a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

ENGR 499 - Engineering Capstone Design Project

Final Design Report

Group #52

Spatial Tracking Algorithm for Wearable Sensors

Group Member

Zonghao Li #50024132

Vipul Vishnoi #49856123

Bernie Wu #30945125

Project Advisor

Dr. Thomas Johnson, PhD., PEng.

Course Coordinators

Dr. Dwayne Tannant, PhD., P.Eng.

Dr. Claire Yan, PhD., P.Eng.

Dr. Sabine Weyand, PhD

April 10th, 2017

Table of Contents

1 Project description 1

2 Needs and constraint identification 2

2.1 Client needs 2

2.1.1 Data accuracy 2

2.1.2 Low cost 2

2.2 Constraints 2

2.2.1 Software and hardware constraints 2

2.2.2 Geographical constraints 3

3 Project specifications 4

3.1 Realizable use cases 4

3.2 Effective calibration method 4

4 Design process 5

4.1 Alternatives 5

4.2 Methodology – Ellipsoid Mapping Algorithm 6

4.2.1 Modelling magnetometer readings 6

4.2.2 Calculation of hard iron and soft iron error 10

5 Final design details 13

5.1 Prototypes 13

5.1.1 Static prototype 13

5.1.2 Wearable prototype 13

5.2 Software tools 14

5.2.1 3D visualization and post-processing 14

5.2.2 Stickman animation 15

6 Experiments 17

6.1 Static measurements 17

6.1.1 Indoor – EME Foyer 17

6.1.2 Outdoor – Soccer field 18
6.1.3 Metallic interference – Iron rod 18
6.2 Dynamic measurements 19
6.2.1 Arm tracking 20
6.2.2 Treadmill 20
6.2.3 Squats 22
7 Results 23
7.1 Static measurement results 23
7.1.1 Indoor measurement results 23
7.1.2 Outdoor measurement results 25
7.1.3 Iron rod measurement results 25
7.2 Dynamic measurement results 26
7.2.1 Arm tracking results 26
7.2.2 Squat results 27
7.2.3 Treadmill results 28
8 Design evaluation 29
8.1 Assessment of performance 29
8.2 Recommendations and future work 29
9 References 30
Appendix A Glossary 31
Appendix B Multi-criteria optimization on selecting use cases 32
Appendix C Calibrated parameters for each experiment 35
Appendix D Local ideal magnetic field and conversion to LSB 39
Appendix E Source code 40
Appendix E.1 3dplot.py 40
Appendix E.2 Static_calibration.py 44
Appendix E.3 Stickman_animation.py 48

List of Figures

Figure 1: Distortion of Earth's magnetic field near an iron rod 1

Figure 2: Static and dynamic errors 5

Figure 3: Coordinate system [8] 7

Figure 4: Gravitational and magnetic field vector [8] 8

Figure 5: Calibrated (blue) and un-calibrated readings (red) [8] 11

Figure 6: Flowchart of EMA 12

Figure 7: Static prototype with two automated DOF 13

Figure 8: Wearable prototype with wireless capability and multiple IMU support 14

Figure 9: Real-time plot of magnetometer readings 15

Figure 10: Treadmill and squat animation 15

Figure 11: North and South compass animation 16

Figure 12: Experimental layout in EME Foyer Level 0 18

Figure 13: Experimental setup for demonstrating metallic interference 19

Figure 14: Arm tracking experiment 20

Figure 15: Treadmill - motion capturing 21

Figure 16: Treadmill - interference test 22

Figure 17: Un-calibrated (left) and calibrated (right) readings scatter plot at Location 1 24

Figure 18: Mapping ellipsoid to sphere at Location 1 24

Figure 19: Errors in data with varying distances of iron rod from the sensor 26

Figure 20: Arm tracking result 27

Figure 21: Squat motion and stickman animation 27

Figure 22: Treadmill motion and stickman animation 28

1 Project description

Tracking human body motion has multiple applications in robotics, fitness, and medical fields. The purpose of this project is to obtain magnetometer (sensor) data and improve its reliability for accurately modeling the human body motion. Spatial tracking is achieved by using a low cost inertial measurement unit (IMU), which is the integration of an accelerometer, a gyroscope, and a magnetometer. Inevitably, the magnetometer readings cannot be accurate due to hard iron and soft iron errors. Hard iron errors are created by ferromagnetic materials that have a fixed spatial relationship with the magnetometer in the IMU, while soft iron errors are created by ferromagnetic materials that have a variable spatial relationship with the IMU. This project focusses on optimizing the spatial tracking algorithms and calibration methods for a magnetometer to compensate these errors.

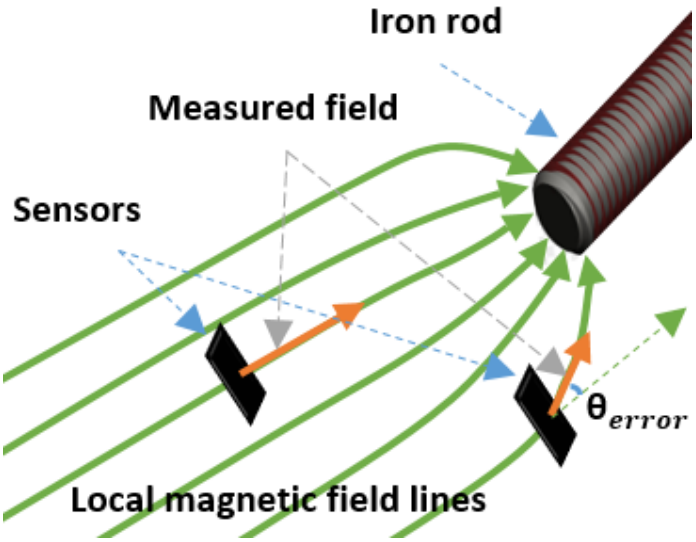


Figure 1: Distortion of Earth's magnetic field near an iron rod

Figure 1 above, shows the distortion of the magnetometer sensor when a ferromagnetic material is brought near its proximity. The sensor that is further away from the iron rod will not be distorted as shown by the measured field (orange vector); however, the sensor that is much closer to the iron rod becomes distorted and a corrupted magnetic field vector is measured. As the iron rod is brought closer to the sensor, the measured magnetic field becomes more distorted causing the sensor to read the wrong data.

2 Needs and constraint identification

2.1 Client needs

The following subsections describe the client needs and expectations.

2.1.1 Data accuracy

Kinetic Reality specializes in human body motion tracking which requires accurate and reliable measurements of position. The need for such accuracy arises from various applications such as athletic performance, medical therapy treatment, and robotics control. The major causes of inaccurate sensor readings are hard and soft iron errors. Examples of hard iron errors are metal on the circuit board while soft iron errors are created by nearby objects such as metal posts, rebar, or metallic components in sports equipment. Our project needs to fully characterize these errors and present the data accuracy for each use case.

2.1.2 Low cost

Nowadays, with the development of sensor technologies, the compromise between the sensors' low cost and high performance is drawing attention. Consumer level sensors such as IMUs in smartphones are sold for low prices, but are not as accurate as the high-end products. On the other hand, the high-end sensors are accurate but very costly. The client need is to exploit low cost hardware and combine this with sophisticated software algorithms to deliver performance that is acceptable to a broad market suitable for applications such as sports, physiotherapy, and other possible areas.

2.2 Constraints

This section outlines the constraints of this project. The major constraints of this project are hardware, software, and geographical constraints. The following subsections will describe the constraints in more detail.

2.2.1 Software and hardware constraints

The magnetometer is a 3-axis electronic compass transducer that uses the Hall Effect to sense the strength and direction of the magnetic field. With a magnetic field strength ranging ±

48000 μT, the magnetometer will output 16-bit data points on the x, y, and z directions. The sensitivity scale factor is 0.6 μT/LSB (Least Significant Bit) with an initial calibration tolerance at zero-field output of ±500 LSB [1] [2].

The project objective is to improve the accuracy of magnetometer by optimizing the spatial tracking algorithm. The software development environment is critical to the implementation and verification of the algorithms. Open source development tools have been selected for the project and the tools consists of a compiler (GNU Compiler Collection), a debugger (GDB), and a linker (OpenOCD) to interface between the development environment on the computer and the target hardware.

2.2.2 Geographical constraints

Magnetic inclination and declination, which varies significantly with the longitude and latitude respectively, are two important offsets that should be compensated before conducting any experiments. Magnetic inclination is the angle formed by the earth's magnetic field and the horizontal reference. Positive magnetic inclination implies the magnetic field is pointing downward. In Kelowna, this angle is about 60-65 degrees and the magnetic field strength in Kelowna is measured to be 55,092.4 nT. Magnetic declination is the angle between the geographic north and magnetic north, as the magnetic poles are not coincidental with the geographic poles. It could either be pointing east or west [3] [4]. To minimize the distortions from soft and hard iron errors and to obtain an even magnetic field magnitude in all orientations, the test field should be large enough with few metallic objects nearby.

3 Project specifications

The use cases and feasible calibration methods will be outlined in this section.

3.1 Realizable use cases

The use cases will ensure a controlled environment to help us evaluate the sensor readings and estimate the sensitivity of the magnetometer to nearby metal and ferrite objects. Due to limitations in space and time, no more than six activities were considered: soccer, ice skating, walking, squat, swimming, and weightlifting. These six activities were put through a multi-criteria decision making (MCDM) method to select the best two activities to perform. Appendix B shows how these decisions were made.

These activities will introduce various levels of complexities into our use cases. For example, a squat test would be a good starting point as it can be easily modeled. Furthermore, activities such as ice-skating, swimming, and soccer will introduce dynamic errors due to the motion and environment. These activities will cover a wide range of body movements, analyze the effects of ferromagnetic interference around the magnetometer, and should also help determine the sensor limitations and sensitivity.

3.2 Effective calibration method

One of the most challenging parts of this project is the ability to accurately calibrate the magnetometer readings in the presence of metal and other ferrite components. Calibration can be divided into two categories called static and dynamic calibration as shown in Figure 2. Static calibration is defined for an object at rest, while dynamic calibration is defined for an object in motion. Static calibration is affected by both hard and soft iron errors, while dynamic calibration is only affected by soft iron errors. Hard iron error accounts for most of the errors in a static calibration (±10 degrees of error after correction); moreover, soft iron error correction can reduce the errors to ±2 degrees. Conversely, since dynamic errors are caused by motion and local surroundings, it can have drastic effects even after static calibration is implemented, resulting in undesirable data (possibly up to 100% error) from the sensor. This occurs due to the effects of ambulatory measurements from the sensor with respect to nearby metal objects and can be very complex to analyze. Calibration errors within 10% is acceptable and deemed accurate for this project.

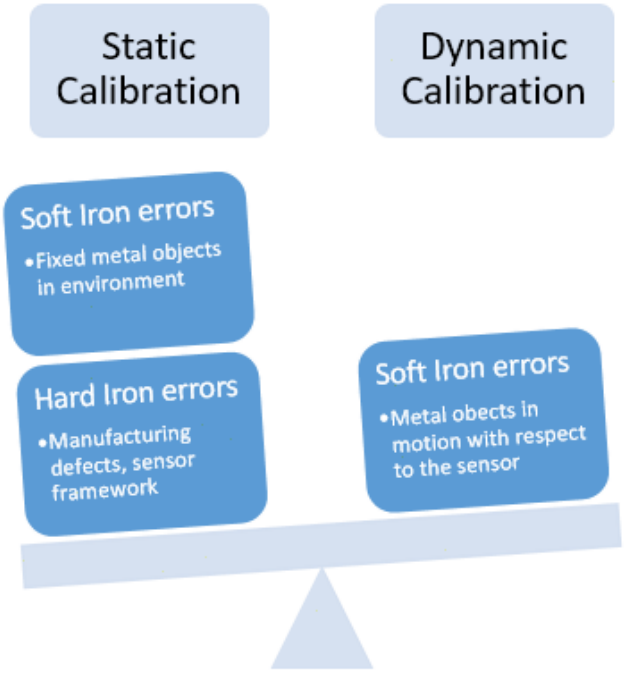


Figure 2: Static and dynamic errors

4 Design process

4.1 Alternatives

The key objective of this project is to mitigate or remove the soft iron and hard iron errors. There are several algorithms for calibrating a magnetometer. Manon Kok and Thomas B. Schon [5], summarized two classes of calibration approaches (see Table 1). One is based on a methodology called “Scalar Checking” which is minimizing the discrepancy between the measured and actual local magnetic field [6]. There are some intuitive and simple calibration tools roughly based on this idea [7]. Another class is the Ellipsoid Mapping Algorithm (EMA) [8] [9] [10].

