CALIBRATING THE MAGNETOMETER

Posted on July 28, 2018

Table of Contents

1. Foreword and Introduction
2. Preparing the Graphics
    - Creating the Wireframe
    - Creating a Perspective View
    - Drawing in Pygame
3. Quaternion Rotation
    - Quaternion basics
    - Testing Quaternion Rotation in Pygame
    - Implementing Rotations with MEMS Gyrometer Data
4. Calibrating the Magnetometer
    - Error Modelling
    - Measurement Model
    - Calibration
5. Extended Kalman Filter Implementation
    - Kalman Filter States
    - Accelerometer Data
    - Magnetometer Data
    - Quaternion EKF Implementation
6. Conclusion

In order to remove the bias from the 3-axis gyroscope, we would require a reference for at least 2 of the axis for a fully defined solution. This can be visualized simply.First, position your fingers similar to that of Fleming's Right-hand Rule. The 3 fingers show the x (thumb), y (index), z (middle). You will realize that when you lock the direction of one finger in place, you could still rotate your hand in some way. However, if you lock the direction of any 2 fingers, you'll find that there is no way for you to rotate your hand anymore. This means that we will need at least 2 references for the axis to lock the 3 directional vectors in place.

In order to do so, we are going to use the accelerometer (gravity vector always points to the ground) and the magnetometer (Earth's magnetic field always points to the Magnetic North). In this post, I will talk about the calibration of the Magnetometer. Calibration of the accelerometer is not done because the raw data is already quite "clean".

What we want to get from the magnetometer is the North directional vector. However, there are all sorts of errors that influence the actual measurement thus there is a need to perform a calibration for the magnetometer before using it. The magnetometer module that I was using was the LSM303D. It came together with the Pololu MinIMU-9 v3 which is now discontinued but the model is not really that important because the calibration method mentioned here should work for all magnetometers.

I got my study materials from a few journal papers, but this website was the most helpful because it actually explains the actually method concisely. In fact, what I am going to explain here will be almost similar to what is written in the website so you can just check out either one (I hope mine is easier to understand!).

Error Modelling

In this section, I will list down a few of the common errors found in a magnetometer together with its mathematical model.

Scale Factor

Alright, this is not really an error due to noise but we will require some form of scaling for the input in order to generate outputs that fall within our desired range. For example, we might want to normalize our 3 direction vectors to have a value between 0 to 1. This can be modeled mathematically with a diagonal matrix as

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Misalignment Error

When setting up the sensors on module, the 3 axis will most probably be somewhat misaligned because we can never position it perfectly. This causes the measurement axis to be skewed such that it is not exactly orthogonal. This can be modeled via

$$N = \begin{bmatrix} n_{x,x} & n_{y,x} & n_{z,x} \\ n_{x,y} & n_{y,y} & n_{z,y} \\ n_{x,z} & n_{y,z} & n_{z,z} \end{bmatrix}$$

where each column represents the orientation of the respective axis with respect to the orthogonal axis.

Categories

Select Category

Archives

September 2019 (3)
February 2019 (1)
January 2019 (1)
September 2018 (1)
July 2018 (4)
March 2018 (3)
February 2018 (5)
November 2017 (6)
July 2017 (2)
April 2017 (6)
March 2017 (4)
February 2017 (2)
January 2017 (9)
December 2016 (24)
November 2016 (1)

Bias Error

The bias error is the error that shifts the measurement values away from the actual value by a fixed amount. If the mean of the actual measurement is 100, then having a bias of 10 would mean that the mean of the measurement reading will be 110. This can be modeled by a simple offset vector as shown below.

$$b_m = \begin{bmatrix} b_{m,x} \\ b_{m,y} \\ b_{m,z} \end{bmatrix}$$

Hard Iron Error

Hard iron errors appear due to the presence of permanent magnets and remanence of magnetized iron. The effect of hard iron errors is the same as having a bias (think of it as the permanent magnet pulling the North direction vector towards it) so the error model is similar as well.

$$b_{hi} = \begin{bmatrix} b_{hi,x} \\ b_{hi,y} \\ b_{hi,z} \end{bmatrix}$$

Soft Iron Error

Soft iron errors appear due to the presence of material that influences or distorts a magnetic field, but does not necessarily generate a magnetic field itself. The presence of iron and nickel for example, will generate a distortion in the measured magnetic field. While hard-iron distortion is constant regardless of orientation, the distortion produced by soft-iron materials is dependent upon the orientation of the material relative to the sensor and the magnetic field. Thus, in order to model this error, a more complicated matrix is required such as shown below.

$$A_{si} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

where each column represents the orientation and scale of the respective axis with respect to the orthogonal axis.

