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:

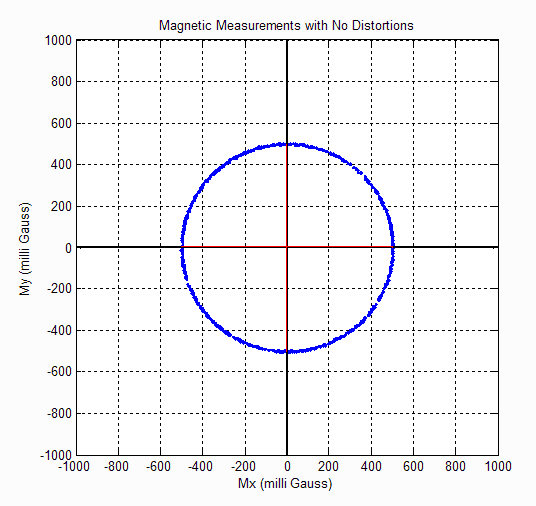
1. **Positioning**: The sensor is placed in six different orientations:
   * +X axis up
   * -X axis up
   * +Y axis up
   * -Y axis up
   * +Z axis up
   * -Z axis up
2. **Data Collection**: In each position, the sensor’s output is recorded. Ideally, these measurements should reflect the true magnetic 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.

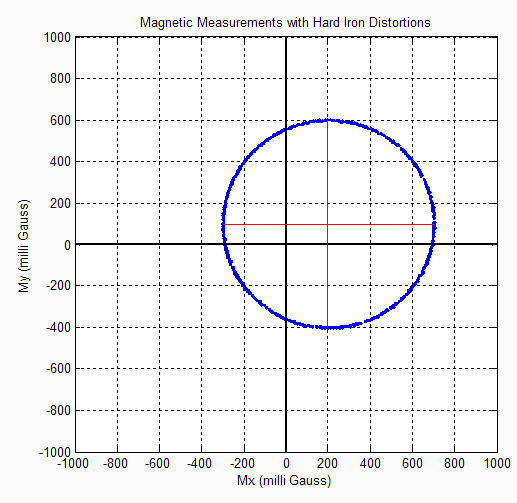
[This method is particularly useful because it doesn’t require any special equipment or reference fields, just the ability to accurately position the sensor in the six specified orientations1](https://community.st.com/t5/mems-sensors/6-point-calibration-of-magnetometer/td-p/274987)[2](https://www.st.com/resource/en/design_tip/dt0053-6point-tumble-sensor-calibrat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

**Elipsoid to sphere fitting**

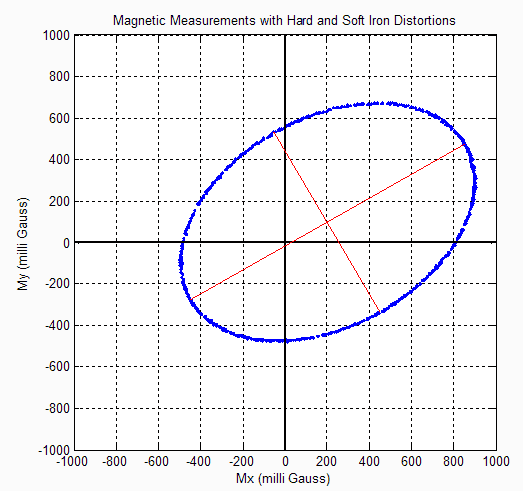
Reference 1 does a nice job of showing the effects of Hard and Soft Iron distortion and the effect of calibration using fitting.

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

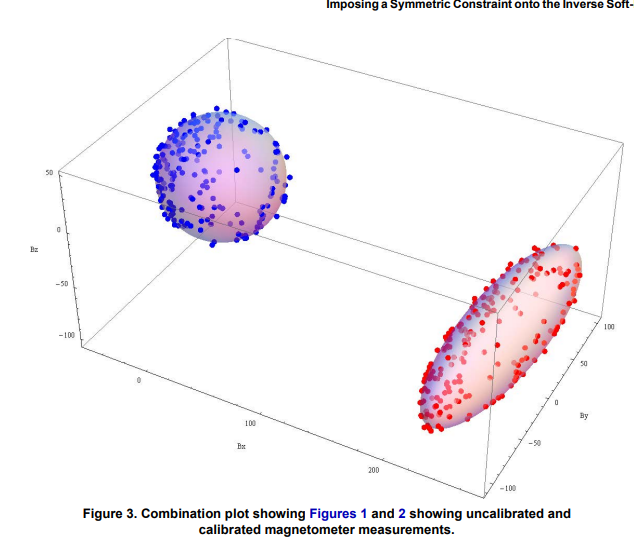


When the magnetometer is exposed to Hard Iron distortion such as a magnet or electronic sources it shifts the sphere in the plot:  