Compared to the first calibration approach, EMA has the following advantages: firstly, a number of literature resources related to this idea are available; secondly, it is more implementable in microcontrollers; thirdly, it is computationally lighter than the “Scalar Checking” approach as the latter involves “a cost function which is quartic in the magnetometer bias” [7]. Certainly, EMA has its disadvantages, and majorly requires a large dataset in order to provide better accuracy. There are some open-source software tools that

provide robust calibration performance based on this ellipsoid fitting approach. One of them is called "MagCal", which was developed by the Positioning, Location and Navigation Group (PLAN) at the University of Calgary [9].

Table 1: Algorithm selection

Calibration class	Advantages	Disadvantages
Scalar Checking	Could be simplified and easily implemented	Computationally heavy, less resources and materials
Ellipsoid Mapping Algorithm (EMA)	Good accuracy and feasibility, more resources available	Need many datasets to yield robust performance

4.2 Methodology – Ellipsoid Mapping Algorithm

As discussed above, our project uses the Ellipsoid Mapping Algorithm (EMA) as our static calibration approach.

4.2.1 Modelling magnetometer readings

In order to implement EMA in our project, it is necessary to analyze its mathematic explanation and derivation. To begin, the hard iron error can be modeled by the vector V:

$$V = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \tag{1}$$

The soft iron error is originally defined as the magnetic field interference induced by the geomagnetic field onto ferromagnetic components in the surrounding. This will scale the sensor readings up or down in tri-axes. Due to the imperfect chip fabrication process, the orthogonality and the scaling factor of three sensor axes will have defects. Therefore, in order to use one term to govern all these errors, the soft iron error W could be written as the product of the originally defined soft iron error W_{soft}, the gain discrepancies in three axes W_{Gain}, and the orthogonality imperfection of the sensor, $W_{NonOrthog.}$

$$W = W_{soft}\, W_{Gain}\, W_{NonOrthog} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix} \tag{2}$$

Assuming that the induced soft iron error is linearly related to the geomagnetic field measured in the rotated sensor reference frame is a 3 by 3 matrix W_{Soft} [11], and is also valid for W_{Gain} and $W_{NonOrthog}$, W is thereby also a 3 by 3 matrix. It can also be shown that W is not singular $(det(W) \neq 0)$ and therefore a non-trival (or inverse) exists.

Define the rotation matrices of the sensor roll $R_x(\phi)$, pitch $R_y(\theta)$, and yaw $R_z(\psi)$ (Figure 3) as:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \quad (3)$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \quad (4)$$

$$R_z(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5)$$

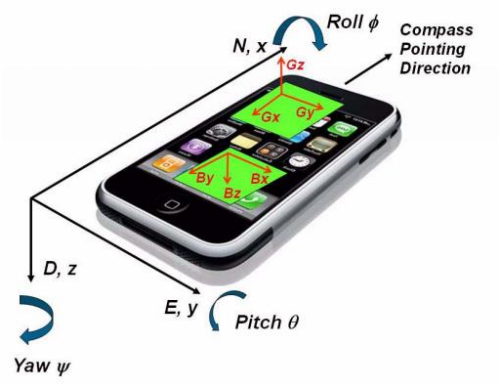


Figure 3: Coordinate system [8]

Denote the well-known Euler angle R as:

$$R = R_x(\phi)R_y(\theta)R_z(\psi) \quad (6)$$

Since R is an orthogonal matrix, it has the following identity:

$$R^T R = I \quad (7)$$

With the absence of the hard iron error and soft iron error, the magnetic field sensor readings can be defined as:

$$B_p = RB_r = RB\begin{bmatrix} cos\delta \\ 0 \\ sin\delta \end{bmatrix} \tag{8}$$

where B_r is the magnetic field at a reference point, B_p is the output magnetic field readings from the magnetometer, and δ is the local inclination angle (Figure 4). Moreover, equation (8) implies that the undistorted magnetometer data should be forming a sphere with radius B. This is the ultimate objective of EMA.

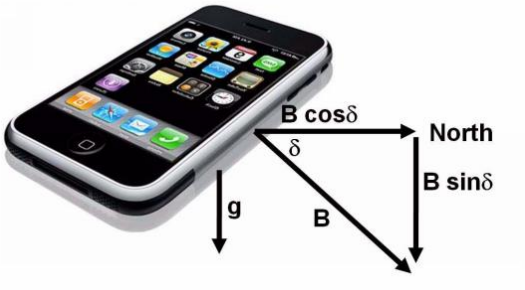


Figure 4: Gravitational and magnetic field vector [8]

With the consideration of both soft iron and hard iron errors, the actual magnetometer readings can be formulated by:

$$B_p = WRB_r + V \tag{9}$$

The major objective of the derivation of EMA is to justify that B_p is an ellipsoid. Generally, for an arbitrary ellipsoid, centered at v, as a function of x, can be expressed by the following equation:

$$(x - v)^T Q(x - v) = constant \tag{10}$$

where Q has to be a symmetric matrix:

$$Q = \begin{bmatrix} A & D & E & G \\ D & B & F & H \\ E & F & C & I \\ G & H & I & J \end{bmatrix} \tag{11}$$

Notice that equation (9), we first move the hard iron bias to the left-hand-side of the equation, and then multiply the inverse of the soft iron error W at both sides to yield:

$$W^{-1}(B_p - V) = RB_r \tag{12}$$

then taking the transpose of both side:

$$(B_p - V)^T (W^{-1})^T = B_r^T R^T \tag{13}$$

If we multiply the LHS and RHS of equations (12) and (13):

$$\left(B_p - V\right)^T (W^{-1})^T W^{-1} \left(B_p - V\right) = B_r^T R^T R B_r \tag{14}$$

by using the matrix associative property and equation (7), the following can be derived:

$$\left(B_p - V\right)^T A \left(B_p - V\right) = B^2 \tag{15}$$

where:

$$B_r^T B_r = B \begin{bmatrix} cos\delta \\ 0 \\ sin\delta \end{bmatrix}^T B \begin{bmatrix} cos\delta \\ 0 \\ sin\delta \end{bmatrix} = B^2 \tag{16}$$

$$A = \left[(W^{-1})^T (W^{-1})\right] \tag{17}$$

A is defined as the scaling matrix evaluated from the soft iron error W.

Comparing (10) and (15), we could see that equation (15) is an expression of a quadric surface. Essentially, it describes an ellipsoid whose loci are the magnetometer readings B_p, centered at V, and rescaled in major and minor axes by A. Also, it can be easily shown that matrix A is symmetric. Therefore, we could conclude that the magnetometer readings can be modelled as an ellipsoid.

Now revisiting equation (12), since in real life the hard iron error V and soft iron error W^{-1} (or W) are usually unknown, in order to precisely map the magnetometer readings to an ellipsoid, calculating these two errors is the a critical step. If we approach this problem reversely, assuming we somehow know the calibrated hard iron error V_{cal} and the soft iron error W_{cal}^{-1} by using EMA, the calibrated magnetometer readings B_{cal} can be expressed by:

$$B_{cal} = W_{cal}^{-1}(B_p - V_{cal}) \tag{18}$$

If EMA functions correctly, then it should imply that:

$$\begin{aligned} W_{cal}^{-1} W &= I \\ V - V_{cal} &= 0 \end{aligned} \tag{19}$$

With condition (19) applying to (18), the calibrated data should be:

$$B_{cal} = R B_r \tag{20}$$

Equation (20) is identical to (8), which is the ideal magnetometer readings with the absence of hard iron and soft iron errors. This reveals that essentially once EMA can compute V and W^{-1} accurately, the calibrated sensor data can a sphere.

4.2.2 Calculation of hard iron and soft iron error

The calculation of V_{cal} and W_{cal}^{-1} can be found in [9] [10] [12] [13]. In short, people usually approach this issue by using least-square fitting problem. From equation (11), in general, in order for a quadric surface to be an ellipsoid, one sufficient but not necessary condition is given by [5]:

$$4J - I^2 > 0 \tag{21}$$

However, using this inequality constraint to fit sensor data to an ellipsoid is computationally expensive [12]. K. Kanatani, proposed a new method called "bias" [14], could further decrease the degree of freedom of the constraint in (21) thereby increasing the efficiency of the optimization. The new constraint is defined as:

$$4J - I^2 = 1 \tag{22}$$

Rewriting (22) in the matrix form:

$$v^T C v = 1 \tag{23}$$

where C is defined as the constraint matrix C:

$$C_1 = \begin{bmatrix} -1 & 1 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 0 & 0 & -4 \end{bmatrix} \tag{24}$$

$$C = \begin{bmatrix} C_1 & 0_{6\times4} \\ 0_{4\times6} & 0_{4\times4} \end{bmatrix}$$

And v is the coefficient of the quadric surface:

$$v = [A, B, C, D, E, F, G, H, I, J]^T \tag{25}$$

Eventually, this least-square fitting problem constrained by (22) becomes solving a set of linear equations using Lagrange Multiplier method [9]:

$$\begin{gathered} DD^T v = \lambda C v \\ v^T C v = 1 \end{gathered} \tag{26}$$

where λ is the Lagrange Multiplier, and D is defined as the "design matrix":

$$D_{10\times n} = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 & \dots & x_n^2 \\ y_1^2 & y_2^2 & y_3^2 & \dots & y_n^2 \\ z_1^2 & z_2^2 & z_3^2 & \dots & z_n^2 \\ 2y_1z_1 & 2y_2z_2 & 2y_3z_3 & \dots & 2y_nz_n \\ 2x_1z_1 & 2x_2z_2 & 2x_3z_3 & \dots & 2x_nz_n \\ 2x_1y_1 & 2x_2y_2 & 2x_3y_3 & \dots & 2x_ny_n \\ 2x_1 & 2x_2 & 2x_3 & \dots & 2x_n \\ 2y_1 & 2y_2 & 2y_3 & \dots & 2y_n \\ 2z_1 & 2z_2 & 2z_3 & \dots & 2z_n \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \quad (27)$$

here, $x_n, y_n,$ and z_n are the actual magnetometer readings at an instantaneous time. The detailed solutions of (26) can be found in [10] [12] [13]. But one key point of these solutions is to calculate the coefficient vector v of the ellipsoid. In a nutshell, the hard iron error V and soft iron error W^{-1} are formulated by:

$$V = -Q^{-1}U$$
$$W^{-1} = \frac{H_M}{\sqrt{V^TQV - J}} \quad (28)$$

where:

$$Q = \begin{bmatrix} \mathrm{A} & \mathrm{D} & \mathrm{E} \\ \mathrm{D} & \mathrm{B} & \mathrm{F} \\ \mathrm{E} & \mathrm{F} & \mathrm{C} \end{bmatrix}$$
$$U = \begin{bmatrix} \mathrm{G} \\ \mathrm{H} \\ \mathrm{I} \end{bmatrix} \quad (29)$$

H_m is the local raw magnetic field strength, and its unit is LSB (See).