Measurement Model

Using the mathematical model of the above errors, we can then summarize the measurement model as such:

$$h_m = SN(A_{si}h + b_{hi}) + b_m$$

where

h is the actual magnetic field
h_m is the measurement reading from the magnetometer with errors

We can then combine some of the errors together to get a more simplified form:

$$h_m = Ah + b$$

where

$A = SNA_{si}$

$b = SNb_{hi} + b_m$

Calibration

With the above measurement model, we can now manipulate the equation to make h the subject instead.

$$h = A^{-1}(h_m - b) \quad \text{----------} \quad (1)$$

where

h is the actual magnetic field
h_m is the measurement reading from the magnetometer with errors

Therefore, if we know the values of the A^{-1} and b matrix, we will be able to solve for the actual magnetic field. As such, this section will focus on how to estimate the values of the A^{-1} and b matrix using measurement data.

Firstly, there is actually another set of criteria that we have. That is, we want to set the magnitude of the actual magnetic field to be 1. In mathematical equation, that is

$$h^T h = 1 \quad \text{----------} \quad (2)$$

$$\begin{bmatrix} h_x & h_y & h_z \end{bmatrix} \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix} = 1$$

If we substitute equation (1) into equation (2), we will get the following equation.

$$(h_m - b)^T (A^{-1})^T A^{-1} (h_m - b) = 1$$

Let $Q = (A^{-1})^T A^{-1}$. Therefore, Q must be a symmetric matrix since it is the product of a matrix multiplied by its transpose.

$$(h_m - b)^T Q(h_m - b) = 1$$

Expanding the above equation and simplifying,

$$h_m{}^T Q h_m - h_m{}^T Q b - b^T Q h_m + b^T Q b - 1 = 0 \quad \text{----------} \quad (3)$$

since Q is symmetric, $b^T Q h = h^T Q b$. Therefore, we can futher simplify equation (3) to

$$h_m{}^T Q h_m - 2 h_m{}^T Q b + b^T Q b - 1 = 0 \quad \text{----------} \quad (4)$$

Equation (4) can now be rewritten in quadric form (a quadric is a generalization of conic sections (ellipses, parabolas, and hyperbolas))

$$h_m{}^T Q h_m + h_m{}^T n + d = 0 \quad \text{----------} \quad (5)$$

where

$Q = (A^{-1})^T A^{-1}$

$n = -2Qb$

$d = b^T Q b - 1$

There are many different types of quadric surfaces and they can all be verified mathematically. However, we are not going to do that here because there is a more obvious answer. h is supposed to be a unit direction vector so it would only be natural that the set of all directions would form a unit circle. In other words, Equation (5) must be that of an ellipse (a circle is a special case of an ellipse). On a side note, you can find a list of all quadric surfaces and their definition from this site.

Anyway, what we need to do now is to collect some data so that we can use it for our calibration. In order to collect the data, we need the Arduino to send it to the computer, and Python on the computer to save the data (you can always use any other programs to achieve this). The Arduino program is exactly the same as the one in the previous section but I'll provide it below just in case. The python program simply collects the data for a period of time and saves the file in a csv format (Please don't mind the class name because I was too lazy to change it from my previous project). I will also provide the data that I have collected so that you can replicate the graph that I will show later.

