describe calibration as:**

***When gyros are manufactured, they have some zero-offset error, much like magnetometer hard-offset error, that can make measurements difficult. It's easy to detect and remove this offset, we just take many measurements and look for the 'offset' from zero***

***For example, here's a common MPU-6050. If we put it down flat on a table, and take measurements, we will see that neither X, Y or Z (red, green, blue) are at zero. Even though its not moving! That's the zero offset. In this case, its about -0.43 for X, 0.32 for Y and 0.34 for Z.***



***Once calibrated, you can see that there's still a little noise (there always is!) but its only 10% as much as the offset, so we'll get much better measurements. We can try to improve the noisy measurements with filtering if necessary.***

**Even better Lady Ada compares performance of many gyroscopes:** [Comparing Gyroscopes | Adafruit SensorLab - Gyroscope Calibration | Adafruit Learning System](https://learn.adafruit.com/adafruit-sensorlab-gyroscope-calibration/comparing-gyroscopes).

***We compare two basic measurements - the zero offset and the zero noise. Zero offset is easy to correct for, and you should calibrate your gyroscope once its mounted in the final project/PCB - simply take a lot of measurements, find the offset that would bring the gyro to zero. The noise that you get when not moving can be minimized with low pass filtering (sometimes handled in-chip).***

**To implement gyro calibration is relatively easy. In the 6-point calibration sketch it is set up as:**

|  |
| --- |
| /\* estimates the gyro biases \*/  int calibrateGyro() {    // take samples and find bias    \_gxbD = 0;    \_gybD = 0;    \_gzbD = 0;    for (size\_t i=0; i < \_numSamples; i++) {      Imu.Read();      \_gxbD += (Imu.gyro\_x\_radps() + \_gxb)/((double)\_numSamples);      \_gybD += (Imu.gyro\_y\_radps() + \_gyb)/((double)\_numSamples);      \_gzbD += (Imu.gyro\_z\_radps() + \_gzb)/((double)\_numSamples);      delay(20);    }    \_gxb = (float)\_gxbD;    \_gyb = (float)\_gybD;    \_gzb = (float)\_gzbD;    return 1;  } |

**Basically it takes the number of specified samples and performs an average of scaled gyro data. Once you have that you can simply subtract it from the scaled gyro values.**

**MPU AXIS ALIGNMENT**

**I think Kris Winner in one of the Issues on his Github page (**[**https://github.com/kriswiner/MPU9250/issues/345**](https://github.com/kriswiner/MPU9250/issues/345)**) for the MPU-9250 probably explains it best.**

|  |
| --- |
| *The Madgwick and Mahony filters (and quaternions in general I believe) work*  *in a right-handed coordinate system. So the data have to be "provided" to*  *conform to this. Thus NED, ENU (the two most common orientation*  *conventions) or even NWU will all work. As long as the sensor data is*  *provided in a manner consistent with the chosen convention.*  *So first step, the user decides which edge of the sensor board will be*  *pointing to true North when the quaternions read 1 0 0 0. This is one of*  *two absolute references in the system (the other is gravity). Once the*  *board edge facing North is decided (yes, it is your choice!), then it is a*  *simply matter to find out which accel axis point North, no? Then West, then*  *Up. Then the filter should get the data as AN, AW, AU...same for the other*  *two sensors.*  *Then once you do this, and assuming you have chosen the sensor*  *configurations reasonably, calibrated the sensors sufficiently, are running*  *the fusioniterations sufficiently fast, you have taken into account the*  *local magnetic declination, etc you can check the behavior of your absolute*  *orientation engine. What do the quaternions say when the sensors are*  *pointed North (Yaw or Heading = 0)? Should be 1 0 0 0. Try turnin the board*  *90 degrees. What happens now? etc.* |

**When I asked a similar question to X-IO Technologies (Sebastian Madgwick) this is the answer I got**

|  |
| --- |
| *It does not matter if you align the magnetometer with the other sensors; align the other sensors with the magnetometer; or align all sensors to some arbitrary axes. Choose your XYZ axes directions and then use measurements corresponding to those directions.* |

**Think the only caveat I would add is that whatever axis you choose should follow the right handed rule. For more on Coordinate Systems check Appendix 5.**

**Asking CoPilot the same question:**

|  |
| --- |
| *For accurate orientation estimation using Madgwick and Mahony filters, proper axis alignment of the gyroscope, accelerometer, and magnetometer is crucial. Here are some key points to consider:*   1. ***Consistent Coordinate System****: Ensure all sensors (gyroscope, accelerometer, and magnetometer) are aligned to the same coordinate system. Typically, this means aligning the x, y, and z axes of each sensor.* 2. ***Calibration****: Perform a thorough calibration of each sensor to correct for any biases, scale factors, and misalignments. This includes:*    * ***Accelerometer****: Calibrate for offset and scale.*    * ***Gyroscope****: Calibrate for bias (drift) and scale.*    * ***Magnetometer****: Calibrate for hard and soft iron distortions.* 3. ***Mounting Orientation****: Physically mount the sensors in such a way that their axes are aligned. This often involves ensuring that the sensor boards are mounted parallel to each other.* 4. ***Sensor Fusion Algorithm****: Both Madgwick and Mahony filters use quaternion-based sensor fusion algorithms to combine the data from the accelerometer, gyroscope, and magnetometer. Proper axis alignment ensures that the data from each sensor contributes correctly to the orientation estimation.* 5. ***Reference Frames****: Be consistent with the reference frames used. For example, if using a North-East-Down (NED) frame, ensure all sensor data is transformed accordingly.*   [*For more detailed guidance, you can refer to resources like the MathWorks documentation on sensor fusion1*](https://www.mathworks.com/videos/sensor-fusion-part-2-fusing-a-mag-accel-and-gyro-to-estimate-orientation-1569411056638.html) [*and the MDPI paper on joint calibration and axes alignment2*](https://www.mathworks.com/help/nav/ref/ahrs.html)*.*  [*1*](https://www.mathworks.com/videos/sensor-fusion-part-2-fusing-a-mag-accel-and-gyro-to-estimate-orientation-1569411056638.html)*:* [*MathWorks - Fusing a Mag, Accel, and Gyro to Estimate Orientation*](https://www.mathworks.com/videos/sensor-fusion-part-2-fusing-a-mag-accel-and-gyro-to-estimate-orientation-1569411056638.html) [*2*](https://www.mathworks.com/help/nav/ref/ahrs.html)*:* [*MDPI - Accelerometer and Magnetometer Joint Calibration and Axes Alignment*](https://www.mdpi.com/2227-7080/8/1/11) |

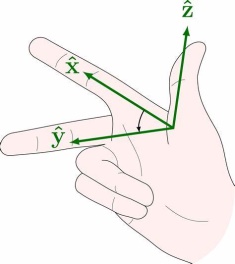
**Now that is out of the way just some basic reminders for looking at axis alignment and the right handed rule:**

**From wikibooks:**

|  |
| --- |
| **Answer:** Vectors in the plane of the page are drawn as arrows on the page. A vector that goes into the plane of the screen is typically drawn as circles with an inscribed X. A vector that comes out of the plane of the screen is typically drawn as circles with dots at their centers. The X is meant to represent the fletching on the back of an arrow or dart while the dot is meant to represent the tip of the arrow.  [https://upload.wikimedia.org/wikipedia/commons/thumb/c/cb/Notation_for_vectors_in_or_out_of_a_plane.svg/220px-Notation_for_vectors_in_or_out_of_a_plane.svg.png](https://en.wikibooks.org/wiki/File:Notation_for_vectors_in_or_out_of_a_plane.svg) |

**Right Handed Rule – from Wikipedia**

|  |  |  |
| --- | --- | --- |
| *For****right-handed****coordinates, if the thumb of a person's right hand points along the z-axis in the positive direction (third coordinate vector), then the fingers curl from the positive x-axis (first coordinate vector) toward the positive y-axis (second coordinate vector). When viewed at a position along the positive z-axis, the ¼ turn from the positive x- to the positive y-axis is****counter-clockwise****.*  ***For right-handed coordinates, use the right hand. For left-handed coordinates, use the left hand.*** | | |
| ***Axis/vector*** | ***Two fingers and thumb*** | ***Curled fingers*** |
| *x (or first vector)* | *First or index* | *Fingers extended* |
| *y (or second vector)* | *Second finger or palm* | *Fingers curled 90°* |
| *z (or third vector)* | *Thumb* | *Thumb* |