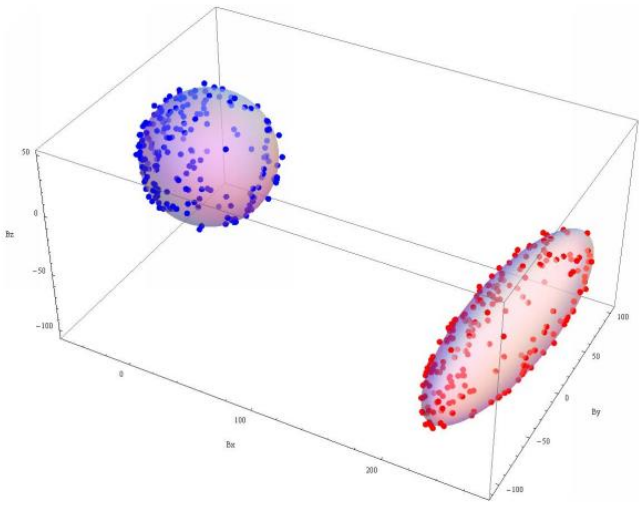


Figure 5: Calibrated (blue) and un-calibrated readings (red) [8]

Since (28) is fundamentally solving v, thereby all the properties of this ellipsoid are determined; eventually V and W^{-1} are calculated. And as described previously in equations (8) and (20), by applying correct V and W^{-1} calculated from EMA, the magnetometer readings will be calibrated and they should map to an ellipsoid shifted by V to the origin and then reshaped by W^{-1} to a sphere, whose radius is equal to the magnitude of the local magnetic field. The entire EMA can be summarized by Figure 6

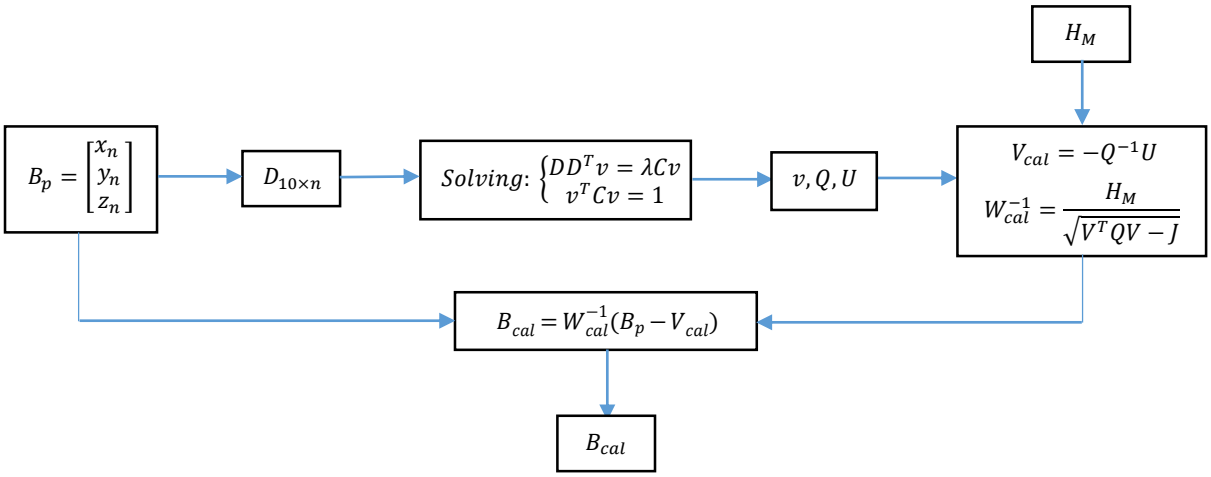


Figure 6: Flowchart of EMA

5 Final design details

5.1 Prototypes

Our project consists of two prototypes: one for static calibration and a second one for a wearable sensor prototype for modeling human body motion.

5.1.1 Static prototype

In order to calculate the A_{inv} matrix, we need to rotate the IMU about a fixed point in the x, y and z axes (yaw, roll, pitch); however, creating three independent rotations about a fixed point can be quite challenging and time consuming. Moreover, one also has to consider the magnetic interference from the servos that could distort the calibration matrix. Therefore, we need to place the servos sufficiently far apart to minimize their magnetic interference to the sensors. Figure 7 shows our implementation of a static prototype with two automated degrees of freedom. The third rotation is achieved manually by rotating the IMU 90 degrees about the x and y axes.

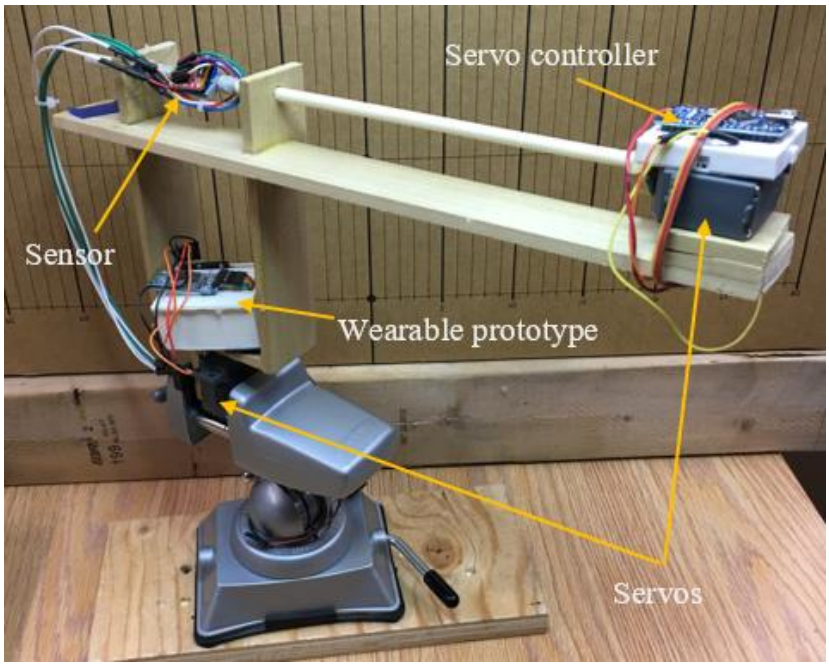


Figure 7: Static prototype with two automated DOF

5.1.2 Wearable prototype

Our final design prototype consists of a wearable sensor with wireless capabilities up to 20 meters, support for multiple IMUs and reliable data acquisition. The device is equipped with

a HC-06 Bluetooth module. The prototype is also cross platform compatible and supports Linux, Windows and Mac operating systems. Figure 8 shows the wearable prototype.

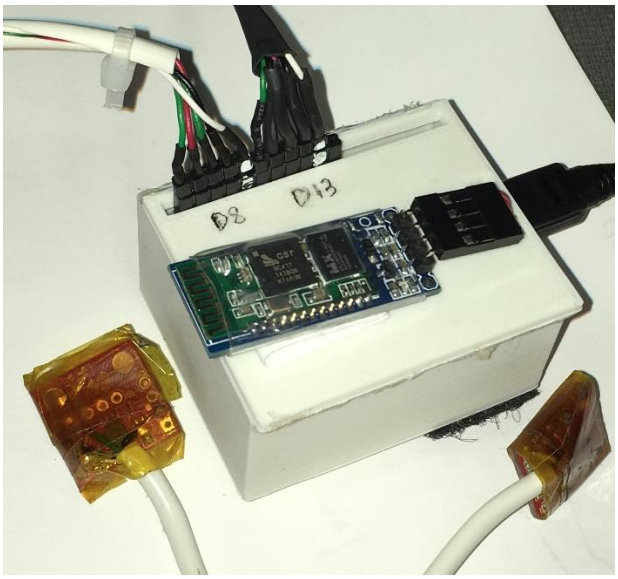


Figure 8: Wearable prototype with wireless capability and multiple IMU support

5.2 Software tools

A few useful tools are presented in the subsections below. These tools are developed to visualize real-time data, calculate and apply calibration, and model the human body motion using a stickman animation.

5.2.1 3D visualization and post-processing

A python script named `3dplot.py` (see Appendix E.1) was created to visualize real-time magnetometer readings in a three-dimensional space. The script uses PyQtGraph, an open source GUI library built on PyQt4 and numpy, to plot 3D graphics. Figure 9 shows an example of a real-time plot from the script. After collecting sufficient data, `static_calibration.py` (see Appendix E.2) script is executed to calculate the calibration matrix, evaluate the standard deviation, variance and a histogram plot of the data set. It also provides a comparison between un-calibrated and calibrated data after applying EMA in real-time.

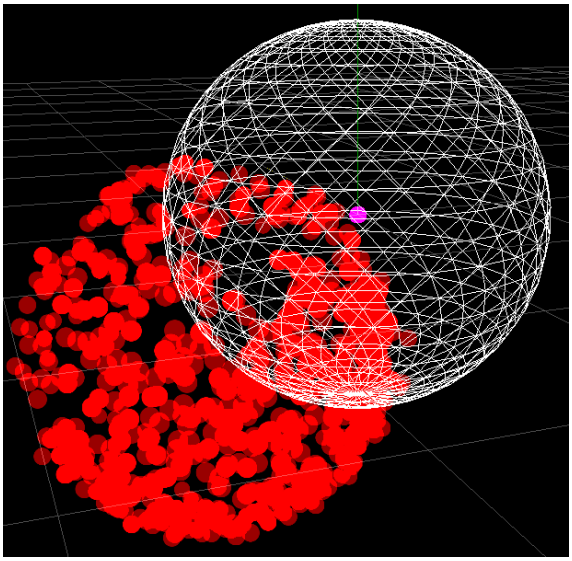


Figure 9: Real-time plot of magnetometer readings

5.2.2 Stickman animation

To visualize the human body motion, a stickman animation script `stickman_animation.py` (see Appendix E.3) was made using a python library called Pygame. For slower movement, it is safe to assume that the torso of the test subject is stationary as this simplifies the tracking algorithm. The IMU is attached just above the knee and models the upper leg of the stickman. The second IMU is attached near the ankle and controls the lower leg.

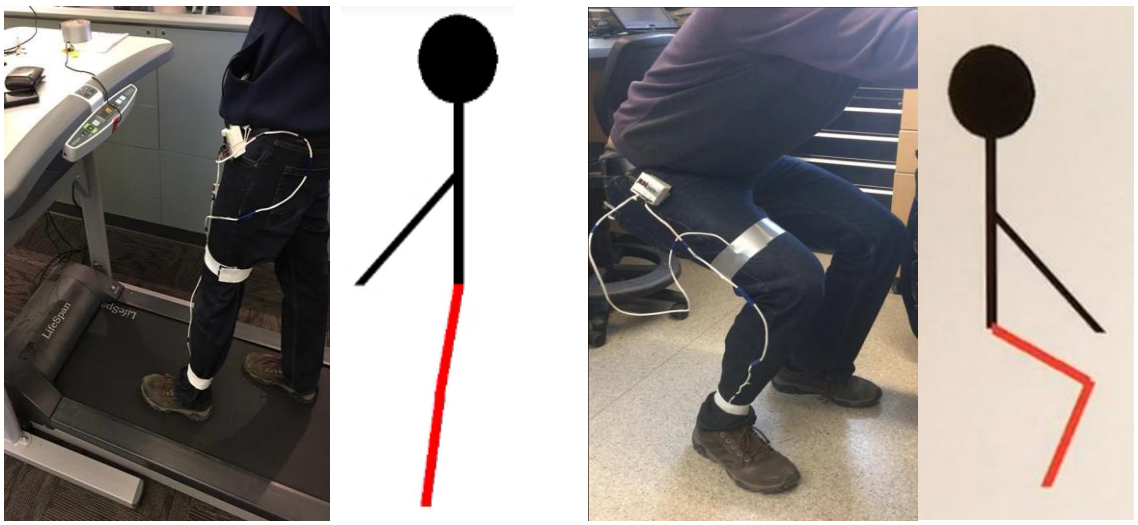


Figure 10: Treadmill and squat animation

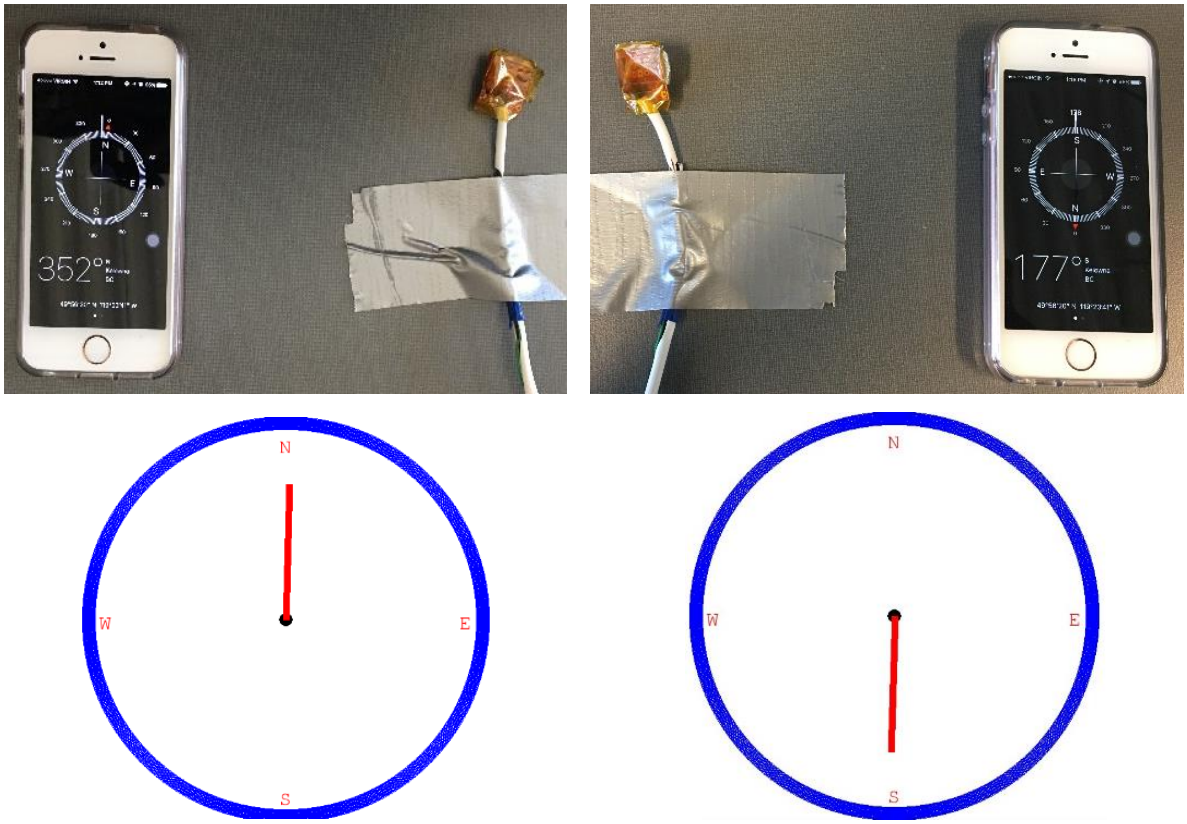


Figure 11: North and South compass animation

Another python script was made to measure the four coordinal directions (NSWE). A simple compass with the four directions was mapped onto the screen. We confirmed the accuracy of the animation by comparing it to a smartphone compass application.