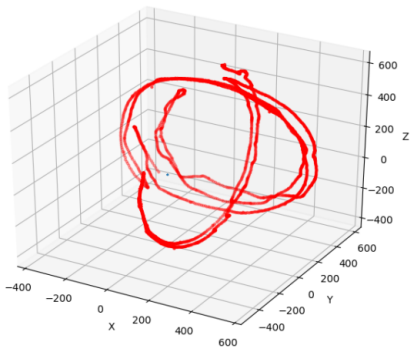
- Arduino Code
- Python Data Collection Code
- Magnetometer Data

Alright, go ahead and collect your data if you happen to use the same sensor as me. Otherwise, you can still use part of the code for the communication between Arduino and Python.

Right now, if we were to simply plot our raw data, the data points will most probably appear on the surface of an ellipse which has a center that has an offset from the origin. However, after our calibration, if we do a scatter plot of our data taken in all directions, we would expect that the points will now lie close to the surface of a unit circle. That is the main objective of this calibration process. Before I begin, here is the calibration code. The algorithm is based on this journal paper (Least squares ellipsoid fitting) and this journal paper (scaling the solution).

- Python Magnetometer Calibration Code

Let's take a look at the raw data that I have gotten from the sensor. I must say that I did not rotate the magnetometer in all directions while collecting the data thus there are blank spaces in the graph as shown below. You would possible get better results if you were to capture the data more thoroughly.



UNCALIBRATED MAGNETOMETER DATA

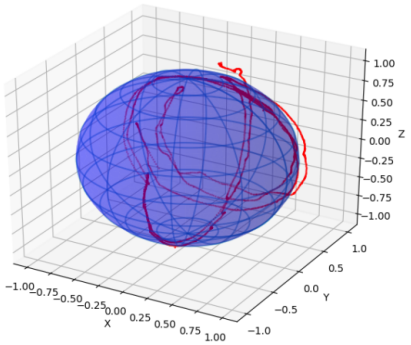
If you take a careful look at the above graph, you'll see a small blue dot at (0, 0, 0). That is actually a unit circle centered at the origin so we want our data to be scaled down such that it lies on that blue circle. You should try to run the program on your computer because then you'll be able to rotate the 3D graph to get a more complete view. Anyway, I am not going to go through the least squares fitting algorithm from the first paper (because I don't really understand the mathematics behind it as well..) but after applying the fitting algorithm, we will be able to solve for the values of Q, n and d from equation (5).

Next, we will have to determine the values of A^{-1} and b so that we can apply equation (1) to perform our calibration. This is done via the following equations based on the second paper.

$$b = -Q^{-1}n$$

$$A^{-1} = \frac{1}{\sqrt{n^T M^{-1} n - d}} M^{1/2}$$

Now that we have the required variables to use equation (1), we are finally able to perform the calibration method. What we have to do now is simply apply equation (1) to all measurement points! For my data set, I was able to get the following graph.



CALIBRATED MAGNETOMETER DATA

And we are done! The magnetometer is now ready for use! On a side note, I do feel that there is still room for improvement in the above calibration process. For my magnetometer, the direction vector is slightly skewed to a particular direction, meaning that an actual 90 degrees show up as 80 degrees and a -90 degrees show up as -80 degrees. I suppose there are ways to correct these but I have not yet found the time to look things up. For now, let us use the above method and proceed to the next section!

Go to Next Section

PREVIOUS POST
Quaternion Rotation

NEXT POST
Extended Kalman Filter Implementation

POSTED IN ATTITUDE DETERMINATION WITH QUATERNION USING EXTENDED KALMAN FILTER

18 comments on "*Calibrating the Magnetometer*"

Guillaume
May 13, 2019 at 4:47 pm

Hi,

I used your code for the calibration and when i use it, i have some times two problems.
First, d > 0 so np.dot(n.T, np.dot(Qinv, n)) – d < 0 so the sqrt doesn't work
Then , my matrix linalg.sqrtm(Q)) doesn't have a real part so my A_1 matrix is nul if i don't have any problems.

Did you had this problem ?

rikisenia.L
May 13, 2019 at 9:06 pm

Hi Guillaume,

Sorry, I've never encountered that problem before.
Have you tried plotting your raw data to check its distribution?
The points should fit on an ellipse.