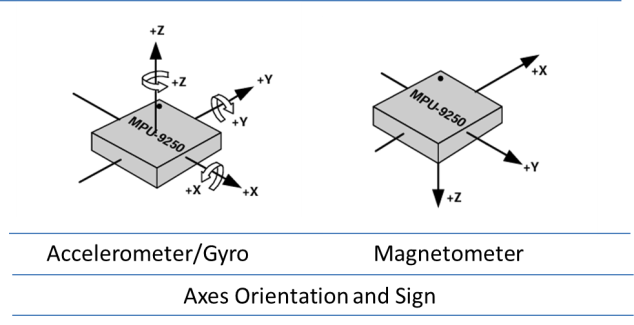


=====================================================================

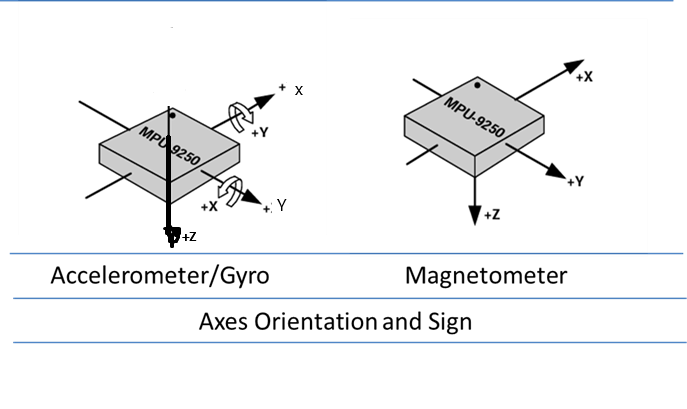
**MPU-9250**

* **Triple-axis MEMS gyroscope with user-programmable full-scale range of ±250 dps, ±500 dps, ±1000 dps, and ±2000 dps**
* **Triple-axis MEMS accelerometer with programmable full scale range of ±2g, ±4g, ±8g, and ±16g**
* **Triple-axis silicon monolithic Hall-effect magnetic sensor with full scale measurement range to ±4900 µT**

**Now with the above in mind lets look as the library axis rotation selected. According the *MPU-9250* data sheet the axes for the accel/gyro and magnetometer are:**

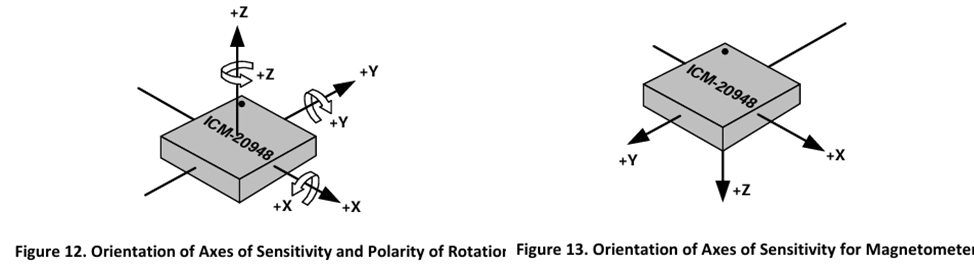


**The library is assuming that the x- and y-axis of the accel/gyro are swapped and the z-axis is negated. That would mean we are aligning those axes with the Magnetometer which is a NED frame of reference aligned with the body frame.**



**ICM-20948**

* Triple-axis MEMS gyroscope with user-programmable full-scale range of ±250 dps, ±500 dps, ±1000 dps, and ±2000 dps
* Triple-axis MEMS accelerometer with programmable full scale range of ±2g, ±4g, ±8g, and ±16g
* Triple-axis silicon monolithic Hall-effect magnetic sensor with full scale measurement range to ±4900 µT

**Again looking at the data sheet you see that it accel/gyro and magnetometer have the same axis orientation:**

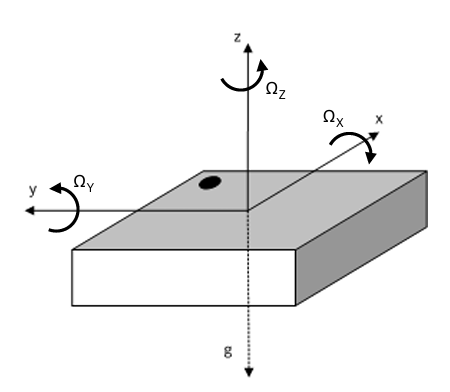
**So the library assumes that the x- and y-axis of the accel/gyro are swapped and the z-axis is negated. That would mean we are aligning those axes with the Magnetometer which is a NED frame of reference aligned with the body frame.**

**Arduino Nano 33 BLE Sense v2**

**The Arduino Nano 33 BLE Sense V2 uses the BMI270 Accel/Gyro and the BMM150 Magnetometer from Bostec.**

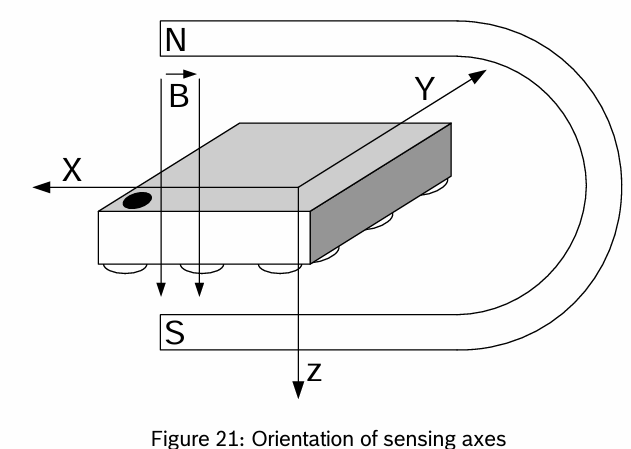
**The BMI270 has the following characteristics:**

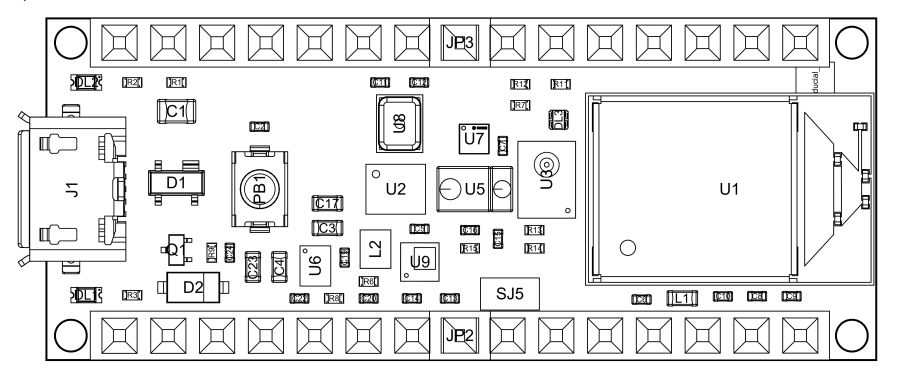
* 16-bit 3-axis accelerometer with ±2g/±4g/±8g/±16g range
* 16-bit 3-axis gyroscope with ±125dps/±250dps/±500dps/±1000dps/±2000dps range



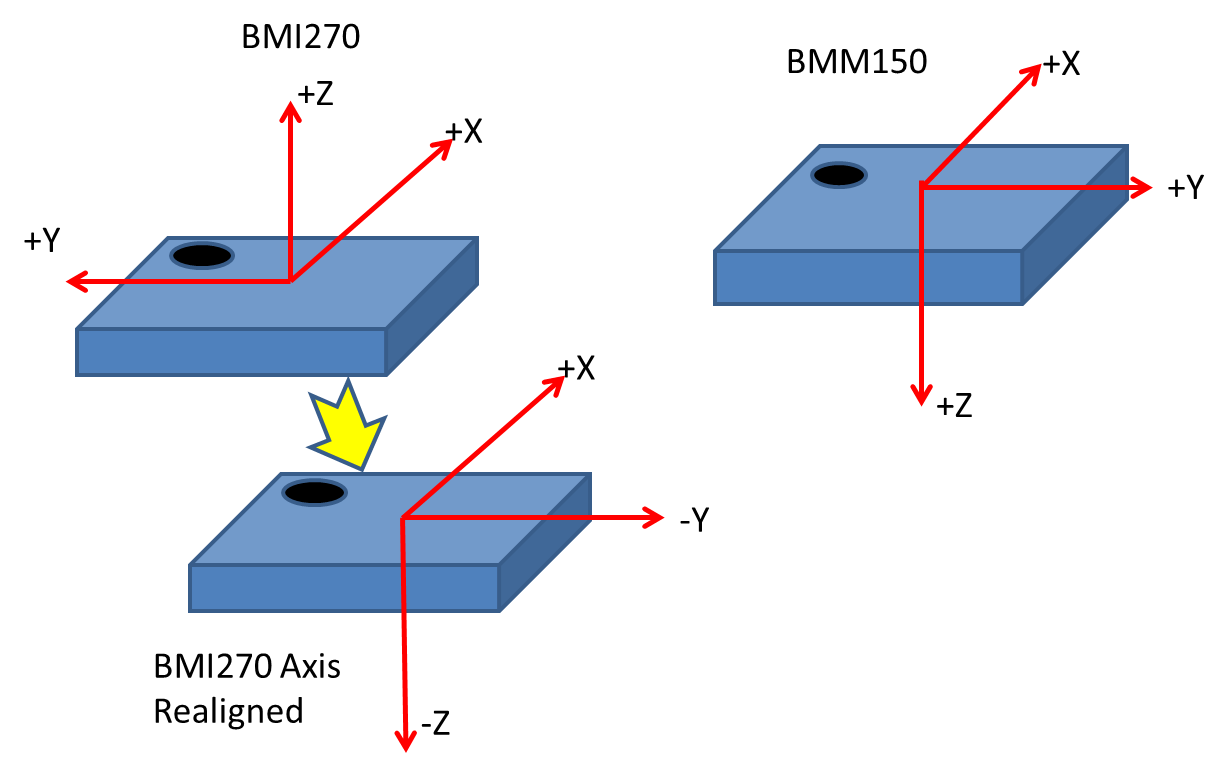
**The BMM150 Magnetometer:**

* ±1300µT (x, y-axis),
* ±2500µT (z-axis)
* Magnetic field resolution of ~0.3µT



**The board itself has the chips oriented as follows (U2 – BMI270, U7 – BMM150:**

**To align the sensor axes in a NED system:**



**realigned , x = x, y = -y, z = -z. Now if you align the mag with the Accel/gyro madgwick update would be:** MadgwickUpdate(gx, gy, gz, ax, ay, az, mx, -my, -mz, deltat);.