6 Experiments

This section describes the experimental setup and provides justification for the tests which were conducted to verify the calibration algorithm. A detailed discussion and analysis of measurement results is given later in section 7.

Experiments can be subdivided into static and dynamic measurements. The static prototype is used for static measurements and the wearable prototype for dynamic measurements. The purpose of the static measurements is to calculate the calibration matrix in different locations and then analyze the gathered data. The calibration matrix obtained from the static measurements is applied to the dynamic measurements where we focus more on distortion effects.

6.1 Static measurements

To demonstrate the variations in the magnetometer data readings we have conducted three static measurements.

6.1.1 Indoor – EME Foyer

This experiment provides a relationship between the magnetic field strength inside a building compared to free space. We evaluated the magnetic field in different locations within the same building to see how the magnetic field changed. This gave us an insight on how reliable a static calibration would be in different locations within the same building. For example, if a user performed a static calibration at location 1, this experiment will help us evaluate if the same calibration would work in locations 2 and 3. A layout of the experimental setup is shown in the figure below.

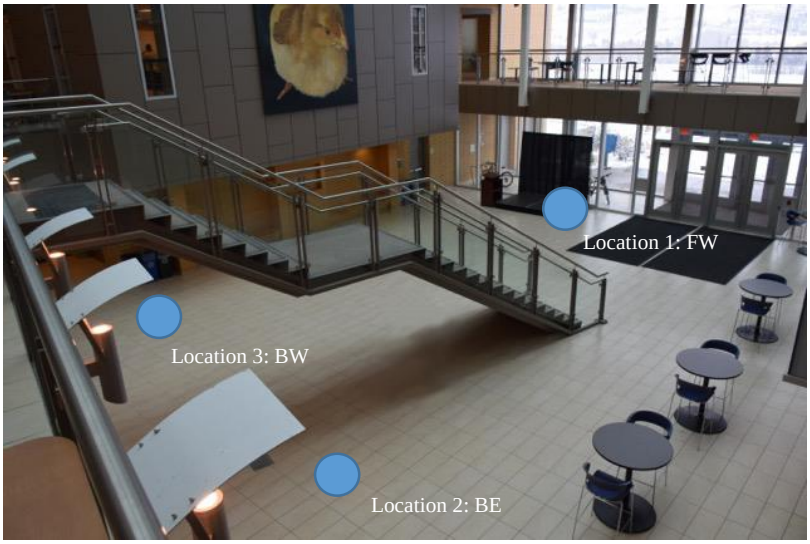


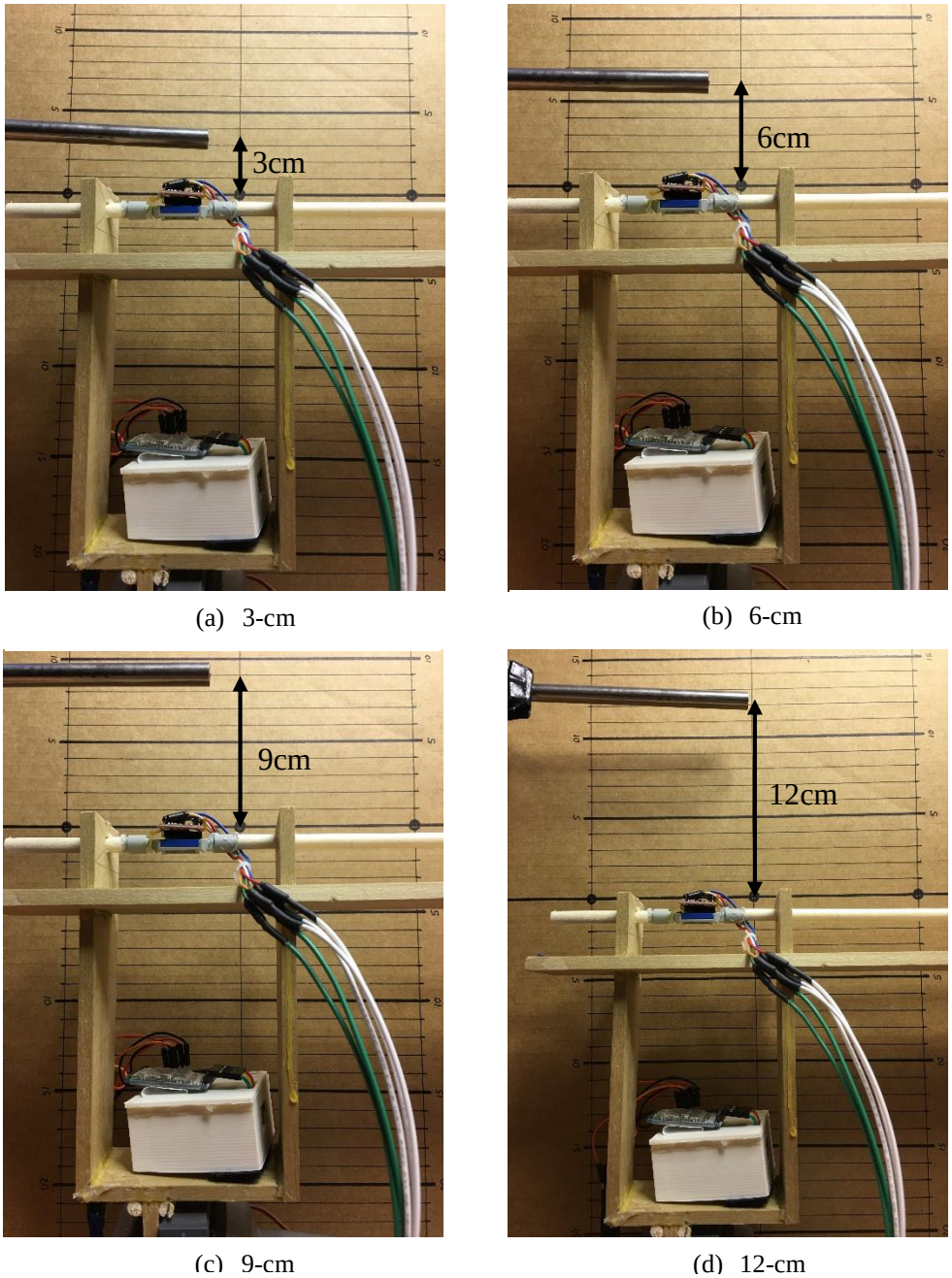
Figure 12: Experimental layout in EME Foyer Level 0

6.1.2 Outdoor – Soccer field

Although soccer is not one of the use cases that we decided to analyze, it provides useful insight on how the magnetometer readings compare to an indoor environment. The purpose of this measurement is to demonstrate the hard iron errors caused by the sensor reference frame. This experiment was relatively simple to perform, as we only needed to run the static prototype to gather data and then evaluate the calibration matrix using the static calibration script.

6.1.3 Metallic interference – Iron rod

To provide a better understanding on the effects of metallic objects on magnetometer readings, we performed the static calibration procedure with an iron rod at different distances. A spatial variation of the iron rod would give us an idea of how close a metal object should be to distort the magnetometer readings. Figure 13 below shows four experiments with 3-cm, 6-cm, 9-cm and 12-cm distances between the IMU and the iron rod.



(a) 3-cm (b) 6-cm

(c) 9-cm (d) 12-cm

Figure 13: Experimental setup for demonstrating metallic interference

6.2 Dynamic measurements

This section presents the experimental setup for dynamic measurements. These experiments use the calibration matrix obtained from the static measurements conducted at the same location. For example, the treadmill test was conducted in the library building; therefore, the calibration matrix from the library is used.

6.2.1 Arm tracking

The arm tracking experiment involves a test subject pivoting their arm about their elbow from the horizontal axis to 90 degrees. This experiment demonstrates how we can track the relative motion of the arm based on a fixed reference position.

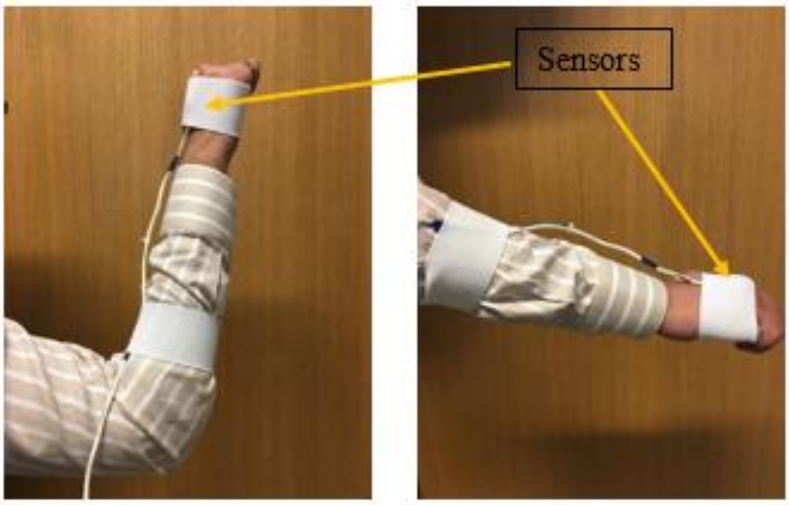


Figure 14: Arm tracking experiment

6.2.2 Treadmill

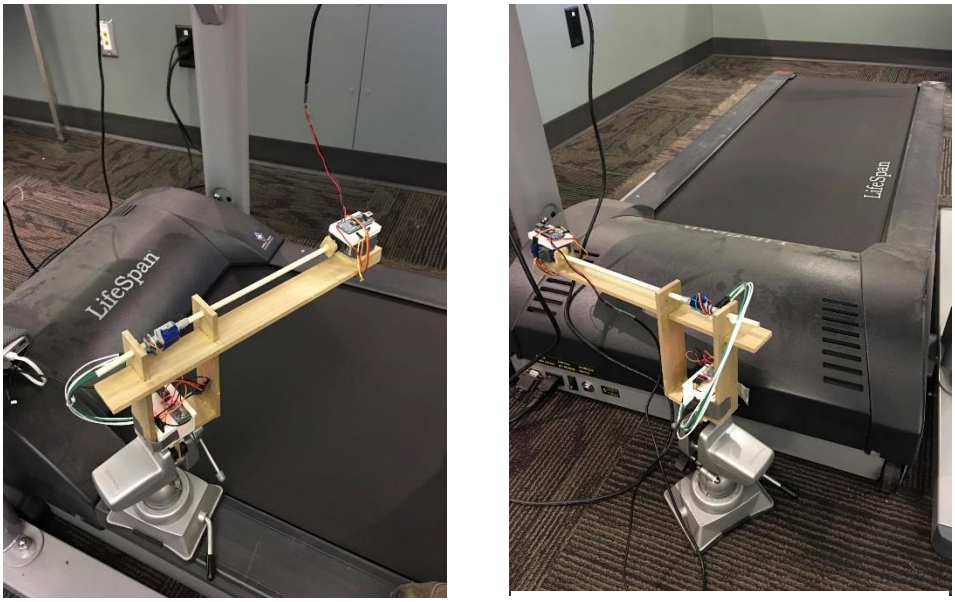
We successfully modeled the human body motion on a treadmill using a stickman animation. This experiment demonstrates the effect of surrounding ferromagnetic interferences and the magnetic field from the electric motor to the sensor. We also performed an interference experiment where we evaluated the magnetometer readings at different positions with respect to the motor and at different operating speeds.