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**

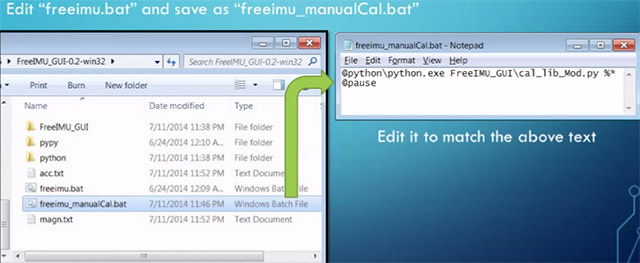
**From** [Adafruit post](https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/calibration) **on the calibration of the LSM303 accelerometer and magnetometer it states that:**

***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.***

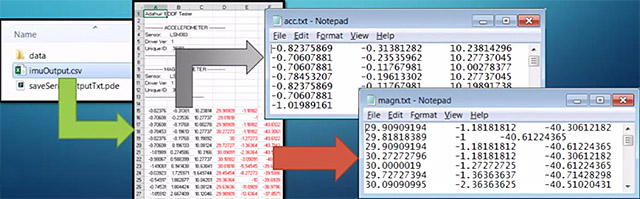
**Unfortunately with Fabio Versano’s untimely passing the original GUI hasn’t been updated from PYQT4. But fortunately some of us has either converted the GUI to PYQT5 or generated a Windows Executable from the original PYT4 so no need to even install Python.**

**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.

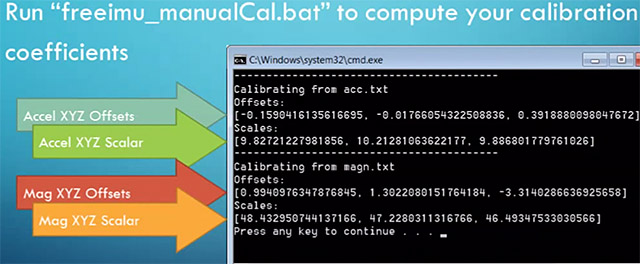
*In the video* [*Adafruit 10 DOF IMU Calibration with FreeIMU GUI Toolset*](https://www.youtube.com/watch?v=G6EtAeidDXk) *it also shows you how to use that shows how a couple of simple config file edits let you run the FreeIMU GUI Toolset in manual mode. Reference 2 includes the instructions on how to 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:

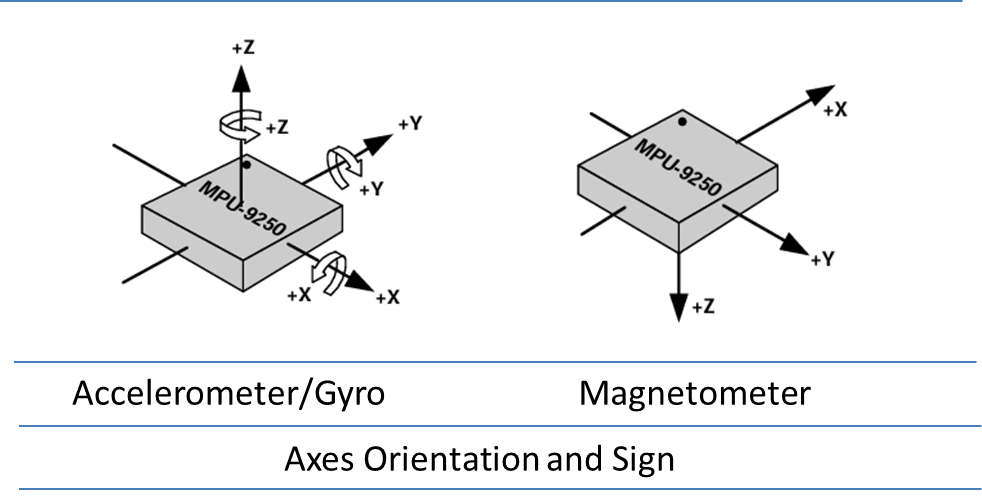
Magx: 0.17922 uT/LSB

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.**

**Magneto 1.2**

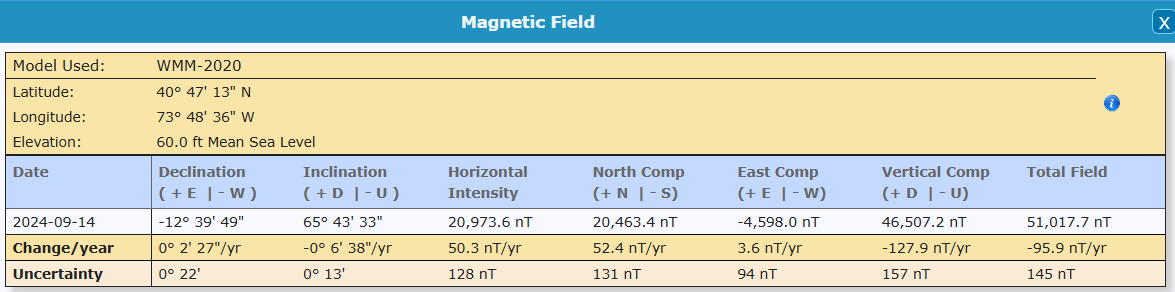
**From Copilot:**

*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:

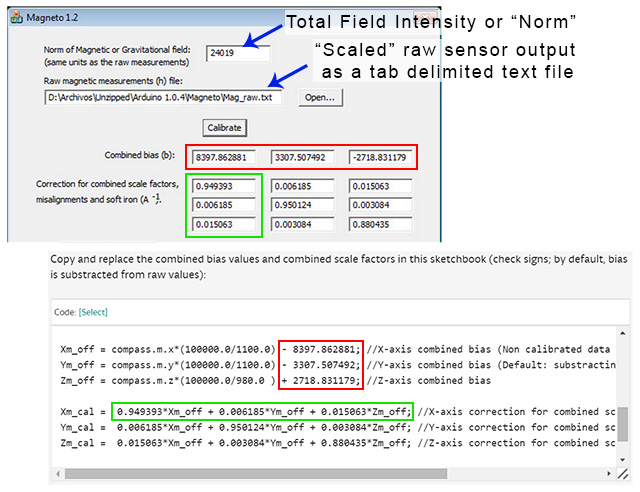
*“Magneto expects to receive raw data in +- format (a value of zero indicating a null field in the current axis), but not necessarily normalized to +-1.0.*

*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.

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)

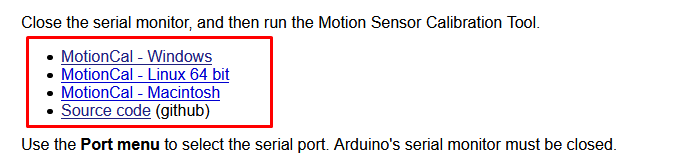
But to quote since I don’t like to hit links:

[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:

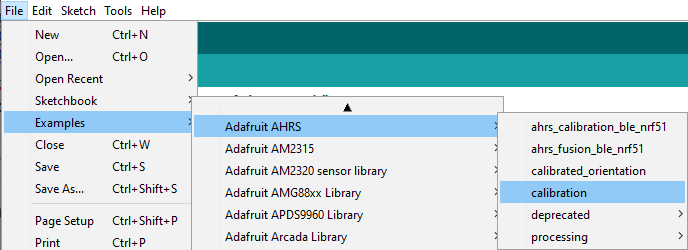
[](https://learn.adafruit.com/assets/87416)

And click the one that matches your computer the best.

# Step 2 - Configure & Upload the AHRS calibration Example

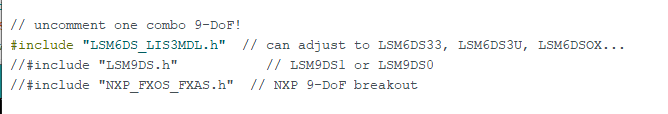
Next we have to tell the microcontroller board to send the magnetometer (and, if there is one, accelerometer and gyroscope) data out over serial in the right format.

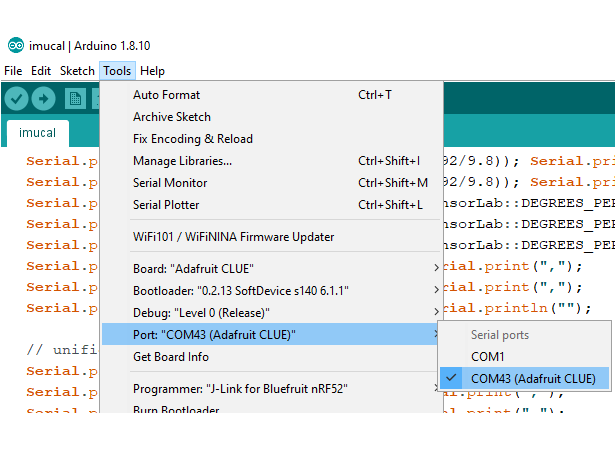
Open up the Adafruit\_AHRS->calibration example

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

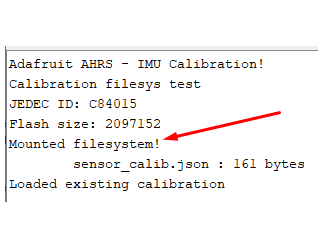
At the top of the sketch you'll see a section where you can #include different sensor sets. Not every sensor-set is defined, but our most popular ones are! (You'll need sensors that are Adafruit\_Sensor compatible.)