**MPU-9250 Calibration Comparisons.**

**The comparisons were done using the freeIMU\_Gui sketch and adding a 5 options for no calibration and the calibration methods discussed above.**

**For each the sketch takes an average of 100 readings while the MPU-9250 is motion less and then normalizes the results for comparison.**

**The same data that was used for FreeIMU\_gui was used by Magneto 1.2 with the exception that the data was converted to g’s and uT’s from the Cnts recorded by FreeIMU\_GUI.**

**NOTE: Experimented with using Magneto 1.2 to calibrate the accelerometer but don’t think I did it right. The data is close only in one case compared to Min/Max and FreeIMU calibration.**

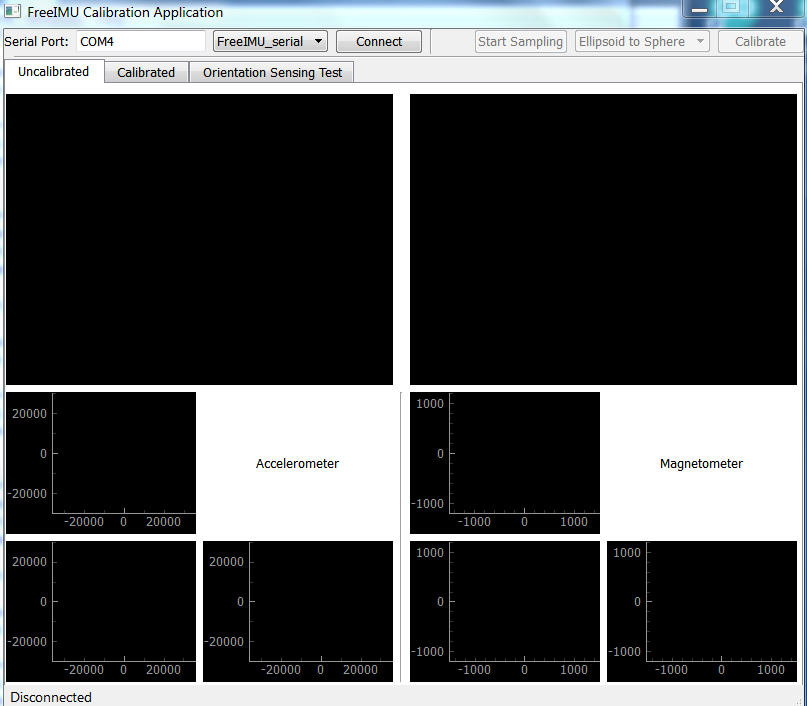
**Which one is more accurate than the others ??????**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ****Calibration**** | ****Ax**** | ****Ay**** | ****Az**** | ****Mx**** | ****My**** | ****Mz**** |
| ****No Calibration**** | **-0.022759** | **-0.100129** | **-1.055997** | **0.665429** | **0.722207** | **0.188737** |
| ****FreeIMU**** | **-0.037976** | **-0.082286** | **-0.983792** | **0.135063** | **0.509187** | **0.851521** |
| ****Motion Cal**** | **0** | **0** | **0** | **0.120326** | **0.490594** | **0.863040** |
| ****Magneto 1.2**** | **0.005139** | **-0.035185** | **-0.989124** | **0.124463** | **0.515762** | **0.847643** |
| ****Min/Max**** | **-0.032398** | **-0.108801** | **-1.054241** | **0.122629** | **0.532700** | **0.837373** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ****Calibration**** | ****Ax**** | ****Ay**** | ****Az**** | ****Mx**** | ****My**** | ****Mz**** |
| No Cal | 0.042984 | 0.996876 | -0.049605 | 0.549364 | -0.687921 | -0.474304 |
| FreeIMU Avg | 0.027684 | 1.015095 | 0.012024 | 0.30728 | -0.945406 | 0.125361 |
| MotionCal | 0 | 0 | 0 | 0.29012 | -0.948288 | 0.128762 |
| Magneto1.2 | -0.008669 | 0.042733 | -0.00371 | 0.289586 | -0.950179 | 0.115324 |
| Min/Max | 0.035453 | 0.985617 | -0.035038 | 0.278928 | -0.953162 | 0.116967 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ****Calibration**** | ****Ax**** | ****Ay**** | ****Az**** | ****Mx**** | ****My**** | ****Mz**** |
| No Cal | 1.007872 | -0.07482 | -0.005015 | -0.730439 | -0.144621 | -0.667491 |
| FreeIMU Avg | 0.987631 | -0.055896 | 0.056252 | -0.971967 | -0.172845 | 0.226426 |
| MotionCal | 0 | 0 | 0 | -0.946564 | -0.196635 | 0.255639 |
| Magneto1.2 | 0.01081 | 1.007685 | 0.045439 | -0.95309 | -0.18246 | 0.241509 |
| Min/Max | 0.999335 | -0.083722 | 0.01862 | -0.961202 | -0.16257 | 0.22285 |

**Appendix 1. FreeIMU GUI**



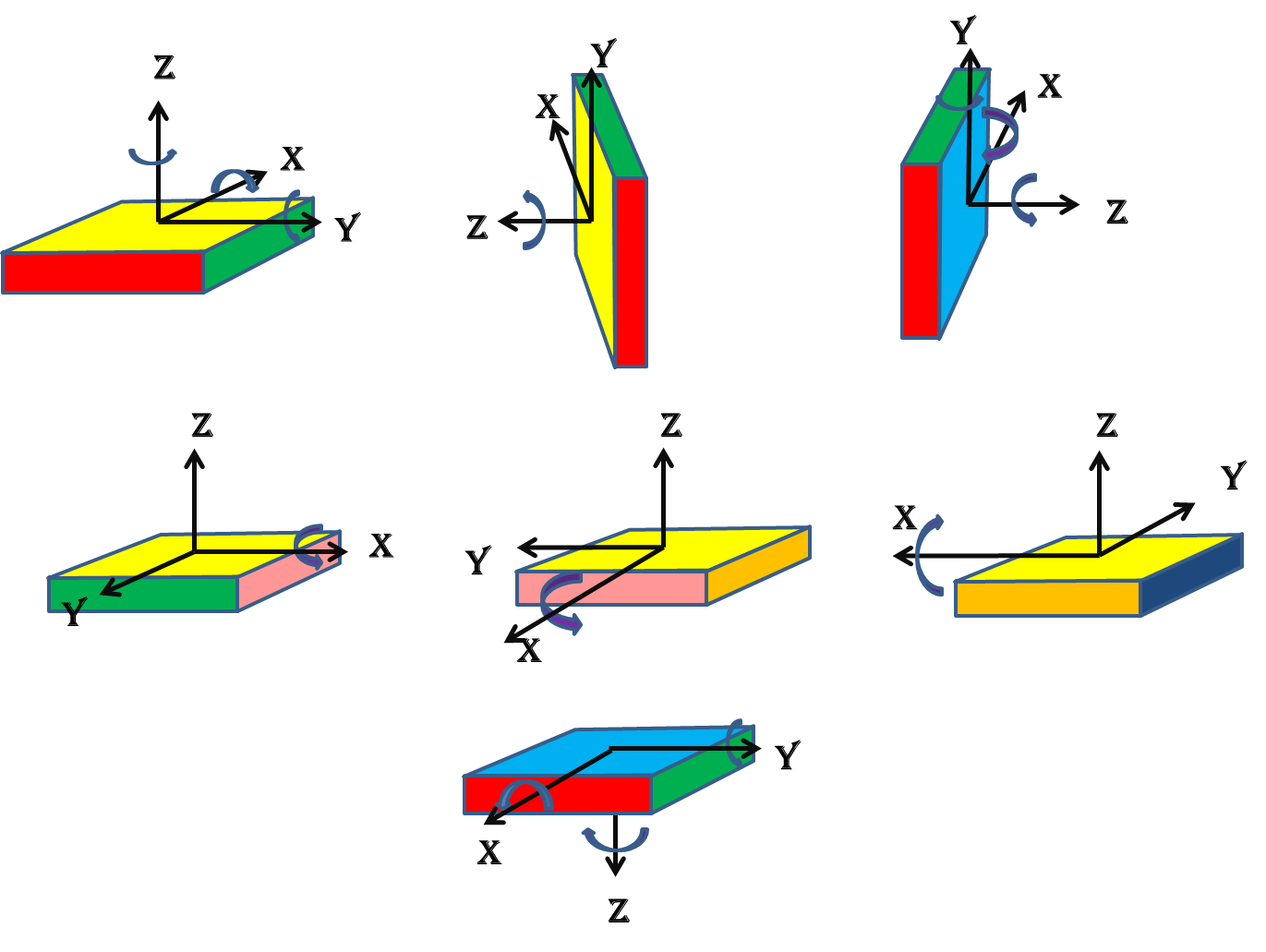
To use:

1.Type the com port for the Arduino in the “Serial Port” box

2.Click “Connect”

3.Once connected click on “Start Sampling”. This will start the GUI sending requests for data from the FreeIMU\_serial sketch that you loaded previously.