6.2.2.1 Motion capturing

Figure 15: Treadmill - motion capturing

A calibration matrix obtained at the library was used to perform the motion capturing test. We placed two sensors: one on the upper thigh, and another on the lower leg of the test subject (see Figure 15). We then modeled the motion using the animation script.

6.2.2.2 Interference test



(a) Location 1 (b) Location 2

Figure 16: Treadmill - interference test

In order to see if our use cases do not require a dynamic calibration, it is necessary to evaluate how much distortion the electric motor and the surrounding metal would have on the magnetometer. To do this, we ran the static calibration script at different locations around the treadmill and observed the difference in the calibration matrix. This test is similar to the iron rod test; however, we introduced another variable, which is the motor speed at each location. We picked three locations around the treadmill (see Figure 16) and at each location we changed the treadmill speed to 1, 2, and 3 mph.

6.2.3 Squats

Similar to the previous motion tracking experiments, the squat test also demonstrates the dynamic motion of the human body. Instead of a linear motion observed in treadmill and arm tracking experiments, the squat experiment explores multiple range of motions (see Figure 21).

7 Results

7.1 Static measurement results

7.1.1 Indoor measurement results

The indoor measurements are collected from the EME atrium. As section 6.1.1 illustrates, three measurements were conducted at different locations. Each measurement contains the following information: hard iron error V and soft iron error A^{-1}, as well as the scatter plots of both un-calibrated and calibrated sensor readings. For simplicity, the scatter plot and calibration result tables are only provided for Location 1 (see Appendix C for all test locations).

Table 2: Calibration parameters for Location 1

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-21.5192	26.4176	-77.7396
Soft Iron Error A^{-1} (LSB)	1.45872514	0.00213059	-0.00538276
	0.00213059	1.4955676	0.02561731
	-0.00538276	0.02561731	1.51597528

Note: the derivation of these units can be found in Appendix D

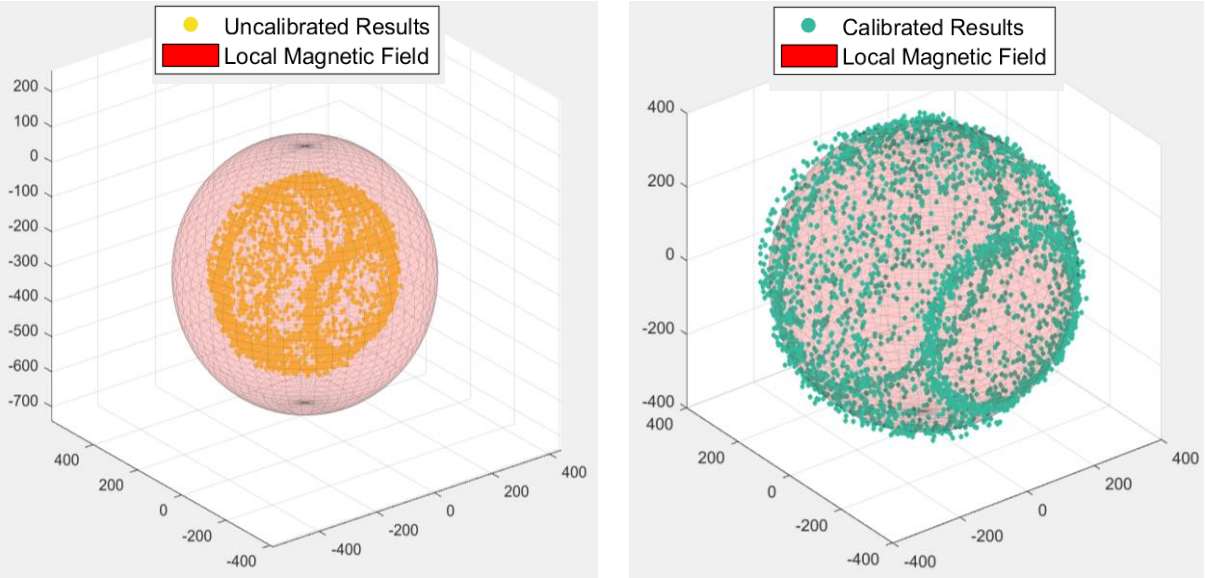


Figure 17: Un-calibrated (left) and calibrated (right) readings scatter plot at Location 1

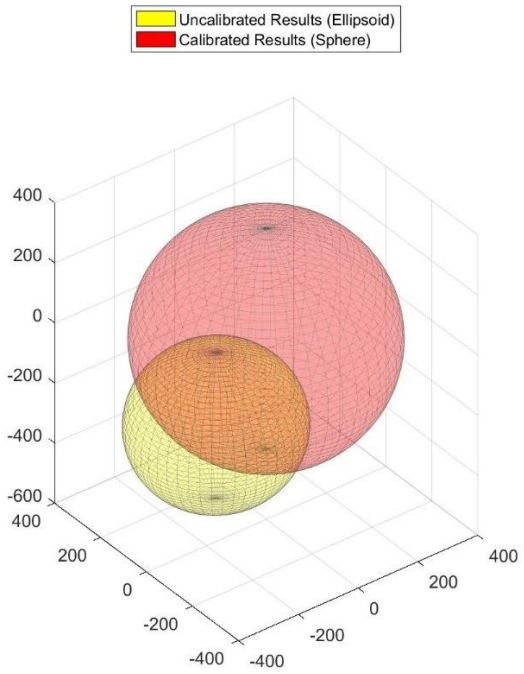


Figure 18: Mapping ellipsoid to sphere at Location 1

The same experiment setup was used in all three tests. From Table C.1 and Table C.2, one can see that the hard iron bias V in the x, y, and z components are almost identical in all the measurements. This is due to the fact that V is defined as the stationary sensor frame. As long as the hardware setup is fixed, this parameter will not fluctuate. This results in an offset of the un-calibrated data shown in Figure 17. The soft iron error A^{-1} is a scaling factor applied to the sensor readings. Particularly, the diagonal component represents how the magnetometer data is scaled by the soft iron error W_{soft} (equation 2). In this scenario, W_{gain}

and $W_{NonOrthog}$ contribute to the off-diagonal components, which should be approximately zero as a reliable magnetometer will have very small gain discrepancies or sensor frame misalignment. For instance, the A^{-1} diagonal of Table C.1 shows that the x, y, and z magnetometer readings are scaled up by roughly 1.5 times after applying EMA. This means the magnetic field at the measuring location is about 67% of the ideal value. In addition, the off-diagonal elements are almost zero, thus the gain and orthogonality defects are within tolerance. Moreover, since the diagonal elements are consistent, the ambient magnetic field is isotropic and has no dominant magnetic distortion in either orientation. We can see that Figure 18 shows the un-calibrated data (yellow) forming a sphere or an ellipsoid with unit semi-major to semi-minor ratio.

In summary, three different tests at different locations within the EME atrium yield similar outcomes. It can be concluded that the same hardware setup will have similar hard iron bias and as long as the diagonal elements are consistent, the local magnetic field distribution is uniform.

7.1.2 Outdoor measurement results

Calibration results of the soccer field test can be found in Table C.3. As expected, the hard iron bias is identical to the above tests since the same hardware setup is used. The diagonal of A^{-1} matrix indicates that the magnitude of the magnetic field is about 90% of the ideal value. It is observed that ambient magnetic field outside is less distorted compared to the EME test results. Also, each diagonal element's value is identical (±0.03) showing that there are no strong magnetic field interferences nearby.

7.1.3 Iron rod measurement results

It is also very critical to evaluate the impact of nearby metallic objects on the accuracy of sensor measurements. The calibration results are summarized in the Table C.4, Table C.5, Table C.6, Table C.7, and Table C.8. Figure 19 shows the relationship between the calibration error and the distance to a nearby metal.

Two important observations can be made: firstly, the ambient magnetic field decreases with the increase in distance from the iron rod to the sensor (from 65% of the ideal value at 3-cm to 55% at 12-cm). This is caused by the magnetic field distortion caused by the iron rod.

From this observation, it can be noted that the iron rod distorts the data; however, the overall animation can still be reconstructed.

Secondly, Figure 19 demonstrates that the closer an iron object is to the magnetometer, the greater the error in the measured data. When the sensor is sufficiently far apart (more than 9-cm) then the standard deviation of the calibrated sensor data is identical to the case with no surrounding iron object.

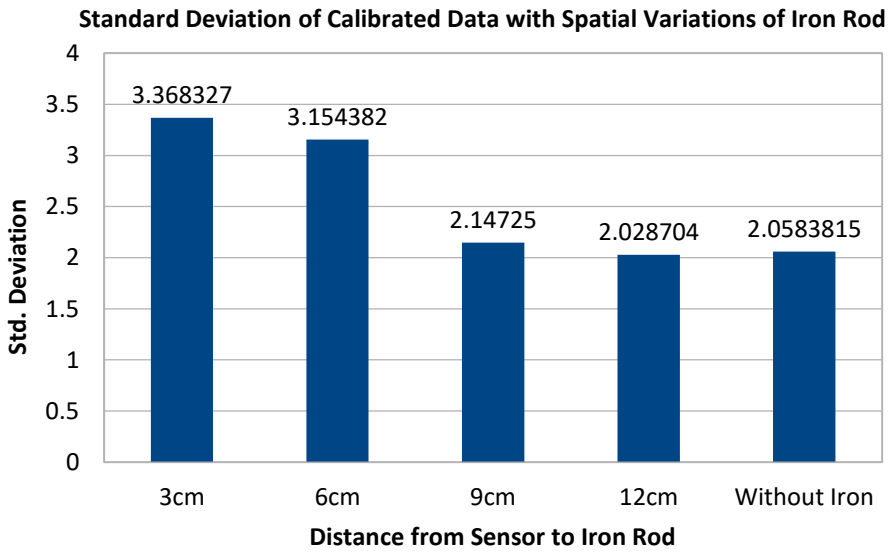


Figure 19: Errors in data with varying distances of iron rod from the sensor

7.2 Dynamic measurement results

7.2.1 Arm tracking results

The test subject repeatedly bent their arm vertically from 0 to 90 degrees with constant speed, and the angle between the horizontal axis and the arm is plotted as a function of time, shown in Figure 20.

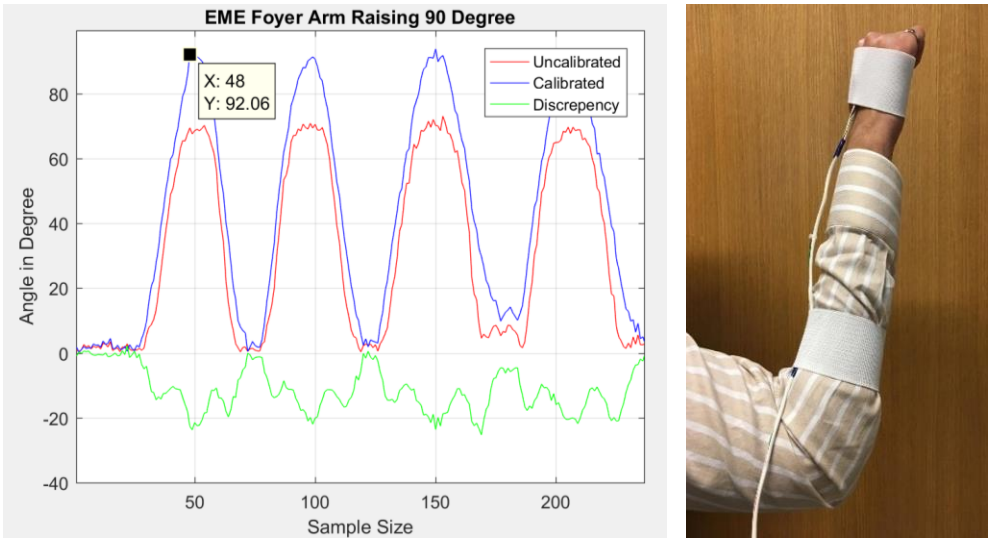


Figure 20: Arm tracking result

Table C.9 was applied to calibrate the dynamic data. We can observe the error in the un-calibrated measurement (red) which is up to 20 degrees. This experiment shows the importance of EMA in order to accurately capture the dynamic motion.