Guillaume

May 13, 2019 at 9:50 pm

Yes i did and the fitting is working.
I think that's a sign problem because I tried on a few mesures and i noticed that Q is not a positive definite matrix contrary to -Q .
Moreover, sometimes i don't have any problem to use the code and I compared a few matrix Q. They are rougthly the same in absolute value. I saw the same think on d.

Do you have any idea where that come from ?

rikisenia.L
May 14, 2019 at 9:01 pm

Hi Guillaume,

Sorry, I don't quite get your question.
Where did what came from?

Guillaume
May 14, 2019 at 9:55 pm

My matrixs (Q,n,d) seems to be the opposate of the matrixs that i was supposed to have and I don't know why sometimes I have the right matrixs and other times I don't.
As an example :
On my first try i had this :

[Q,n,d]
Out[1213]:
[array([[-0.57940453, -0.03100187, -0.031071811],
[-0.03100187, -0.59777722, 0.00777032],
[-0.03107181, 0.00777032, -0.55223712]]),
array([[0.03381524],
[0.00080195],
[-0.00064805]]),
0.09853586907367784]

Q's eigenvalues are negatifs so Q is not a positive definite matrix so

linalg.sqrtm(Q)
Out[1214]:
array([[0.+0.76063311j, 0.+0.02028939j, 0.+0.02073993j],
[0.+0.02028939j, 0.+0.77287538j, 0.-0.00540421j],
[0.+0.02073993j, 0.-0.00540421j, 0.+0.74281745j]])

So Ainv = array([[0, 0, 0],
[0, 0,0],
[0, 0, 0]])

In my second test i had :

[Q,n,d]
Out[1217]:
[array([[0.58448706, 0.00300477, -0.01577515],
[0.00300477, 0.57566363, 0.01023874],
[-0.01577515, 0.01023874, 0.57150988]]),
array([[-0.02098693],
[0.02258697],
[0.00746044]]),
-0.08809713491939605]

So Q is a positive definite matrix so there is no problem and
Ainv =
array([[1.22849926, 0.00324453, -0.01668946],
[0.00324453, 1.21925511, 0.01088614],
[-0.01668946, 0.01088614, 1.21473764]])

I wonder if you knew where did that sign difference came from ?

rikisenia.L
May 14, 2019 at 10:29 pm

Hi Guillaume,

Are the data for the 2 tries taken in a similar fashion?
I mean rotated sufficiently so that there are data points all around the ellipse?

Guillaume
May 15, 2019 at 8:31 pm

Hi,

I turned my magnetometers randomly, but yes, when I plot my data I see that they describe an ellipsoid.

rikisenia.L
May 16, 2019 at 10:57 pm

Hi Guillaume,

The fitting algorithm is based on the following paper:
https://ieeexplore.ieee.org/document/1290055/authors#authors
If you can access it, I would highly recommend you to read through it.

Based on the journal paper, it seems that most of the time, the matrix (in equation 15) will be positive definite.
However, in some cases, it might not be, even when the data describes an ellipsoid.
In such cases, the author has provided an iterative algorithm which would help to solve the least squares fitting.
However, the code that I have provided only solves for the common case whereby the matrix is positive definite.

My guess is that your first data set happens to be the case whereby the matrix is not positive definite and it will require an iterative algorithm to search for a solution.
You might encounter such problem if your data describes a long/thin ellipsoid.
In such cases, you might want to scale the input first before running it in the code that I have provided or you can also consider implementing the iterative algorithm to make the program more robust.

vicko.p
June 9, 2020 at 6:56 pm

Thank you for your explanation, Guillaume. I would like to ask about how to scale your magnetometer data input. I also have a problem about matrix q,n,d which is not definite positive matrix. I would like to know what is your type data of raw magnetometer and what scaling do you use for magnetometer [in Arduino code], which has affected my plotting calibration result to be Ainv equal to NaN. For your information, I am using type data: int_16t and scaling for magnetometer [after checking the IMU specification]: 1370.

Thank you

Nxt
May 16, 2019 at 12:22 am

Hi,

first of all, great job!