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

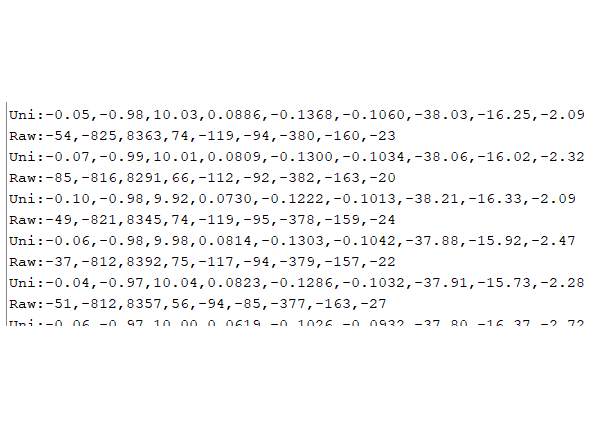
[](https://learn.adafruit.com/assets/88465)

[](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)

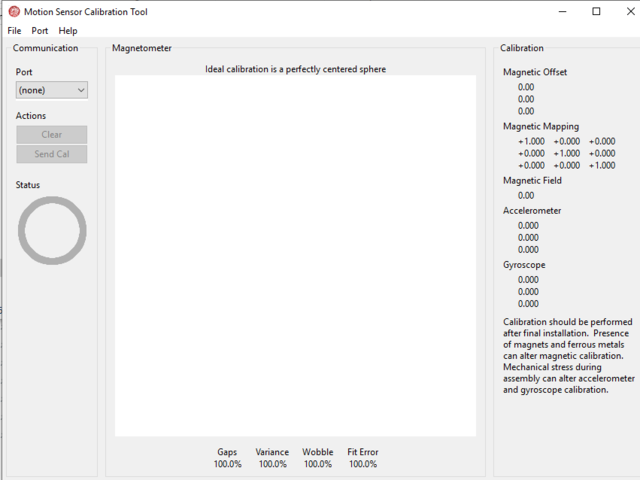
You'll then see a stream of data that looks like:  
Raw:-58,-815,8362,76,-121,-95,-375,-159,-24  
Uni:-0.07,-0.98,10.00,0.0832,-0.1327,-0.1046,-37.50,-15.93,-2.50

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

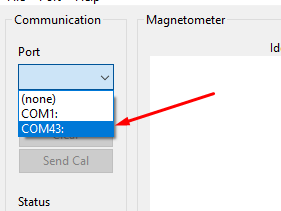
The middle three numbers are gyroscope data - if you don't have an gyroscope, they will be 0