7.2.2 Squat results

Table C.9 was applied to calibrate squat readings. It can be observed that the hard iron offsets are different with respect to previous results. The experiment results are mostly demonstrated by animation. Comparing the stickman animation and actual human body motion visually, we concluded that the animation indeed reconstructed squat motion in real-time.

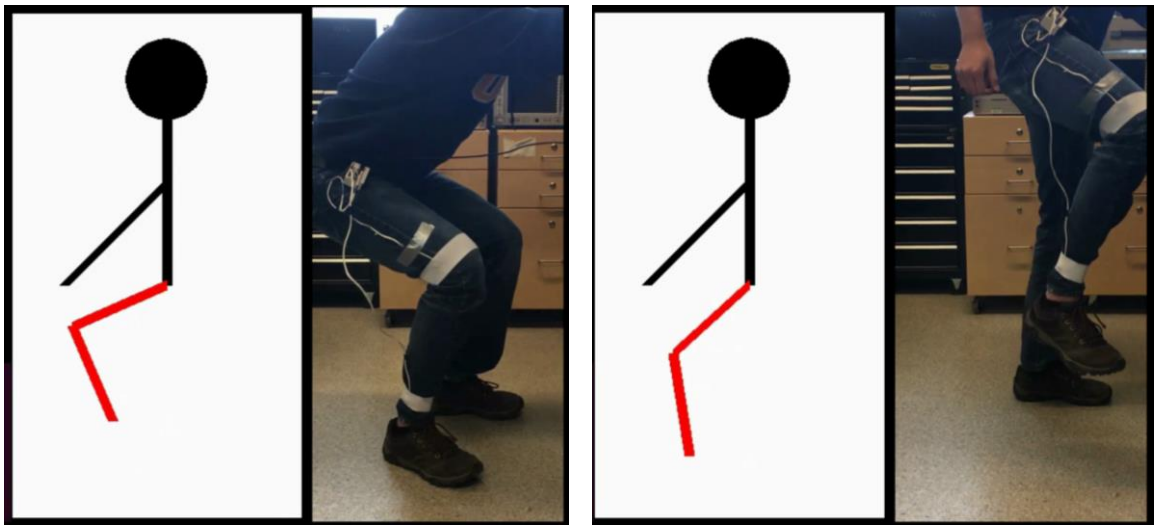


Figure 21: Squat motion and stickman animation

7.2.3 Treadmill results

To analyze the magnetic interference generated from the treadmill motor we conducted static measurements around the motor (see Figure 16). Two variables were of interest here: one is the speed of the treadmill and other is the distance from the sensor to the motor. The results are summarized in Appendix C.

Location 1, which is 15-inches away from the electric motor, the calibrated results are included in Table C.10, Table C.11, Table C.12 and Table C.13. We can see that the diagonal elements of A^{-1} matrix do not fluctuate as the speed increases; however, similar to the iron rod test before, the measured ambient magnetic field at location 1 was about 57% of the ideal value. At location 2, the sensor was placed 5-inches away from the electric motor and a large magnetic field strength was detected. The average diagonal value was below 1, revealing that the electric motor affected the magnitude of the magnetometer readings. In spite of this scaling effect, all x, y, and z components were distorted with approximately the same magnitude, meaning that the interference is fairly isotropic. By using EMA, the treadmill motion was reconstructed using the animation script and both the upper and lower legs were accurately modelled. Thus, we conclude that we do not require a dynamic calibration in our use cases as we found that static calibration was sufficient to model the human body motion.

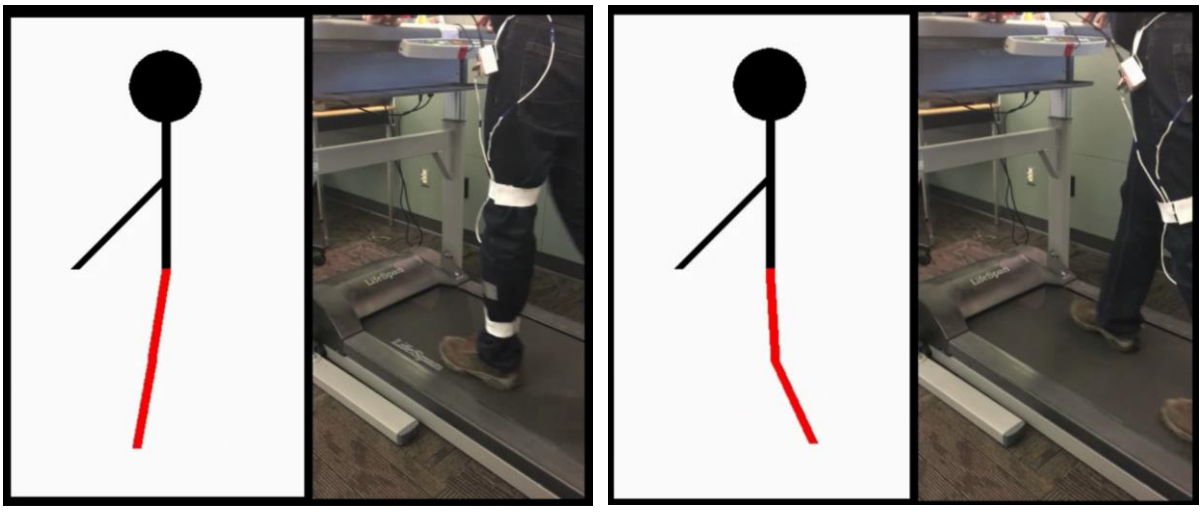


Figure 22: Treadmill motion and stickman animation

8 Design evaluation

8.1 Assessment of performance

The squat use case was performed first. To verify if this use case was a success, we needed to accurately reconstruct the motion using the stickman animation script. We concluded that as the motion of the stickman was relatively similar to the motion of the test subject, the static calibration is sufficient.

We initially thought that in the treadmill use case, the electric motor would cause magnetic interference to the sensor and give worse distorted data, and that the speed of the motor will generate stronger magnetic interference. To verify if the electric motor would cause any distortion we conducted several tests at different locations relative to the motor and at varying speeds. The experiment concluded with the speed of the motor not causing much interference, and the motor did not generate any noticeable distortion. However, the distance of the prototype to the motor did cause a change in the measured data.

In both of our use cases, we were able to model the human body motion using the stickman animation. We concluded that the static calibration data is significantly important and is sufficient to track motion. Thus, in our use cases we did not require a dynamic calibration.

8.2 Recommendations and future work

To be certain the findings are accurate, there are two possible improvements. This includes analyzing more use cases with more dynamic motion. Some examples of dynamic motion can be sprinting, tackling, and rolling. The experiments are not limited to only land based sports and can include aquatic sports such as swimming, water polo, and diving. We experimented with two sensors, but for future improvements more IMUs can be included.

For simplicity, our data range only looked at 2D plane. This was done as the 2D movement was easier to implement and demonstrate. To further improve the accuracy and range of data, a 3D animation can be adopted for future works.

9 References

[1] "MPU -9250 Product Specification Revision 1.0," 2014.

[2] "AK8963 3-axis Electronic Compass, 1st ed.," 2013.

[3] Geomag, "Magnetic declination," 2015. [Online]. Available: http://geomag.nrcan.gc.ca/mag_fld/magdec-en.php. [Accessed 2016].

[4] N. Bowditch, "The American Practical Navigator," 1995.

[5] M. Kok and T. B. Schon, "Magnetometer calibration using inertial sensors," *IEEE Sensors Journal*, vol. 16, no. 14, pp. 5679-5689, 2016.

[6] R. Alonso and M. D. Shuster, "Complete linear attitude-independent magnetometer calibration," *The Journal of the Astronautical Sciences*, vol. 50, no. 4, pp. 477-490, 2002.

[7] K. Winer, "Simple and Effective Magnetometer Calibration".

[8] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira and B. Cardeira, "Geometric approach to strapdown magnetometer calibration in sensor frame," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 2, pp. 1293-1306, 2011.

[9] V. Renaudin, M. H. Afzal and G. Lachapelle, "Complete triaxis magnetometer calibration in the magnetic domain," *Journal of Sensors*, 2010.

[10] Q. Li and J. G. Griffiths, "Least Square Ellipsoid Specific Fitting," *IEEE Xplore*, 2004.

[11] T. Ozyagcilar, "Calibrating an eCompass in the Presence of Hard- and Soft-Iron," *NXP*, 2015.

[12] A. Fitzgibbon, M. Pilu and R. B. Fisher, "Direct Least Square Fitting of Ellipses," *PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 21, no. 5, 1999.

[13] F. L. Bookstein, "Fitting Conic Sections to Scattered Data," *Computer Graphics and Image Processing*, vol. 9, pp. 56-71, 1979.

[14] K. Kanatani, "Statistical Bias of Conic Fitting and Renormalization," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 16, no. 3, pp. 320-326, 1994.

[15] K. Y. a. C. Hwang, "Multiple attribute decision making - An introduction," 1995.

[16] R. Alonso and M. D. Shuster, "A Fast Robust Algorithm for Attitude-Independent Magnetometer-Bias determination".

Appendix A Glossary

IMU	Inertial Measurement Unit
EMA	Ellipsoid Mapping Algorithm
PCB	Printed Circuit Board
EME	Engineering Management and Education
LSB	Least Significant Bit

Appendix B Multi-criteria optimization on selecting use cases

There exists a large variety of sports and narrowing it down to a few was difficult. Therefore, the few sports and activities mentioned were chosen with the following factors in mind; accessibility, limited ferromagnetic material in the vicinity, and testability.

Table B.1: Six types of activities

A1	Squat	A4	Ice Skating
A2	Treadmill	A5	Weight Lifting
A3	Swimming	A6	Soccer

Therefore, a Multi Criteria Decision Making process (MCDM) was adopted. This type of decision making is unbiased and is strictly dependent on numbers and weights. Two MCDM methods were selected for analyzing the measurements; these are the Simple Additive Method (SAW) and the Weighted Product Method (WPM). These two types of decision making are simple and widely used in MCDM [15].

Table B.2: Weighted table

		Alternative						
Attributes	**Weights**	A1	A2	A3	A4	A5	A6	**Max**
Degree of Motion*	0.15	10	10	1	1	10	1	10
Modelling in 2D *	0.10	10	10	8	1	8	1	10
Magnetic Interference *	0.40	8	8	9	7	7	10	10
Simplicity of testing *	0.10	10	10	1	3	7	7	10
								Min
# of IMU to Model Motion	0.15	4	4	8	4	6	4	4
Drop-in Rate	0.10	0.01	0.01	8.5	4.5	5	0.01	4.5

* Units are from a 10-point scale, from 1 (worst) to 10 (best).

The six attributes were chosen and weighted based on the importance and relevance to the data collection. The weights of the magnetic interference are weighed heavily because we found that this was the most important part in data collection. Degree of motion and number of IMUs were also weighted more than the others because the complexity of the motion will disrupt the data collection, and the number of IMU is hard to implement once it reaches a staggering number. In the degree of motion and modelling in the 2D section, 10 means the

simplest motion is easy to track and 1 representing the difficulty of tracking. Simplicity of testing says 10 is the easiest to test and 1 being the hardest.

To normalize the data, C1 – C4 are found by using this formula for linear normalization:

$$x_{ij} = \frac{x_{ij}}{\max(x_{ij})}$$

and C5 – C6 are found by using this formula for monetary value: (where the lower number is preferred)

$$x_{ij} = \frac{\max(x_{ij})}{x_{ij}}$$

Table B.3: Normalized data

	A1	A2	A3	A4	A5	A6
C1	1.00	1.00	0.10	0.10	1.00	0.10
C2	1.00	1.00	0.80	0.10	0.80	0.10
C3	0.80	0.80	0.90	0.70	0.70	1.00
C4	1.00	1.00	0.10	0.30	0.70	0.70
C5	1.00	1.00	0.50	1.00	0.67	1.00
C6	0.00	0.00	0.53	1.00	0.90	0.00

The first method applied to the above table was the SAW method, this type of method is dependent on the weight and will greatly affect the results and ranking. The formula used in the SAW method is

$$V(A_i) = \sum_{j=1}^{6} Weight_i \cdot x_{ij}$$

This formula takes the sum of each xij relative to the weights and gives a ranking result. As seen in Table B.4, A1 and A2 tie for first, followed by A5, A6, A3, and A4.