Can you explain to me big difference between matricies mag_Ainv, mag_b hardcoded in Kalman_EKF.py and Ainv, b calculated from your data magnetometer.csv in Elipse_fitting?

Ainv:
[0.000165139 -8.38231e-06 -8.7533e-08
-8.38231e-06 0.000153355 1.43527e-06
-8.7533e-08 1.43527e-06 0.000159855]

b:
[1006.42
463.616
1320.91]

Firstly i though that You made second, better calibration, but differences between values are too big.

Thanks for help 🙂

rikisenia.L
May 16, 2019 at 11:13 pm

Hi Nxt,

When I run the code, I get exactly the same values as the one hard coded in Kalman_EKF.py
Here are my values:

A_inv:
[[2.06423128e-03 -1.04778851e-04 -1.09416190e-06]
[-1.04778851e-04 1.91693168e-03 1.79409312e-05]
[-1.09416190e-06 1.79409312e-05 1.99819154e-03]]

b
[[80.51340236]
[37.08931099]
[105.6731885]]

Total Error: 37.501352

I even downloaded the files from my blog and tried them.
Are you sure you are using my magnetometer file and code as it is?

Nxt
May 18, 2019 at 11:01 pm

Thanks for reply,

You're right, I've downloaded Python code and magnetometer file again and now have same results 🙂

Greetings!

Pierce
June 20, 2019 at 7:21 am

Hi, thanks for writing this and supplying the code.

You may have mentioned this, but I didn't see it when I read back through the article: Why do you scale the data to the unit sphere / why did you want your data to range from 0-1? By normalizing, don't you lose information about the true magnetic field data (not the raw data). I thought that the radius of the sphere was supposed to represent the magnitude of the magnetic field, so by normalizing it we are losing what that value actually is?

Thanks

rikisenia.L
June 20, 2019 at 10:09 pm

Hi Pierce,

You are right in that we lose some information when we normalize the magnetic field data.
In this application, the aim was to obtain the direction vector of Earth's magnetic field hence the data was scaled to a unit sphere.
This actually means that I am assuming that the magnitude of the field is the same everywhere I go as well.

Seyed
March 2, 2020 at 11:59 am

I think there is a mistake in your code! loopRate = 50 and then you have 1/loopRate
This is always equal to zero 1/loopRate = 0
I think it has to be loopRate = 50.0

rikisenia.L
March 17, 2020 at 4:16 pm

Hi Seyed,

You are right about that if you are using python 2.x
I am using python 3.x which does floating point calculation even when integers are used.
You can take a look here for more information.

https://en.wikibooks.org/wiki/Python_Programming/Operators

Veeresh Dammur
March 29, 2021 at 2:55 am

I am using LSM9DS1 IMU to get 9 axis sensor data and I am not able to save the correct sensor read values on the python side