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

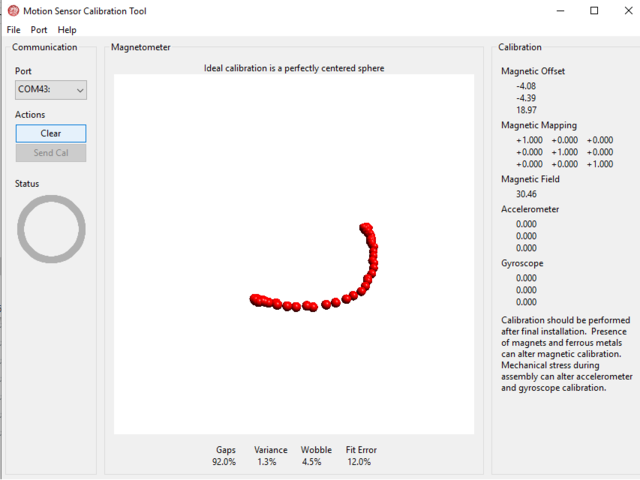
# Step 3 - Run MotionCal

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

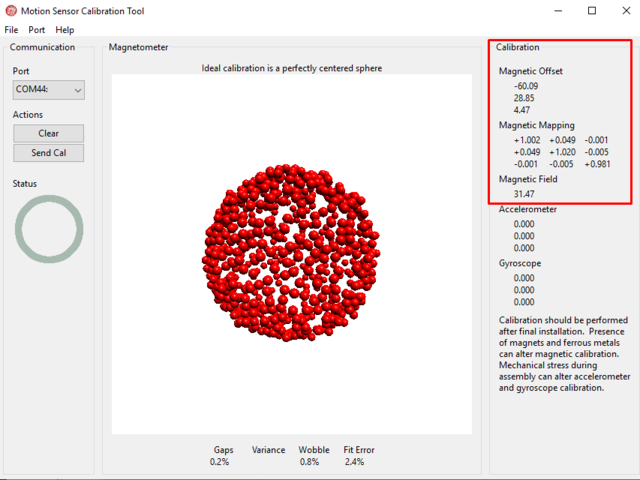
**Close the serial port**, and launch MotionCal

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

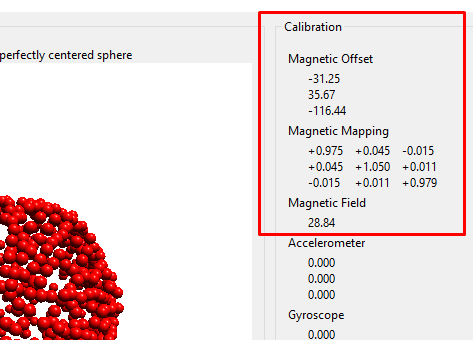
Select the same COM / Serial port you used in Arduino

[](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!

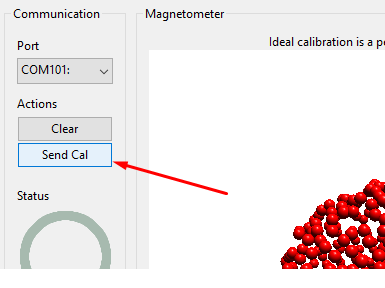
[](https://learn.adafruit.com/assets/88471)

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]

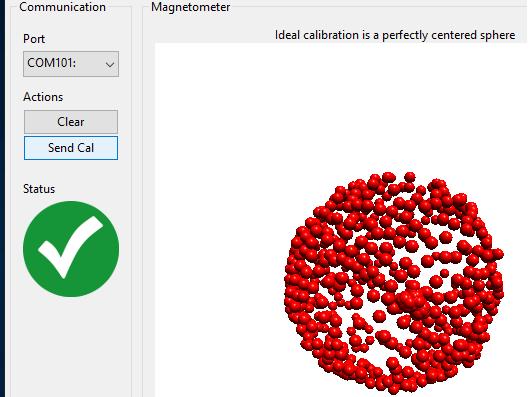
**Take a screenshot of this display, so you can refer to these numbers later!**

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).

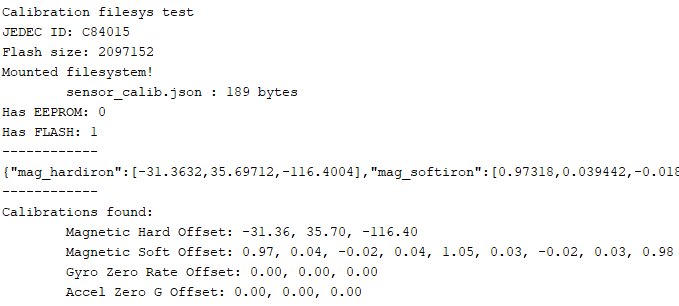
Once you can click the button, try clicking it (we had to try a few times?)

[](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.