Table B.4: SAW ranking

VA	A1	A2	A3	A4	A5	A6
	0.820	0.820	0.593	0.585	0.770	0.645

A second MCDM test was conducted to verify if the first method was correct. The WPM does not need the data to be normalized and is found by using

$$W(A_i) = \prod_{j=1}^{6} x_{ij}^{Weight_j}$$

Table B.5: WPM ranking

W	A1	A2	A3	A4	A5	A6
	4.177601	4.1776	1.752268	1.698717	2.993907	2.478568

The WPM gives the same result as the SAW method, and thus A1 and A2 were selected for experimentation.

Appendix C Calibrated parameters for each experiment

Table C.1: Calibration parameters for Location 2

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-23.7191	25.8491	-79.7497
Soft Iron Error A^{-1} (LSB)	1.3644117	-0.02876942	0.00879457
	-0.02876942	1.40920841	0.02205621
	0.00879457	0.02205621	1.4039751

Table C.2 Calibration parameters for Location 3

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-21.8094	26.7798	-78.8359
Soft Iron Error A^{-1} (LSB)	1.62053788	0.04049994	-0.00900594
	0.04049994	1.63700836	0.02286413
	-0.00900594	0.02286413	1.65092567

Table C.3 Calibration parameters for soccer field

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-11.235417	26.369801	-72.719978
Soft Iron Error A^{-1} (LSB)	1.123152	-0.028356	0.002464
	-0.028356	1.138645	0.034030
	0.002464	0.034030	1.160710

Table C.4 Calibration parameters for iron rod 3cm

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-24.931214	13.968291	-73.851471
Soft Iron Error A^{-1} (LSB)	1.544635	-0.070067	0.019779
	-0.070067	1.693021	0.019148
	0.019779	0.019148	1.570299

Table C.5 Calibration parameters for iron rod 6cm

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-13.942923	25.528779	-78.024124
	1.713417	0.020241	-0.009076

Soft Iron Error A^{-1} (LSB)	0.020241	1.689201	0.023976
	-0.009076	0.023976	1.693885

Table C.6 Calibration parameters for iron rod 9cm

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-14.013501	26.234846	-77.833084
Soft Iron Error A^{-1} (LSB)	1.746986	-0.004439	-0.004551
	-0.004439	1.818080	0.026821
	-0.004551	0.026821	1.830637

Table C.7 Calibration parameters for iron rod 12cm

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-14.013501	26.234846	-77.833084
Soft Iron Error A^{-1} (LSB)	1.707860	0.001922	-0.005250
	0.001922	1.772085	0.024127
	-0.005250	0.024127	1.785207

Table C.8 Calibration parameters without iron rod

Calibrated Results (14-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-14.122966	25.801856	-77.923390
Soft Iron Error A^{-1} (LSB)	1.622833	0.016320	-0.012252
	0.016320	1.697670	0.023570
	-0.012252	0.023570	1.708936

Table C.9 Calibration parameters in Microwave lab

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-297.84494954	9.45077617053	-178.231899118
Soft Iron Error A^{-1} (LSB)	1.232502	-0.024398	-0.032277
	-0.024398	1.314246	-0.010070
	-0.032277	-0.010070	1.271461

Table C.10 Calibration parameters at Location 1 with speed 0

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-293.08948587	74.6508333707	-243.97338062

Soft Iron Error A^{-1} (LSB)	1.834875	-0.073242	-0.023924
	-0.073242	1.726339	0.011754
	-0.023924	0.011754	1.702158

Table C.11 Calibration parameters at Location 1 with speed 1 miles/hr.

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-281.793765881	82.7682431498	-249.773528549
Soft Iron Error A^{-1} (LSB)	1.784980	-0.064495	-0.007440
	-0.064495	1.785911	-0.009001
	-0.007440	-0.009001	1.736080

Table C.12 Calibration parameters at Location 1 with speed 2 miles/hr.

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-287.343485973	72.1508295741	-248.990304474
Soft Iron Error A^{-1} (LSB)	1.736297	-0.027302	-0.021580
	-0.027302	1.732159	-0.003896
	-0.021580	-0.003896	1.670623

Table C.13 Calibration parameters at Location 1 with speed 3 miles/hr.

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-292.890401904	74.569715087	-246.275297225
Soft Iron Error A^{-1} (LSB)	1.717860	0.002982	-0.000683
	0.002982	1.623288	0.018899
	-0.000683	0.018899	1.607769

Table C.14 Calibration parameters at Location 2 with speed 0

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-263.602380269	41.3691814233	-262.543385064
Soft Iron Error A^{-1} (LSB)	0.619110	-0.011889	-0.001038
	-0.011889	0.630580	0.000583
	-0.001038	0.000583	0.610248

Table C.15 Calibration parameters at Location 2 with speed 1

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis

Hard Iron Bias V (LSB)	-272.050691812	54.186463009	-265.573126519
Soft Iron Error A^{-1} (LSB)	0.679557	-0.012388	-0.000325
	-0.012388	0.680415	0.001826
	-0.000325	0.001826	0.653914

Table C.16 Calibration parameters at Location 2 with speed 2

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-257.169742231	69.0592080497	-258.445480306
Soft Iron Error A^{-1} (LSB)	0.705807	0.0126644	-0.007704
	0.012644	0.667183	0.006610
	-0.007704	0.006610	0.662231

Table C.17 Calibration parameters at Location 2 with speed 3

Calibrated Results (16-bit)	X-Axis	Y-Axis	Z-Axis
Hard Iron Bias V (LSB)	-264.190121855	57.8587391054	-264.145191299
Soft Iron Error A^{-1} (LSB)	0.684689	-0.007972	-0.002254
	-0.007972	0.691162	0.001763
	-0.002254	0.001763	0.668134

Appendix D Local ideal magnetic field and conversion to LSB

To obtain the Gravitation Field (raw format): get the Total Field for your location from here: Say in Kelowna:

1) The total field: 55,092.4 nT
2) Covert this values to Gauss: 55,092.4 nT = 0.551 G
3) Convert total field to raw value total field in LSB, which is essentially the raw gravitation field H_m
    a. $\text{Gain} = \frac{1}{\text{Sensitivity Factor}}$:
        i. For 14-bit resolution: $\text{Gain} = \frac{1}{\left[0.6\left(\frac{\mu T}{LSB}\right)\right]} = 1.67\left(\frac{LSB}{\mu T}\right) = 0.00167\left(\frac{LSB}{nT}\right) = 167\left(\frac{LSB}{Gauss}\right)$
        ii. For 16-bit resolution: $\text{Gain} = \frac{1}{\left[0.15\left(\frac{\mu T}{LSB}\right)\right]} = 6.67\left(\frac{LSB}{\mu T}\right) = 0.00667\left(\frac{LSB}{nT}\right) = 667\left(\frac{LSB}{Gauss}\right)$
    b. $\text{Raw Total Field} = \text{Gain} \times \text{Total Field}$
        i. For 14-bit resolution: $\text{Raw Total Field} = 167 \times 0.551 = 92$ LSB
        ii. For 16-bit resolution: $\text{Raw Total Field} = 667 \times 0.551 = 367.5$ LSB
4) The value read from the magnetometer we used is the magnetometer values in mGauss

3.3 Magnetometer Specifications

Typical Operating Circuit of section 4.2, VDD = 2.5V, VDDIO = 2.5V, T_A=25°C, unless otherwise noted.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
MAGNETOMETER SENSITIVITY					
Full-Scale Range			±4800		µT
ADC Word Length			14		bits
Sensitivity Scale Factor			0.6		µT / LSB
ZERO-FIELD OUTPUT					
Initial Calibration Tolerance			±500		LSB

Note: The sensitivity on the specification is essentially the gain with the unit flipped.

Appendix E Source code

Appendix E.1 3dplot.py

```
"""
This script enables plotting data in real time using the pyqtgraph library.

Date Modified   Modified by     Changes
Nov 22, 2016    Vipul Vishnoi   Initial commit
Jan 24, 2017    Vipul Vishnoi   Added writing data to a file
Feb 06, 2017    Vipul Vishnoi   Added multiple IMU support
Feb 10, 2017    Vipul Vishnoi   Minor updates
Feb 11, 2017    Zonghao Li        Modified plot defaults

"""

from pyqtgraph.Qt import QtCore, QtGui
import pyqtgraph.opengl as gl
import numpy as np

import serial
import sys
import time
import multiprocessing
import os

"""
 Name:      Serial_data_get
 Purpose:   To get serial data from port and modify message before pushing
            to queue.
 Param:     q           - Queue handle
            sample_name - File name to store mag data
"""
def serial_data_get(q, sample_name):
    # Serial port reading setup
    ser = serial.Serial(sys.argv[1], int(sys.argv[2]))

    # Output magnetometer data stream defined
    fd = open("../../data/%s/%s" % (sample_name, sample_name +
"_data.txt"), "w")
    fd2 = open("../../data/%s/%s" % (sample_name, sample_name +
"_data2.txt"), "w")
    os.chmod("../../data/%s/%s" % (sample_name, sample_name + "_data.txt"),
0777)
    os.chmod("../../data/%s/%s" % (sample_name, sample_name +
"_data2.txt"), 0777)

    #Acquiring data
    while True:
        # Read Serial data
        try:
            line = ser.readline()
            try:
                line = line.rstrip('\n').rstrip('\r')
                ID = int(line.split(' ')[0]) # Support multiple imu;
associate them different ID
                x = float(line.split(' ')[1]) # x sensor reading
                y = float(line.split(' ')[2]) # y sensor reading
```

```
                z = float(line.split(' ')[3]) # z sensor reading
                mag_xyz = [ID, x, y, z]

                if (ID == 1):
                    # if sensor 1 reads data
                    fd.write("%.5f %.5f %.5f\n" % (mag_xyz[1], mag_xyz[2],
mag_xyz[3]))
                elif (ID == 2):
                    # if sensor 2 reads data
                    fd2.write("%.5f %.5f %.5f\n" % (mag_xyz[1], mag_xyz[2],
mag_xyz[3]))
                else:
                    print "Invalid IMU ID"
                    raise

                # Put sensor data to queue
                q.put(mag_xyz)

                # Print sensor readings in terminal
                print "Queuing: " + str(mag_xyz)
                time.sleep(0.1)

            # Exception handler
            except KeyboardInterrupt:
                raise

            except:
                print "Invalid data"
                pass

        except serial.serialutil.SerialException:
            print "Serial read fail, trying again"
            time.sleep(0.2)
            pass

        except KeyboardInterrupt:
            fd.close()
            fd2.close()
            sys.exit(1)

"""
 Name:      serial_data_qtplot
 Purpose:   To plot real time data from queue.
 Param:     q - Queue handle
"""
def serial_data_qtplot(q):
    kelowna_mag_raw = 367.5 # Kelowna reference magnetic field strength
    gain = 0.667 # gain of the sensor
    app = QtGui.QApplication([]) # generate plotting object, using
pyqtgraph application
    w = gl.GLViewWidget()
    w.opts['distance'] = 3000 # This is the initial zoom of the plot
    w.show()
    w.setWindowTitle('Sample magnetometer plots')

    # Define grid resolution
    g = gl.GLGridItem()
    g.scale(500, 500, 500)
    w.addItem(g)

    # Define axes properties
```

```
    ax = gl.GLAxisItem()
    ax.setSize(1000, 1000, 1000)
    w.addItem(ax)

    # Draw reference sphere
    md = gl.MeshData.sphere(rows=25, cols=25, radius=kelowna_mag_raw/gain)
    m4 = gl.GLMeshItem(meshdata=md, smooth=False, drawFaces=False,
drawEdges=True, edgeColor=(1,1,1,0.2))
    m4.translate(0,10,0)
    w.addItem(m4)

    # Maximum depth of queue. Increase this number to plot more data
    max_index = 10000
    index = 0
    index_2 = 0
    p = np.zeros((max_index, 3))
    p_2 = np.zeros((max_index, 3))

    # Draw scatter plot; White (1,1,1), Red(255,0,0), Blue(0,0,255),
Green(0,128,0)
    sp = gl.GLScatterPlotItem(pos=p, color=(255, 0, 0, 0.6), size=50,
pxMode=False)
    sp_2 = gl.GLScatterPlotItem(pos=p_2, color=(0, 0, 255, 0.6), size=50,
pxMode=False)
    w.addItem(sp)
    w.addItem(sp_2)
    wait = 0
    while True:
        if not q.empty():
            wait = 0
            data = q.get() # Require data from q

            if (data[0]==1): # Plot scatter data from IMU 1
                p[index] = (data[1], data[2], data[3])
                index += 1
                if index > max_index - 1:
                    index = 0
                sp.setData(pos=p)

            elif (data[0]==2):  # Plot scatter data from IMU 2
                p_2[index] = (data[1], data[2], data[3])
                index_2 += 1
                if index_2 > max_index - 1:
                    index_2 = 0
                sp_2.setData(pos=p_2)

            app.processEvents()

        # If queue is empty, wait for data in queue. Exit if nothing
        else:
            if (wait > 20):
                break
            else:
                time.sleep(0.5)
                wait += 1

    print "No more data to plot, Exiting"
    sys.exit(app.exec_())


"""
```

```
 Name:      main
 Purpose:   Initialize threads and wait for valid data to appear,
            then start 2nd process to start plotting real time data in
            queue
 Param:     None

"""
def main():
    print "\n"
    print "#-------------------------------------------------------#"
    print "This script is used to read magnetometer readings only."
    print "Press ctrl + c to stop and save the readings.\nPress ctrl + z to
exit the program."
    print "#-------------------------------------------------------#"
    sample_name = raw_input('Enter file name to store data\n')
    if not os.path.exists("../../data/%s" % sample_name): # User define the
measurement name here
        os.makedirs("../../data/%s" % sample_name)
        os.chmod("../../data/%s" % sample_name, 0777)

    q = multiprocessing.Queue() # multiprocessing queue handler

    p1 = multiprocessing.Process(target=serial_data_get, args=(q,
sample_name)) # Processing serial read
    p2 = multiprocessing.Process(target=serial_data_qtplot, args=(q,)) #
Processing scatter data plot
    p1.start()
    while q.empty():
        print "Queue is empty"
        time.sleep(1)
    p2.start()

if __name__ == '__main__':
    main()
```