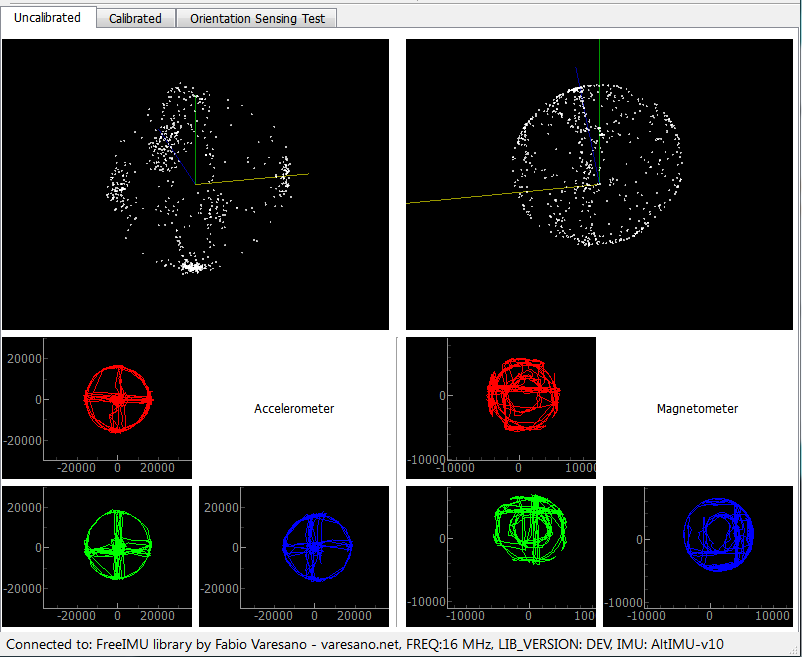
4.I have had the most luck with calibrations that follow the rotation scheme. Just remember to rotate the board slowly and steadily around the axes:



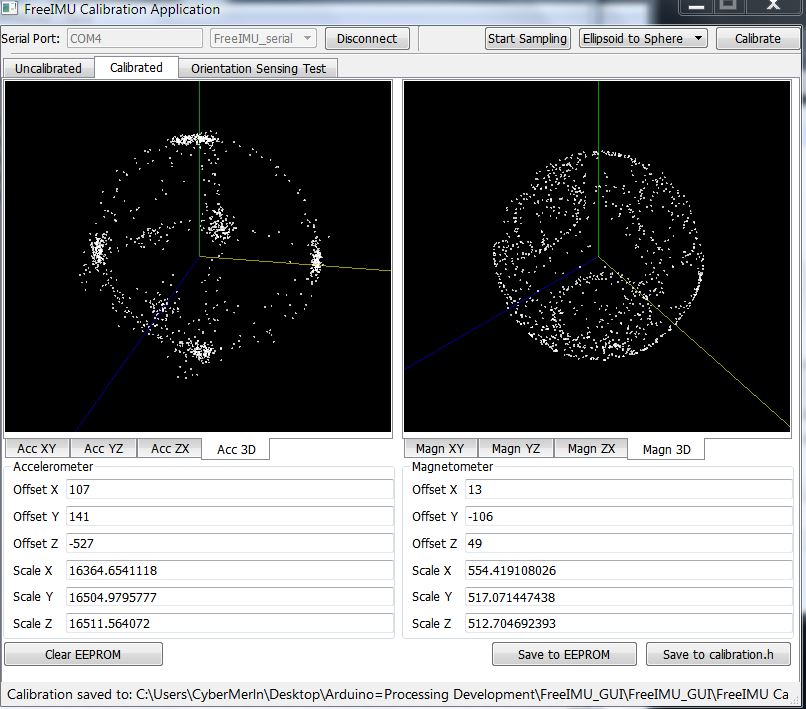
NOTE: Once you start sampling you can put the cursor in anyone of windows and drag the frame around until you center the data points in the window. Also, if you use the scroll wheel on your mouse you can zoom in or out. Helpful when the data isn’t centered around 0, 0 which it won’t be.

The idea is to collect enough data to fully define the sphere/ellipsoid which is done by rotations around all the axes.

Once you are satisfied with your data click on “Stop Sampling” and you should see something like this:



Then click the “Calibrate” button. This will bring you to the screen with the calibration values for the accelerometer and magnetometer. If you select the Accel 3D and Magn 3d tabs on the calibration screen you should wind up seeing something like the below figure:



One last step and you will have your calibration finished. Click on “Save to calibration.h”, select the folder you want to save the calibration file (libraries->FreIMU->calibration.h), and then save it. That’s it for calibration.

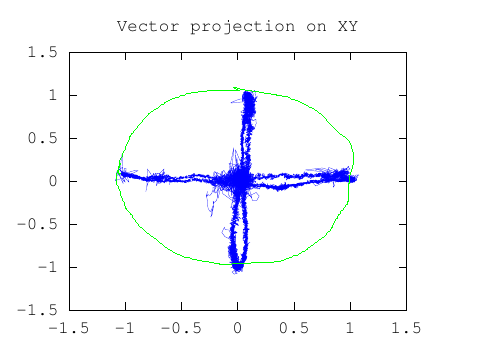
Reload the FreeIMU sketch. That’s it. Re-test with the Free\_cube\_ODO processing sketch.

Additional notes from Fabio’s website (Now Defunct) that you might find interesting during your calibration process:

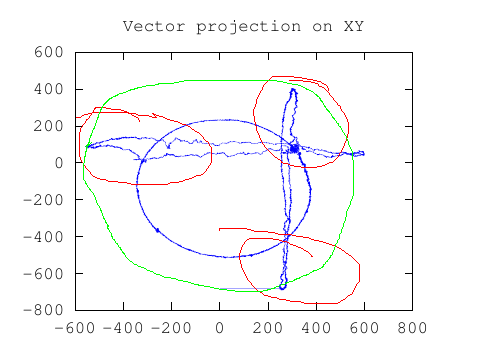
Hi Manor, thanks for sharing Submitted by fabio on Thu, 2012-11-15 09:31. Hi Manor, thanks for sharing your video. However, this is still not a proper calibration approach.

You are not making a circle on the acc XY plane and you are not making the biggest circles on the magnetometer. Both these are crucial to get better results.

This is what you have in your XY plane for the accelerometer:



This is wrong. You did not made complete circle on this one. The green line is what it should look like for proper calibration.



This is what you have for the XY plane for the magnetomter. This is also wrong.. See those areas marked in red by myself? These are synthoms of the fact that you didn't make the biggest circle possible... the green line shows you how this should look like.

Rotation XY plane Submitted by Sjef (not verified) on Thu, 2012-11-15 09:45.

Hello Manor,

In addition to what Fabio said:

You have to rotate around the 4 sides of the box too.

In a way that each side of the box has touched the "ground".

You get the biggest circle in the XY plane when you do this.

The last movement you did in the video (movement of rotating the bottom of the IMU box on the "ground") isn't necesarry.