Other sources: **How to Calibrate a Magnetometer | Digi-Key Electronics**

**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

****

**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**

**Appendix 1. FreeIMU GUI**



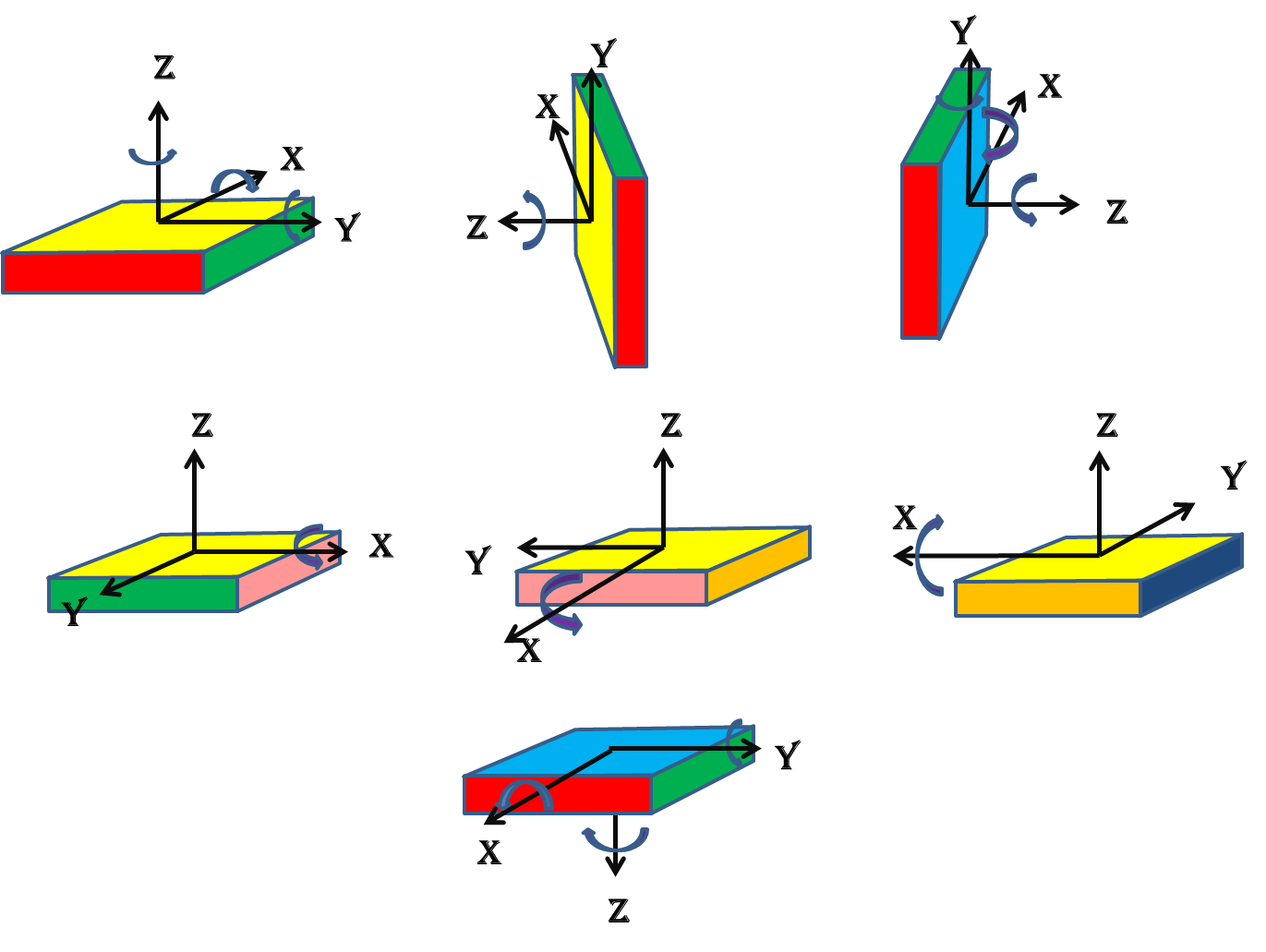
To use:

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.

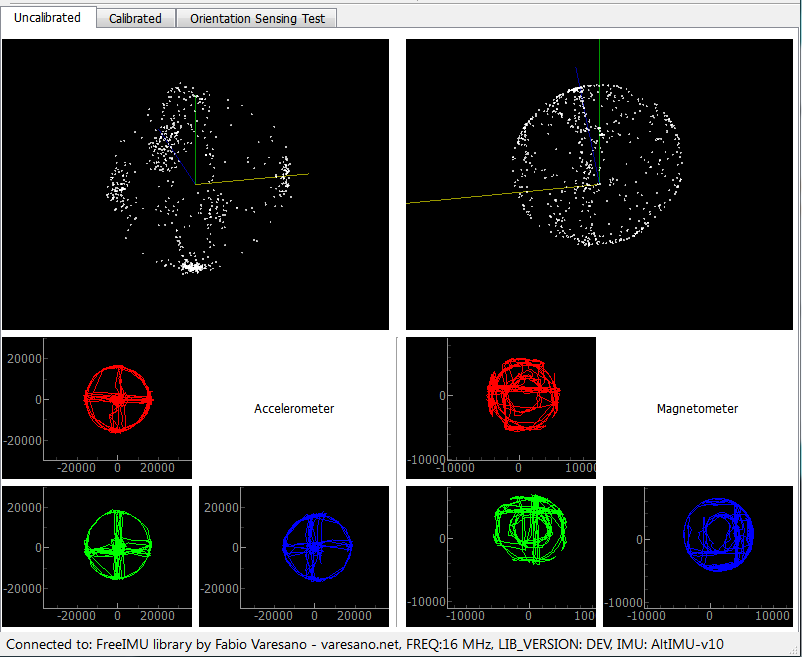
4.I have had the most luck with calibrations that follow the rotation scheme. Just remember to rotate the board slowly and steadily around the axes:



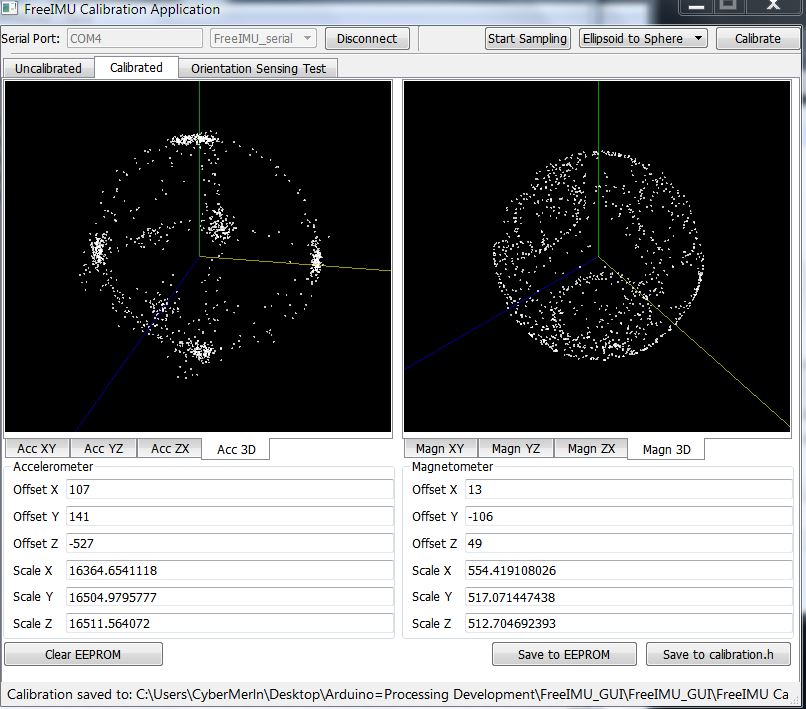
NOTE: Once you start sampling you can put the cursor in anyone of windows and drag the frame around until you center the data points in the window. Also, if you use the scroll wheel on your mouse you can zoom in or out. Helpful when the data isn’t centered around 0, 0 which it won’t be.

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.

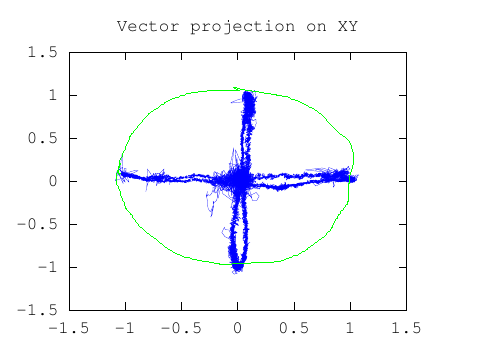
Reload the FreeIMU sketch. That’s it. Re-test with the Free\_cube\_ODO processing sketch.

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

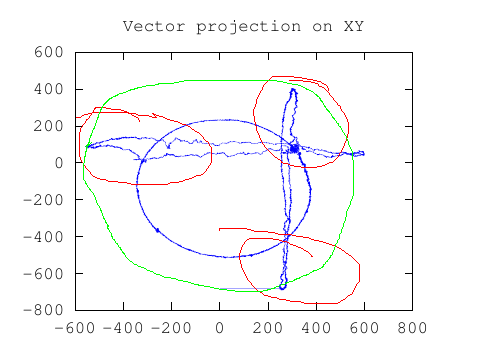
Hi Manor, thanks for sharing Submitted by fabio on Thu, 2012-11-15 09:31. Hi Manor, thanks for sharing your video. However, this is still not a proper calibration approach.