Library: https://github.com/FemmeVerbeek/Arduino_LSM9DS1

Accelerometer unit: g
Gyroscope unit : deg/sec
Magnetometer unit: gauss

I am using the above mentioned LSM9DS1 library to read sensor values from registers. Below are the Arduino code and I am using 4 bytes (float data type) to send it to PC.

```
#include

unsigned long timer = 0;
long loopTime = 10000; // microseconds

float a_x, a_y, a_z;
float g_x, g_y, g_z;
float m_x, m_y, m_z;

void setup()
{ Serial.begin(115200);
while (!Serial);

if (!IMU.begin())
{ Serial.println("Failed to initialize IMU!");
while (1);
}

IMU.accelUnit = GRAVITY; // accelerometer unit: g
IMU.gyroUnit = DEGREEPERSECOND; // gyroscope unit: deg/sec
IMU.magnetUnit = GAUSS; // magnatometer unit : gauss
}

void loop() {
float x, y, z;
float acc_x, acc_y, acc_z;
float gyr_x, gyr_y, gyr_z;
float mag_x, mag_y, mag_z;

timeSync(loopTime);
if (IMU.accelAvailable())
{ IMU.readAccel(acc_x, acc_y, acc_z);

}

if (IMU.gyroAvailable()){
IMU.readGyro(gyr_x, gyr_y, gyr_z);
}

if (IMU.magnetAvailable()){
IMU.readMagnet(mag_x, mag_y, mag_z);
}

a_x = acc_x;
a_y = acc_y;
a_z = acc_z;

g_x = gyr_x;
g_y = gyr_y;
g_z = gyr_z;

m_x = mag_x;
m_y = mag_y;
m  z = mag  z;
```

```
float* data1 = &g_x;
float* data2 = &g_y;
float* data3 = &g_z;
float* data4 = &a_x;
float* data5 = &a_y;
float* data6 = &a_z;
float* data7 = &m_x;
float* data8 = &m_y;
float* data9 = &m_z;

byte* byteData1 = (byte*)(data1);
byte* byteData2 = (byte*)(data2);
byte* byteData3 = (byte*)(data3);
byte* byteData4 = (byte*)(data4);
byte* byteData5 = (byte*)(data5);
byte* byteData6 = (byte*)(data6);
byte* byteData7 = (byte*)(data7);
byte* byteData8 = (byte*)(data8);
byte* byteData9 = (byte*)(data9);

byte buf[36] = {byteData1[0], byteData1[1], byteData1[2], byteData1[3],
byteData2[0], byteData2[1], byteData2[2], byteData2[3],
byteData3[0], byteData3[1], byteData3[2], byteData3[3],
byteData4[0], byteData4[1], byteData4[2], byteData4[3],
byteData5[0], byteData5[1], byteData5[2], byteData5[3],
byteData6[0], byteData6[1], byteData6[2], byteData6[3],
byteData7[0], byteData7[1], byteData7[2], byteData7[3],
byteData8[0], byteData8[1], byteData8[2], byteData8[3],
byteData9[0], byteData9[1], byteData9[2], byteData9[3]};

Serial.write(buf, 36);

// ————————————————————————————————

// String result = String(gyr_x)+ ‘,’ + String(gyr_y) + ‘,’+ String(gyr_z) + ‘,’+\
// String(acc_x)+ ‘,’ + String(acc_y) + ‘,’+ String(acc_z) + ‘,’+\
// String(mag_x)+ ‘,’ + String(mag_y) + ‘,’+ String(mag_z);
// Serial.println(result);

}

void timeSync(unsigned long deltaT)
{
unsigned long currTime = micros();
long timeToDelay = deltaT – (currTime – timer);
if (timeToDelay > 5000)
{
delay(timeToDelay / 1000);
delayMicroseconds(timeToDelay % 1000);
}
else if (timeToDelay > 0)
{
delayMicroseconds(timeToDelay);
}
else
{
// timeToDelay is negative so we start immediately
}
timer = currTime + timeToDelay;
}s
```

Sample serial print on the Arduino side

```
//gyr_x, gyr_y, gyr_z,acc_x, acc_y_acc_z, mag_x, mag_y, mag_z
0.79,0.98,-0.37,-0.01,-0.00,1.00,-0.03,0.38,-0.44
0.67,0.79,-0.43,-0.01,-0.01,1.00,-0.03,0.37,-0.44
0.67,0.79,-0.43,-0.01,-0.01,1.00,-0.03,0.37,-0.44
0.67,0.79,-0.37,-0.01,-0.01,1.00,-0.03,0.37,-0.44
0.73,0.79,-0.49,-0.00,-0.01,1.00,-0.03,0.37,-0.44
0.73,0.79,-0.49,-0.00,-0.01,1.00,-0.03,0.37,-0.44
0.73,0.73,-0.43,-0.00,-0.00,1.00,-0.04,0.38,-0.44
```

Sample data which is store in the CSV file on python side :

```
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0
```

Question :

1. Though I am using 4 bytes to send each sensor data on the Arduino side, why am I getting integer data type on the python side?
2. Are the units of the sensor data correct?

Francesco Setragno
July 5, 2021 at 4:45 pm

How can I perform the same calibration technique in 2D (i.e. planar point)? I tried this code but sometimes it fails, returning NaN.

Comments are closed.