Hope this helps,

**Appendix 2. Basic Arduino Sketch for FreeIMU Calibration.**

|  |
| --- |
| // Include Modified Bolderflight invensense-imu library  #include "mpu9250.h"  /\* Mpu9250 object \*/  bfs**::**Mpu9250 imu**;**  uint32\_t time\_old **=** 0**;**  //Gravity  #define G 9.80665  #define data\_count 100  // scale factors  float accelScale**,** gyroScale**;**  float magScale**[**3**];**  void setup**()** **{**  /\* Serial to display data \*/  **while(!**Serial **&&** millis**()** **<** 5000**)** **{}**  Serial**.**begin**(**115200**);**    // If Teensy 4.x fails print Crashreport to serial monitor  **if** **(**CrashReport**)** **{**  Serial**.**print**(**CrashReport**);**  Serial**.**println**(**"Press any key to continue"**);**  **while** **(**Serial**.**read**()** **!=** **-**1**)** **{**  **}**  **while** **(**Serial**.**read**()** **==** **-**1**)** **{**  **}**  **while** **(**Serial**.**read**()** **!=** **-**1**)** **{**  **}**  **}**  /\* Start the I2C bus \*/  Wire2**.**begin**();**  Wire2**.**setClock**(**400000**);**  /\* I2C bus, 0x68 address \*/  imu**.**Config**(&**Wire2**,** bfs**::**Mpu9250**::**I2C\_ADDR\_PRIM**);**  /\* Initialize and configure IMU \*/  **if** **(!**imu**.**Begin**())** **{**  Serial**.**println**(**"Error initializing communication with IMU"**);**  **while(**1**)** **{}**  **}**  /\* Set the sample rate divider \*/  // rate = 1000 / (srd + 1)  // = 1000/20 = 50 hz  // = 100 hz  **if** **(!**imu**.**ConfigSrd**(**9**))** **{**  Serial**.**println**(**"Error configured SRD"**);**  **while(**1**)** **{}**  **}**  /\* Accelerometer options  ACCEL\_RANGE\_2G  ACCEL\_RANGE\_4G  ACCEL\_RANGE\_8G  ACCEL\_RANGE\_16G (Default)  \*/  //imu.ConfigAccelRange(bfs::Mpu9250::ACCEL\_RANGE\_16G);  /\* Gyroscope Options  GYRO\_RANGE\_250DPS  GYRO\_RANGE\_500DPS  GYRO\_RANGE\_1000DPS  GYRO\_RANGE\_2000DPS  \*/  //imu.ConfigGyroRange(bfs::Mpu9250::GYRO\_RANGE\_2000DPS);  //Get MPU sensitivity values  imu**.**getScales**(&**accelScale**,** **&**gyroScale**,** magScale**);**  Serial**.**print**(**"Accelerometer Scale: "**);** Serial**.**println**(**accelScale**,** 5**);**  Serial**.**print**(**"Gyro Scale: "**);** Serial**.**println**(**gyroScale**,** 5**);**  Serial**.**println**(**"Magnetometer Scales:"**);**  Serial**.**print**(**"\tMagx: "**);** Serial**.**println**(**magScale**[**0**],**5**);**  Serial**.**print**(**"\tMagy: "**);** Serial**.**println**(**magScale**[**1**],**5**);**  Serial**.**print**(**"\tMagz: "**);** Serial**.**println**(**magScale**[**2**],**5**);**  **}**  //The command from the PC  char cmd**;**  float values**[**9**];**  float values1**[**10**];**  int16\_t raw\_values**[**9**];**  char str**[**128**];**  char str1**[**128**];**  void loop**()** **{**  char cmd **=** '0'**;**  **if(**Serial**.**available**())** **{**  cmd **=** Serial**.**read**();**  **switch(**cmd**)** **{**  **case** 'v'**:** // Send handshake  **{**  //sprintf(str, "FreeIMU library by FREQ: LIB\_VERSION: %s", FREEIMU\_LIB\_VERSION);  Serial**.**print**(**"OK...."**);**  Serial**.**print**(**'\n'**);**  **break;**  **}**  **case** 'b'**:** // Send packed values to GUI  **{**  uint8\_t count **=** serial\_busy\_wait**();**  uint8\_t i\_count **=** 0**;**  **while(**i\_count **<** count**)** **{**  imu**.**Read\_raw**(**raw\_values**);**  **if(**imu**.**new\_imu\_data**())** **{**  i\_count **+=** 1**;**  writeArr**(**raw\_values**,** 9**,** **sizeof(**int16\_t**));**  //writeArr(raw\_values, 6, sizeof(int));  Serial**.**println**();**  **}**  **}**  **break;**  **}**  **case** 'w'**:** //Send word size to GUI  **{**  //Serial.println(sizeof(int16\_t));  Serial**.**write**(sizeof(**int16\_t**));** //in this case the size of an int16.  **break;**  **}**  **case** 'r'**:** // Option to print raw values to serial monitor  **{**  uint8\_t count **=** serial\_busy\_wait**();**  uint8\_t i\_count **=** 0**;**  //for(uint8\_t i=0; i<count; i++) {  **while(**i\_count **<** count**)** **{**  //my3IMU.getUnfilteredRawValues(raw\_values);  **if(**imu**.**Read\_raw**(**raw\_values**)){**  **if(**imu**.**new\_imu\_data**())** **{**  i\_count **+=** 1**;**  sprintf**(**str**,** "%d,%d,%d,%d,%d,%d,%d,%d,%d,"**,** raw\_values**[**0**],** raw\_values**[**1**],** raw\_values**[**2**],** raw\_values**[**3**],** raw\_values**[**4**],** raw\_values**[**5**],** raw\_values**[**6**],** raw\_values**[**7**],** raw\_values**[**8**]);**  Serial**.**print**(**str**);**  Serial**.**print**(**millis**());** Serial**.**print**(**","**);**  Serial**.**println**(**"\r\n"**);**  **}**  **}**  **}**  **break;**  **}**  **default:**  **break;**  **}**  **while** **(**Serial**.**read**()** **!=** **-**1**)**  **;** // lets strip the rest out  **}**  **}**  char serial\_busy\_wait**()** **{**  **while(!**Serial**.**available**())** **{**  **;** // do nothing until ready  **}**  **return** Serial**.**read**();**  **}**  //From FreeIMU library  void writeArr**(**void **\*** varr**,** uint8\_t arr\_length**,** uint8\_t type\_bytes**)** **{**  byte **\*** arr **=** **(**byte**\*)** varr**;**  **for(**uint8\_t i**=**0**;** i**<**arr\_length**;** i**++)** **{**  writeVar**(&**arr**[**i **\*** type\_bytes**],** type\_bytes**);**  **}**  **}**  // thanks to Francesco Ferrara and the Simplo project for the following code!  void writeVar**(**void **\*** val**,** uint8\_t type\_bytes**)** **{**  byte **\*** addr**=(**byte **\*)(**val**);**  **for(**uint8\_t i**=**0**;** i**<**type\_bytes**;** i**++)** **{**  Serial**.**write**(**addr**[**i**]);**  **}**  **}** |

**Appendix 3. MotionCal Calibration Sketch for MPU-9250**

|  |
| --- |
| #include "mpu9250.h"  /\* Mpu9250 object \*/  bfs**::**Mpu9250 imu**;**  uint32\_t time\_old **=** 0**;**  #include <Wire.h>  #include <EEPROM.h>  #include <util/crc16.h>  const int ledPin **=** 13**;**  int ledState **=** LOW**;**  int ledFastblinks **=** 0**;**  elapsedMillis ledMillis **=** 0**;**  int loopcount **=** 0**;**  void receiveCalibration**();**  void setup**()** **{**  Serial**.**begin**(**115200**);**  **while** **(!**Serial**)** **;** // wait for serial port open  delay**(**800**);**  Wire2**.**begin**();**  Wire2**.**setClock**(**400000**);**  /\* I2C bus, 0x68 address \*/  imu**.**Config**(&**Wire2**,** bfs**::**Mpu9250**::**I2C\_ADDR\_PRIM**);**  /\* Initialize and configure IMU \*/  **if** **(!**imu**.**Begin**())** **{**  Serial**.**println**(**"Error initializing communication with IMU"**);**  **while(**1**)** **{}**  **}**  /\* Set the sample rate divider \*/  // rate = 1000 / (srd + 1)  // = 1000/20 = 50 hz  // = 100 hz  **if** **(!**imu**.**ConfigSrd**(**9**))** **{**  Serial**.**println**(**"Error configured SRD"**);**  **while(**1**)** **{}**  **}**  //imu.ConfigAccelRange(bfs::Mpu9250::ACCEL\_RANGE\_4G);  pinMode**(**ledPin**,** OUTPUT**);**  **}**  float accel\_zerog**[**3**],** gyro\_zerorate**[**3**],** mag\_hardiron**[**3**],** mag\_softiron**[**9**];**  float magfield**,** mag\_field**;**  void loop**()** **{**  float raw\_values**[**9**];**  // get and print uncalibrated data  **if(**imu**.**Read**(**raw\_values**)){**  **if(**imu**.**new\_imu\_data**()){**  //imu.readMotionSensor(ax, ay, az, gx, gy, gz, mx, my, mz);  Serial**.**print**(**"Raw:"**);**  Serial**.**print**(**int**(**raw\_values**[**0**]** **\***8192**/**9.805**));**  Serial**.**print**(**','**);**  Serial**.**print**(**int**(**raw\_values**[**1**]** **\***8192**/**9.805**));**  Serial**.**print**(**','**);**  Serial**.**print**(**int**(**raw\_values**[**2**]** **\***8192**/**9.805**));**  Serial**.**print**(**','**);**  Serial**.**print**(**int **(**raw\_values**[**3**]\***16 **));**  Serial**.**print**(**','**);**  Serial**.**print**(**int **(**raw\_values**[**4**]\***16 **));**  Serial**.**print**(**','**);**  Serial**.**print**(**int **(**raw\_values**[**5**]\***16 **));**  Serial**.**print**(**','**);**  Serial**.**print**(**int **(**raw\_values**[**6**]\***10**));**  Serial**.**print**(**','**);**  Serial**.**print**(**int **(**raw\_values**[**7**]\***10**));**  Serial**.**print**(**','**);**  Serial**.**print**(**int **(**raw\_values**[**8**]\***10**));**  Serial**.**println**(**""**);**  loopcount **=** loopcount **+** 1**;**  **}**  **}**  // check for incoming calibration  receiveCalibration**();**  // occasionally print calibration  **if** **(**loopcount **==** 50 **||** loopcount **>** 100**)** **{**  Serial**.**print**(**"Cal1:"**);**  **for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**  SerialUSB1**.**print**(**accel\_zerog**[**i**],** 3**);**  SerialUSB1**.**print**(**","**);**  **}**  **for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**  SerialUSB1**.**print**(**gyro\_zerorate**[**i**],** 3**);**  SerialUSB1**.**print**(**","**);**  **}**  **for** **(**int i**=**0**;** i**<**3**;** i**++)** **{**  SerialUSB1**.**print**(**mag\_hardiron**[**i**],** 3**);**  SerialUSB1**.**print**(**","**);**  **}**  SerialUSB1**.**println**(**mag\_field**,** 3**);**  loopcount**++;**  **}**  **if** **(**loopcount **>=** 100**)** **{**  Serial**.**print**(**"Cal2:"**);**  **for** **(**int i**=**0**;** i**<**9**;** i**++)** **{**  SerialUSB1**.**print**(**mag\_softiron**[**i**],** 4**);**  **if** **(**i **<** 8**)** SerialUSB1**.**print**(**','**);**  **}**  Serial**.**println**();**  loopcount **=** 0**;**  **}**  // blink LED, slow normally, fast when calibration written  **if** **(**ledMillis **>=** 1000**)** **{**  **if** **(**ledFastblinks **>** 0**)** **{**  ledFastblinks **=** ledFastblinks **-** 1**;**  ledMillis **-=** 125**;**  **}** **else** **{**  ledMillis **-=** 1000**;**  **}**  **if** **(**ledState **==** LOW**)** **{**  ledState **=** HIGH**;**  **}** **else** **{**  ledState **=** LOW**;**  **}**  digitalWrite**(**ledPin**,** ledState**);**  **}**  **}**  byte caldata**[**68**];** // buffer to receive magnetic calibration data  byte calcount**=**0**;**  void receiveCalibration**()** **{**  uint16\_t crc**;**  byte b**,** i**;**  **while** **(**Serial**.**available**())** **{**  b **=** Serial**.**read**();**  **if** **(**calcount **==** 0 **&&** b **!=** 117**)** **{**  // first byte must be 117  **return;**  **}**  **if** **(**calcount **==** 1 **&&** b **!=** 84**)** **{**  // second byte must be 84  calcount **=** 0**;**  **return;**  **}**  // store this byte  caldata**[**calcount**++]** **=** b**;**  **if** **(**calcount **<** 68**)** **{**  // full calibration message is 68 bytes  **return;**  **}**  // verify the crc16 check  crc **=** 0xFFFF**;**  **for** **(**i**=**0**;** i **<** 68**;** i**++)** **{**  crc **=** crc16\_update**(**crc**,** caldata**[**i**]);**  **}**  **if** **(**crc **==** 0**)** **{**  // data looks good, use it  float offsets**[**16**];**  memcpy**(**offsets**,** caldata**+**2**,** 16**\***4**);**  accel\_zerog**[**0**]** **=** offsets**[**0**];**  accel\_zerog**[**1**]** **=** offsets**[**1**];**  accel\_zerog**[**2**]** **=** offsets**[**2**];**    gyro\_zerorate**[**0**]** **=** offsets**[**3**];**  gyro\_zerorate**[**1**]** **=** offsets**[**4**];**  gyro\_zerorate**[**2**]** **=** offsets**[**5**];**    mag\_hardiron**[**0**]** **=** offsets**[**6**];**  mag\_hardiron**[**1**]** **=** offsets**[**7**];**  mag\_hardiron**[**2**]** **=** offsets**[**8**];**  mag\_field **=** offsets**[**9**];**    mag\_softiron**[**0**]** **=** offsets**[**10**];**  mag\_softiron**[**1**]** **=** offsets**[**13**];**  mag\_softiron**[**2**]** **=** offsets**[**14**];**  mag\_softiron**[**3**]** **=** offsets**[**13**];**  mag\_softiron**[**4**]** **=** offsets**[**11**];**  mag\_softiron**[**5**]** **=** offsets**[**15**];**  mag\_softiron**[**6**]** **=** offsets**[**14**];**  mag\_softiron**[**7**]** **=** offsets**[**15**];**  mag\_softiron**[**8**]** **=** offsets**[**12**];**  calcount **=** 0**;**  **return;**  **}**  // look for the 117,84 in the data, before discarding  **for** **(**i**=**2**;** i **<** 67**;** i**++)** **{**  **if** **(**caldata**[**i**]** **==** 117 **&&** caldata**[**i**+**1**]** **==** 84**)** **{**  // found possible start within data  calcount **=** 68 **-** i**;**  memmove**(**caldata**,** caldata **+** i**,** calcount**);**  **return;**  **}**  **}**  // look for 117 in last byte  **if** **(**caldata**[**67**]** **==** 117**)** **{**  caldata**[**0**]** **=** 117**;**  calcount **=** 1**;**  **}** **else** **{**  calcount **=** 0**;**  **}**  **}**  **}**  uint16\_t crc16\_update**(**uint16\_t crc**,** uint8\_t a**)**  **{**  int i**;**  crc **^=** a**;**  **for** **(**i **=** 0**;** i **<** 8**;** i**++)** **{**  **if** **(**crc **&** 1**)** **{**  crc **=** **(**crc **>>** 1**)** **^** 0xA001**;**  **}** **else** **{**  crc **=** **(**crc **>>** 1**);**  **}**  **}**  **return** crc**;**  **}** |