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  // scale factors  float accelScale, gyroScale;  float magScale[3];  //Averages  float ax\_avg, ay\_avg, az\_avg, gx\_avg, gy\_avg, gz\_avg, mx\_avg, my\_avg, mz\_avg;  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 \*/    Wire.begin();    Wire.setClock(400000);    /\* I2C bus,  0x68 address \*/    imu.Config(&Wire, 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("\tMagx: "); Serial.println(magScale[1],5);    Serial.print("\tMagx: "); 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);                i\_count += 1;                writeArr(raw\_values, 9, sizeof(int16\_t)); // writes accelerometer, gyro values & mag if 9150                //writeArr(raw\_values, 6, sizeof(int)); // writes accelerometer, gyro values & mag if 9150                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)){                  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;          }        case 'x': //No calibration          {            uint8\_t count = 100;            uint8\_t i\_count = 0;            //for(uint8\_t i=0; i<count; i++) {            while(i\_count < count) {              if(imu.Read(values1)){                i\_count += 1;                ax\_avg += values1[0]; ay\_avg += values1[1]; az\_avg += values1[2];                mx\_avg += values1[6]; my\_avg += values1[7]; mz\_avg += values1[8];                //sprintf(str1, "%f,%f,%f,%f,%f,%f,%f,%f,%f,", values1[0], values1[1], values1[2], values1[3], values1[4], values1[5], values1[6], values1[7], values1[8]);                //Serial.println(str1);              }            }            float m\_normal = sqrt((mx\_avg/count)\*(mx\_avg/count) + (my\_avg/count)\*(my\_avg/count) + (mz\_avg/count)\*(mz\_avg/count));            Serial.printf("No Cal: %f, %f, %f, %f, %f, %f\n", ax\_avg/count, ay\_avg/count, az\_avg/count, (mx\_avg/count)/m\_normal, (my\_avg/count)/m\_normal, (mz\_avg/count)/m\_normal);            ax\_avg = 0; ay\_avg  = 0;; az\_avg = 0; mx\_avg = 0; my\_avg  = 0; mz\_avg = 0;            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)){**  **//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:

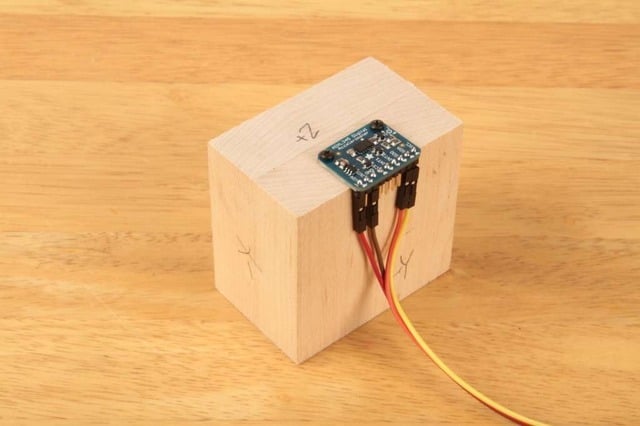
To calibrate the sensor to the gravitational reference, you need to determine the sensor output for each axis when it is precisely aligned with the axis of gravitational pull. Laboratory quality calibration uses precision positioning jigs. The method described here is simple and gives surprisingly good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

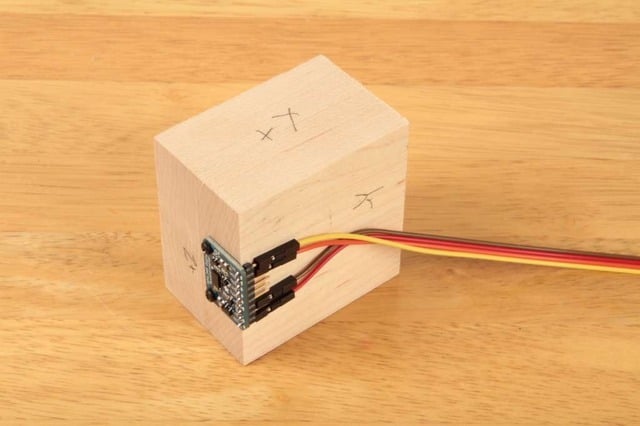
## Position the Block:

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)

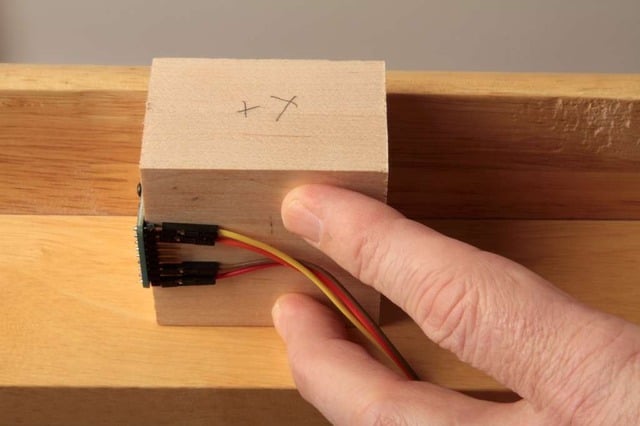
## Reposition the Block:

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.*