**Appendix 4. 6-point Calibration Process for accelerometer**

# ADXL345 Digital Accelerometer

[Programming and Calibration | ADXL345 Digital Accelerometer | Adafruit Learning System](https://learn.adafruit.com/adxl345-digital-accelerometer/programming)

## Calibration Method:

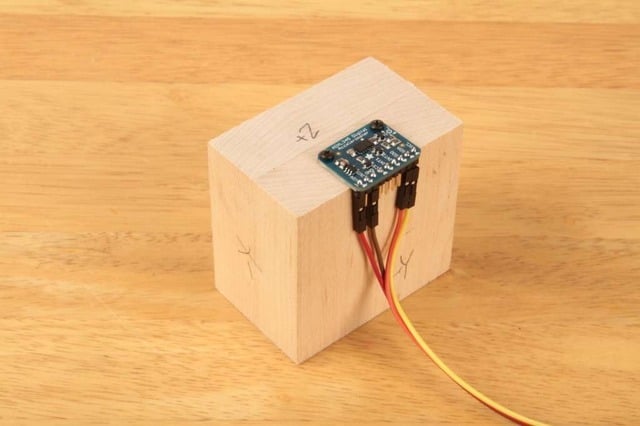
To calibrate the sensor to the gravitational reference, you need to determine the sensor output for each axis when it is precisely aligned with the axis of gravitational pull. Laboratory quality calibration uses precision positioning jigs. The method described here is simple and gives surprisingly good results with just a block of wood.

## Mount the Sensor:

FIrst mount the sensor securely to a block or a box. The size is not important, as long as all the sides are at right angles. The material is not important as long as it is fairly rigid.

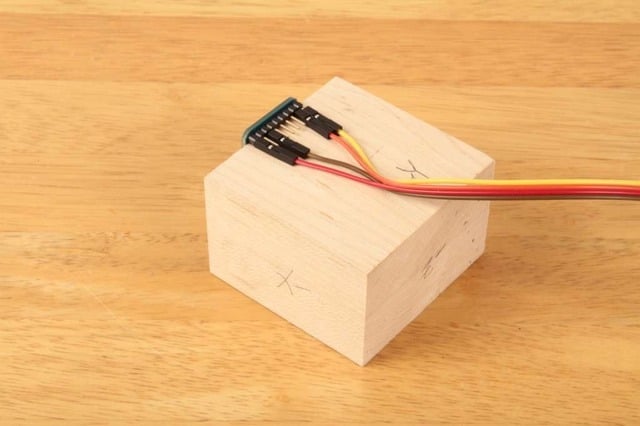
## Load the Calibration Sketch:

Load and run the Calibration sketch below. Open the Serial Monitor and wait for the prompt.

[](https://learn.adafruit.com/assets/6465)

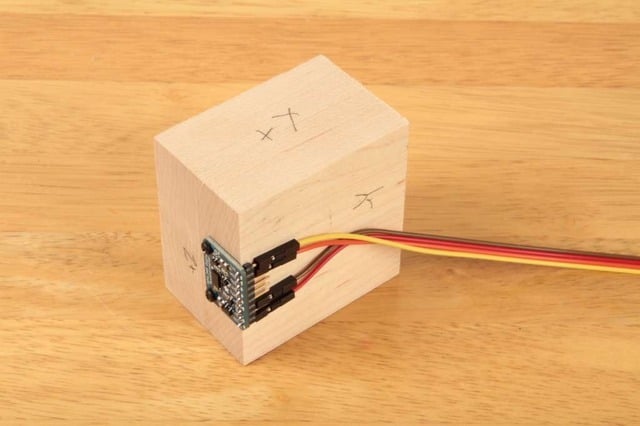
## Position the Block:

Place the block on a firm flat surface such as a sturdy table. Type a character in the Serial Monitor and hit return. The sketch will take a measurement on that axis and print the results.

[](https://learn.adafruit.com/assets/6467)

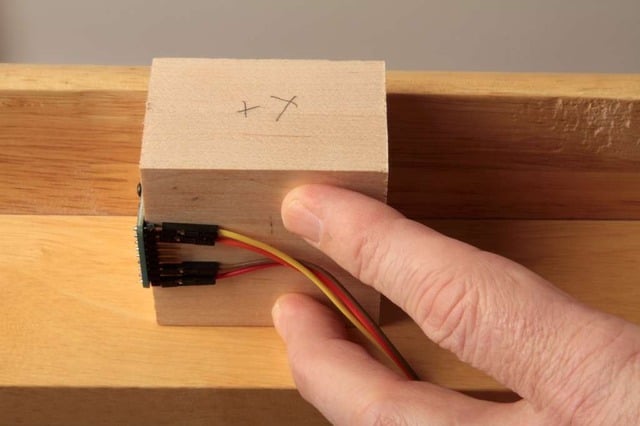
## Reposition the Block:

Turn the block so a different side is flat on the table and type another key to measure that axis.

[](https://learn.adafruit.com/assets/6468)

## Repeat:

Repeat for all six sides of the block to measure the positive and negative aspects of each axis.

[](https://learn.adafruit.com/assets/6469)

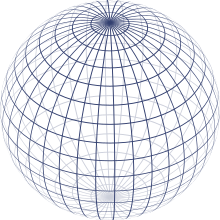
## *(Hint:)*

*For the sides obstructed by the breakout board and/or wires, press the block up against the bottom of the table while taking the reading.*

**Appendix 5. Coordinate Systems from WikiPedia**

## [Earth bounded axes conventions](https://en.wikipedia.org/wiki/Axes_conventions)

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=1)]

[](https://en.wikipedia.org/wiki/File:Sphere_wireframe.svg)Representation of the Earth with parallels and meridians

### World reference frames: ENU and NED

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=2)]

*Main article:*[*Local tangent plane*](https://en.wikipedia.org/wiki/Local_tangent_plane)

Basically, as lab frame or reference frame, there are two kinds of conventions for the frames:

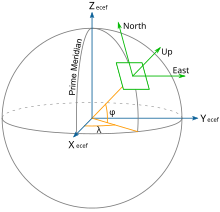
* East, North, Up (ENU), used in geography
* North, East, Down (NED), used specially in aerospace

This frame referenced w.r.t. Global Reference frames like Earth Center Earth Fixed (ECEF) non-inertial system.

#### World reference frames for attitude description

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=3)]

To establish a standard convention to describe attitudes, it is required to establish at least the axes of the reference system and the axes of the rigid body or vehicle. When an ambiguous notation system is used (such as [Euler angles](https://en.wikipedia.org/wiki/Euler_angles)) the convention used should also be stated. Nevertheless, most used notations (matrices and quaternions) are unambiguous.

[](https://en.wikipedia.org/wiki/File:ECEF_ENU_Longitude_Latitude_relationships.svg)Earth Centered Earth Fixed and East, North, Up coordinates.

[Tait–Bryan angles](https://en.wikipedia.org/wiki/Tait%E2%80%93Bryan_angles) are often used to describe a vehicle's attitude with respect to a chosen reference frame, though any other notation can be used. The positive *x*-axis in vehicles points always in the direction of movement. For positive *y*- and *z*-axis, we have to face two different conventions:

* In case of land vehicles like cars, tanks etc., which use the ENU-system (East-North-Up) as external reference (*World frame*), the vehicle's (body's) positive *y*- or pitch axis always points to its left, and the positive *z*- or yaw axis always points up. World frame's origin is fixed at the center of gravity of the vehicle.[[3]](https://en.wikipedia.org/wiki/Axes_conventions#cite_note-3)
* By contrast, in case of air and sea vehicles like submarines, ships, airplanes etc., which use the NED-system (North-East-Down) as external reference (*World frame*), the vehicle's (body's) positive *y*- or pitch axis always points to its right, and its positive *z*- or yaw axis always points down. World frame's origin is fixed at the center of gravity of the vehicle.
* Finally, in case of space vehicles like the [Space Shuttle](https://en.wikipedia.org/wiki/Space_Shuttle) etc., a modification of the latter convention is used, where the vehicle's (body's) positive *y*- or pitch axis again always points to its right, and its positive z- or yaw axis always points down, but “down” now may have two different meanings: If a so-called *local frame* is used as external reference, its positive z-axis points “down” to the center of the Earth as it does in case of the earlier mentioned NED-system,[[4]](https://en.wikipedia.org/wiki/Axes_conventions#cite_note-4) but if the *inertial frame* is used as reference, its positive z-axis will point now to the [north celestial pole](https://en.wikipedia.org/wiki/North_celestial_pole), and its positive x-axis to the Vernal [Equinox](https://en.wikipedia.org/wiki/Equinox)[[5]](https://en.wikipedia.org/wiki/Axes_conventions#cite_note-5) or some other reference meridian.

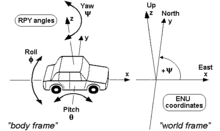
### Frames mounted on vehicles

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=4)]

Specially for aircraft, these frames do not need to agree with the earth-bound frames in the up-down line. It must be agreed what ENU and NED mean in this context.

#### Conventions for land vehicles

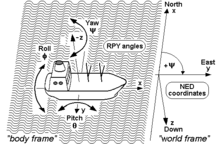
[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=5)]

[](https://en.wikipedia.org/wiki/File:RPY_angles_of_cars.png)RPY angles of cars and other land vehicles

For land vehicles it is rare to describe their complete orientation, except when speaking about [electronic stability control](https://en.wikipedia.org/wiki/Electronic_stability_control) or [satellite navigation](https://en.wikipedia.org/wiki/Satellite_navigation). In this case, the convention is normally the one of the adjacent drawing, where RPY stands for [roll-pitch-yaw](https://en.wikipedia.org/wiki/Roll-pitch-yaw).

#### Conventions for sea vehicles

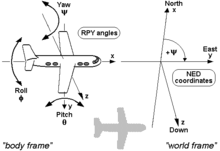
[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=6)]

[](https://en.wikipedia.org/wiki/File:RPY_angles_of_ships.png)RPY angles of ships and other sea vehicles

As well as aircraft, the same terminology is used for the [motion of ships and boats](https://en.wikipedia.org/wiki/Ship_motions). Some words commonly used were introduced in maritime navigation. For example, the *yaw* angle or heading, has a [nautical](https://en.wikipedia.org/wiki/Navigation) origin, with the meaning of "bending out of the course". Etymologically, it is related with the verb 'to go'.[[6]](https://en.wikipedia.org/wiki/Axes_conventions#cite_note-6) It is related to the concept of [bearing](https://en.wikipedia.org/wiki/Bearing_(navigation)). It is typically assigned the shorthand notation *ψ*.[[7]](https://en.wikipedia.org/wiki/Axes_conventions#cite_note-7)

#### Conventions for aircraft local reference frames

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=7)]

[](https://en.wikipedia.org/wiki/File:RPY_angles_of_airplanes.png)RPY angles of airplanes and other air vehicles[](https://en.wikipedia.org/wiki/File:Roll_pitch_yaw_mnemonic.svg)Mnemonics to remember angle names

*Main article:*[*aircraft principal axes*](https://en.wikipedia.org/wiki/Aircraft_principal_axes)

Coordinates to describe an aircraft attitude (Heading, Elevation and Bank) are normally given relative to a reference control frame located in a control tower, and therefore ENU, relative to the position of the control tower on the earth surface.

Coordinates to describe observations made from an aircraft are normally given relative to its intrinsic axes, but normally using as positive the coordinate pointing downwards, where the interesting points are located. Therefore, they are normally NED.

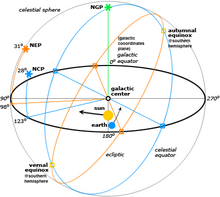
These axes are normally taken so that X axis is the longitudinal axis pointing ahead, Z axis is the vertical axis pointing downwards, and the Y axis is the lateral one, pointing in such a way that the frame is [right-handed](https://en.wikipedia.org/wiki/Right-hand_rule).

The *motion* of an aircraft is often described in terms of rotation about these axes, so rotation about the *X*-axis is called rolling, rotation about the *Y*-axis is called pitching, and rotation about the *Z*-axis is called yawing.

## Frames for space navigation

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=8)]

*Main article:*[*Celestial coordinate system*](https://en.wikipedia.org/wiki/Celestial_coordinate_system)

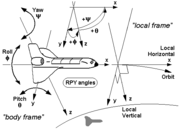
[](https://en.wikipedia.org/wiki/File:Celestial.png)Different reference systems for coordinates in space

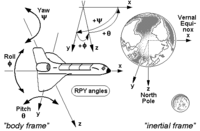
For satellites orbiting the Earth it is normal to use the [Equatorial coordinate system](https://en.wikipedia.org/wiki/Equatorial_coordinate_system). The projection of the Earth's equator onto the celestial sphere is called the [celestial equator](https://en.wikipedia.org/wiki/Celestial_equator). Similarly, the projections of the Earth's north and south geographic poles become the north and south [celestial poles](https://en.wikipedia.org/wiki/Celestial_pole), respectively.

Deep space satellites use other [Celestial coordinate system](https://en.wikipedia.org/wiki/Celestial_coordinate_system), like the [Ecliptic coordinate system](https://en.wikipedia.org/wiki/Ecliptic_coordinate_system).

### Local conventions for space ships as satellites

[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=9)]

[](https://en.wikipedia.org/wiki/File:RPY_angles_of_spaceships_(local_frame).png)

[](https://en.wikipedia.org/wiki/File:RPY_angles_of_spaceships_(inertial_frame).png)

RPY angles of the Space Shuttle and other space vehicles, first using a local frame as reference and second using an inertial frame as reference.

If the goal is to keep the shuttle during its orbits in a constant attitude with respect to the sky, e.g. in order to perform certain astronomical observations, the preferred reference is the *inertial frame*, and the RPY angle vector (0|0|0) describes an attitude then, where the shuttle's wings are kept permanently parallel to the Earth's equator, its nose points permanently to the vernal [equinox](https://en.wikipedia.org/wiki/Equinox), and its belly towards the northern [polar star](https://en.wikipedia.org/wiki/Polar_star) (see picture). (Note that rockets and missiles more commonly follow the conventions for aircraft where the RPY angle vector (0|0|0) points north, rather than toward the vernal equinox).

On the other hand, if the goal is to keep the shuttle during its orbits in a constant attitude with respect to the surface of the Earth, the preferred reference will be the *local frame*, with the RPY angle vector (0|0|0) describing an attitude where the shuttle's wings are parallel to the Earth's surface, its nose points to its heading, and its belly down towards the centre of the Earth (see picture).

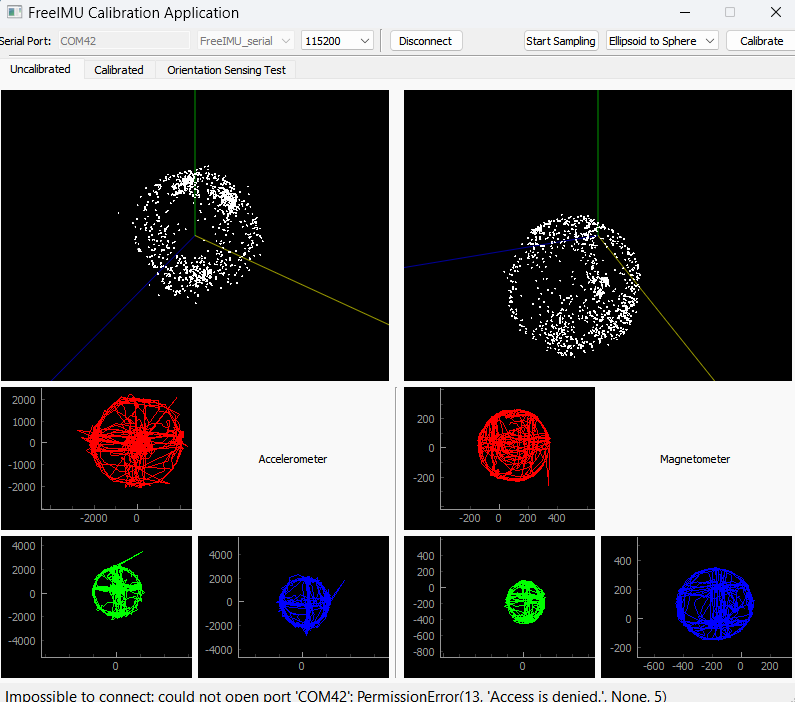
## Frames used to describe attitudes

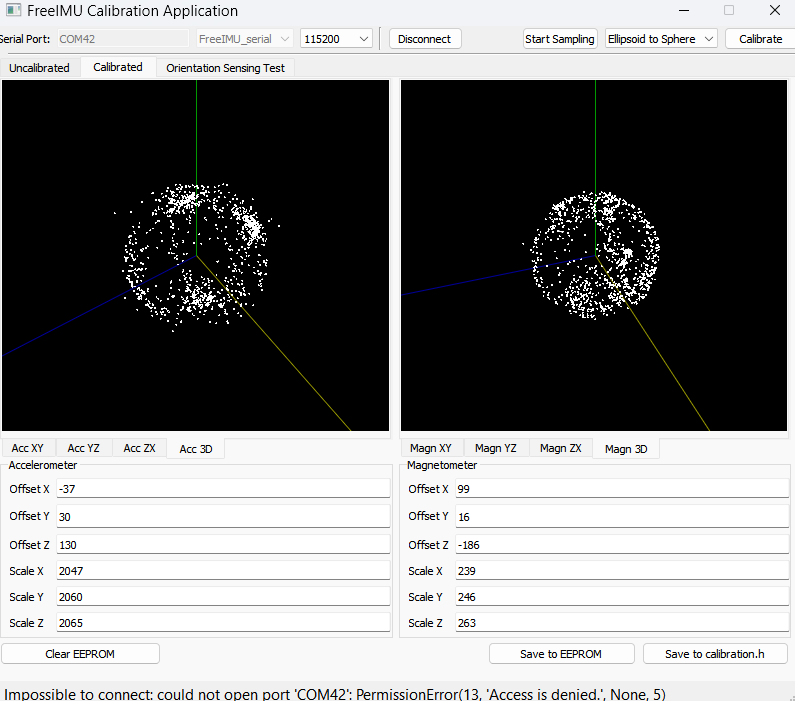
[[edit](https://en.wikipedia.org/w/index.php?title=Axes_conventions&action=edit&section=10)]

Normally the frames used to describe a vehicle's local observations are the same frames used to describe its attitude with respect to the ground tracking stations. i.e. if an ENU frame is used in a tracking station, also ENU frames are used onboard and these frames are also used to refer local observations.

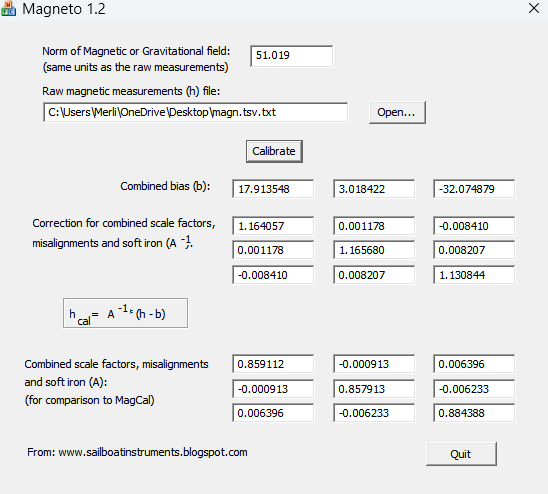
An important case in which this does not apply is aircraft. Aircraft observations are performed downwards and therefore normally NED axes convention applies. Nevertheless, when attitudes with respect to ground stations are given, a relationship between the local earth-bound frame and the onboard ENU frame is used.

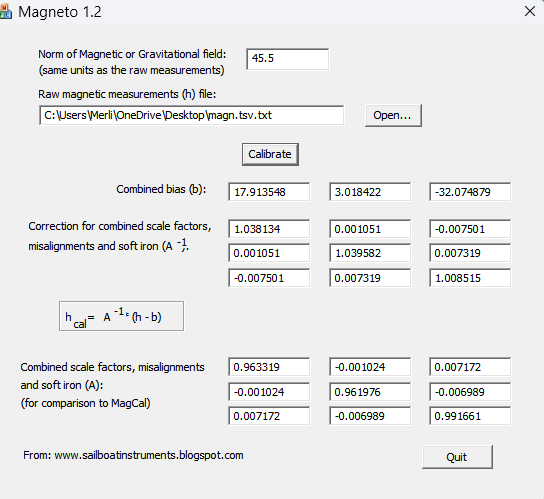
**Appendix 6. Calibration Images.**

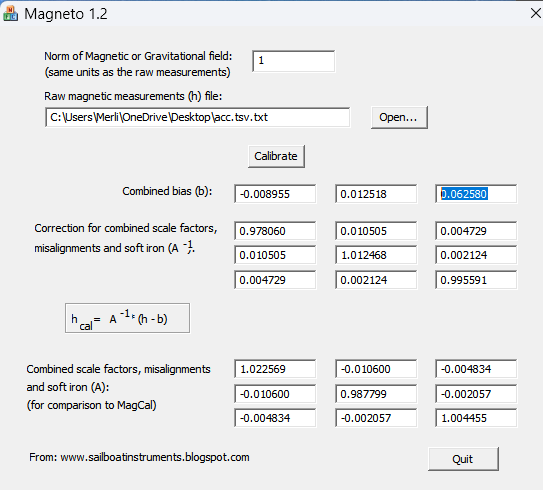
**FreeIMU GUI**

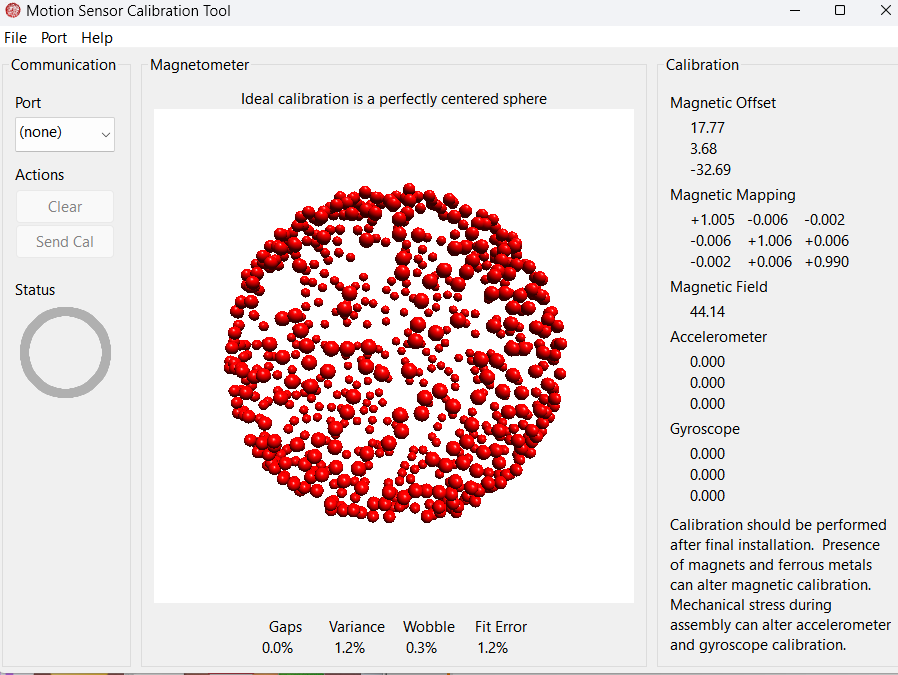


**Magneto 1.2**







**MotionCAL:**

**Data File:  
**

**==============**