Appendix E.2 Static_calibration.py

```
"""
This script enables static data processing (EMA) of Magnetometer data

Date Modified   Modified by     Changes
Feb 2, 2017     Vipul Vishnoi   Initial commit
Feb 11, 2017    Zonghao Li      Add some instrutions on the purpose of this
program;
                                minor changes on the output of
"_data_stats" file
                                it will now print out the combined bias and
A_inv on the terminal directly
Feb 13, 2017    Zonghao Li        Update the gain of the sensor

"""

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
import numpy.matlib
import scipy.linalg as scla
import argparse
import re


"""""
 Name:      static_analysis
 Purpose:   Plot calibrated and uncalibrated data sets as well as evaluate
the mean and variance
 Param:     raw and cal data
            file_name - file name to store plots
"""
def static_analysis(x_raw, y_raw, z_raw, x_cal, y_cal, z_cal, A_inv,
x_bias, y_bias, z_bias, file_name, mag_raw):
    # To plot uncalibrated data
    raw_fig = plt.figure()
    ax1 = raw_fig.gca(projection='3d')

    # To plot calibrated data
    cal_fig = plt.figure()
    ax2 = cal_fig.gca(projection='3d')

    # Data histogram
    diff_fig = plt.figure()
    ax3 = diff_fig.gca()

    ax1.plot_wireframe(x_raw, y_raw, z_raw, color ='r')
    ax1.set_xlabel('x')
    ax1.set_ylabel('y')
    ax1.set_zlabel('z')

    ax2.plot_wireframe(x_cal, y_cal, z_cal, color='g')
    ax2.set_xlabel('x')
    ax2.set_ylabel('y')
    ax2.set_zlabel('z')

    diff = (numpy.sqrt(numpy.square(x_cal) + numpy.square(y_cal) +
numpy.square(z_cal))) - mag_raw
```

```
    f_stat = open("../../data/%s_Bias_&_A_inv.txt" % file_name, 'w')
    f_stat.write("%s  %s  %s\n\n" % (x_bias, y_bias, z_bias))
    f_stat.write("%f  %f  %f\n%f  %f  %f\n%f  %f  %f\n\n" % (A_inv[0,0],
A_inv[0,1], A_inv[0,2], A_inv[1,0], A_inv[1,1], A_inv[1,2], A_inv[2,0],
A_inv[2,1], A_inv[2,2]))
    f_stat.write("%s  %s  %s" % (numpy.mean(diff), numpy.var(diff),
numpy.std(diff)))
    f_stat.close()


    hist, bins = np.histogram(diff, bins=50)
    width = 0.7 * (bins[1] - bins[0])
    center = (bins[:-1] + bins[1:]) / 2
    ax3.bar(center, hist, align='center', width=width)

    raw_fig.savefig("../../data/%s_uncal.png" % file_name)
    cal_fig.savefig("../../data/%s_cal.png" % file_name)
    diff_fig.savefig("../../data/%s_hist.png" % file_name)


"""
 Name:      static_cal
 Purpose:   Calculate A inverse matrix, bias and calibrate the magnetometer
data set
 Param:     sample_path - Path to magnetometer data text file
"""
def static_cal(sample_path):
    mag_raw = 367.5 # Kelowna's raw magnetic field

    # Load data set and parse file_name from sample_path
    data = np.loadtxt(sample_path)
    f_st = re.search('/data/', sample_path)
    f_end = re.search('.txt', sample_path)
    file_name = sample_path[f_st.end():f_end.start()]

    # Write raw data to text file
    gain =  0.667
    raw_data = data* gain
    np.savetxt("../../data/%s_raw.txt" % file_name, raw_data, '%.5f')
    column_num = raw_data.shape[0]

    D = np.ones((10, column_num))
    D[0,:] = np.square(raw_data[:, 0]) # x ^ 2
    D[1,:] = np.square(raw_data[:, 1]) # y ^ 2
    D[2,:] = np.square(raw_data[:, 2]) # z ^ 2
    D[3,:] = 2 * raw_data[:, 1] * raw_data[:, 2] # 2yz
    D[4,:] = 2 * raw_data[:, 0] * raw_data[:, 2] # 2xz
    D[5,:] = 2 * raw_data[:, 0] * raw_data[:, 1] # 2xy
    D[6,:] = 2 * raw_data[:, 0] # 2x
    D[7,:] = 2 * raw_data[:, 1] # 2y
    D[8,:] = 2 * raw_data[:, 2] # 2z
    D[9,:] = np.ones((1, column_num)) #1

    # S = D * DT
    S = np.dot(D, D.transpose())

    # Split S matrix to 4 sub-matrices and evalute the SS matrix
    S_11 = S[0:6,0:6]
    S_12 = S[0:6,6:10]
    S_21 = S_12.transpose()
    S_22 = S[6:10, 6:10]
```

```
    S_22_i = np.linalg.inv(S_22)

    # SS = S_11 - S_12 * S_22_i * S_21
    SS = S_11 - np.dot(np.dot(S_12,S_22_i), S_21)

    # Constraint Matrix C
    # C = [-1, 1, 1, 0, 0, 0;
    #       1,-1, 1, 0, 0, 0;
    #       1, 1,-1, 0, 0, 0;
    #       0, 0, 0,-4, 0, 0;
    #       0, 0, 0, 0,-4, 0;
    #       0, 0, 0, 0, 0,-4];
    # Normalized C:
    C = [[0, 0.5, 0.5, 0, 0, 0],
    [0.5, 0, 0.5, 0, 0, 0],
    [0.5, 0.5, 0, 0, 0, 0],
    [0, 0, 0, -0.25, 0, 0],
    [0, 0, 0, 0, -0.25, 0],
    [0, 0, 0, 0, 0, -0.25]]

    C_i = np.linalg.inv(C)

    # Find eigenvalue of matirix E, E * V - V * D = [0]
    E = np.dot(C, SS)
    w, v = np.linalg.eig(E)

    # Find the zero - based position of the only positive eigenvalue.
    # The associated eigenvector will be in the corresponding column of
matrix
    max_eigen = w[0]
    i = 0
    for k in w:
        if k>0:
            max_eigen = k
            break
        else:
            i = i+1

    # Check the sign of V1, if it is negative, flip the sign
    V1 = v[:, i]
    if V1[0] < 0:
        V1 = -1 * V1

    V2 = np.dot(np.dot(S_22_i, S_21), V1)

    # Setup Ellipsoid matrix
    Ellipsoid = np.ones((10, 1))
    Ellipsoid[0:6,0] = V1
    Ellipsoid[6:10,0] = -1 * V2[0:5]

    # Normalize Ellipsoid matrix to prevent trivial results
    Ellipsoid = Ellipsoid/np.linalg.norm(Ellipsoid)

    # Calculate the combined bias
    Q = [[Ellipsoid[0,0],Ellipsoid[5,0],Ellipsoid[4,0]],
[Ellipsoid[5,0],Ellipsoid[1,0],Ellipsoid[3,0]],
[Ellipsoid[4,0],Ellipsoid[3,0],Ellipsoid[2,0]]]
    Q_i = np.linalg.inv(Q)

    U = [[Ellipsoid[6,0]],[Ellipsoid[7,0]],[Ellipsoid[8,0]]]
```

```
    B = np.dot(Q_i, U)

    # Combined bias - hard Iron offset
    x_bias = -B[0,0]
    y_bias = -B[1,0]
    z_bias = -B[2,0]

    bias = np.matlib.repmat([x_bias, y_bias, z_bias], raw_data.shape[0], 1)

    print "#-------------------------------------------------------#"
    print "This script will conduct static calibration to the targeted
sensor data file."
    print "It will also do some post-processing jobs for you."
    print "Most importantly, it will give you the combined bias and A_inv
matrix."
    print "They will be used in the dynamic calibration."
    print "#-------------------------------------------------------#"
    print "Combined Bias:\n%s" %(bias[0])

    # Find the correction matrix A_inv
    J = Ellipsoid[9,0]
    A_inv = (scla.sqrtm(Q)) * mag_raw/(np.sqrt(np.dot(np.dot(B.transpose(),
Q), B) - J))
    print"\nA_inv:\n%s\n%s\n%s" %(A_inv[0], A_inv[1], A_inv[2])

    # Finally calculate and save the calibrated data
    cal_data = (np.dot(A_inv,(raw_data - bias).transpose())).transpose()
    np.savetxt("../../data/%s_cal.txt" % file_name, cal_data,'%.5e')

    # Store data points to plot and invoke plotting function
    x_raw = raw_data[:, 0]
    y_raw = raw_data[:, 1]
    z_raw = raw_data[:, 2]

    x_cal = cal_data[:, 0]
    y_cal = cal_data[:, 1]
    z_cal = cal_data[:, 2]

    # Analyze calibrated data
    static_analysis(x_raw, y_raw, z_raw, x_cal, y_cal, z_cal, A_inv,
x_bias, y_bias, z_bias, file_name, mag_raw)

"""
 Name:      main
 Purpose:   Initialize command line arguments, invoke static calibration
function and plot calibrated and uncalibrated
            data
 Param:     None
"""
def main():
    parser = argparse.ArgumentParser(description='Python script for static
calibration of magnetometer data')
    parser.add_argument('sample_path', type=str, help='Provide data file to
process')

    args = parser.parse_args()
    sample_path = args.sample_path
    static_cal(sample_path)

if __name__ == '__main__':
    main()
```

Appendix E.3 Stickman_animation.py

```
"""
Draw a stickman and compass to animate sensor data

Date Modified   Modified by     Changes
March 5, 2017   Vipul Vishnoi   Initial commit, add uncal and cal dynamic
reading txt file output;
March 10, 2017  Bernie Wu       Add initial stickman animation
March 13, 2017  Zonghao Li      add the 3d cal reading scatter plot, add
the eCompass and side-view stickman
March 20, 2017  Zonghao Li      Mapping the magnetometer data to stickman
thigh and shin motion
March 31, 2017  Zonghao Li      Modified txt file output function

"""

import pygame
import numpy as np
import serial
import sys
import time
import multiprocessing
import os
import math
import shutil
from pyqtgraph.Qt import QtCore, QtGui # Require install pyqtgraph
import pyqtgraph.opengl as gl # Require install pyopengl
pygame.init()
# initialize colours
black = (0, 0, 0)
white = (255, 255, 255)
red = (255, 0, 0)
blue = (0, 0, 255)
right = 375
thickness = 10
midx = 650
midy = 300


"""
 Name:      Serial_data_get
 Purpose:   To get serial data from port and modify message before pushing
to queue
 Param:     q, q2 - Queue handle
             sample_name - File name to store mag data
"""
def serial_data_get(q, q2, sample_name):
    ser = serial.Serial(sys.argv[1], int(sys.argv[2]))
    data = []

    # Output uncal and cal data to txt files for IMU1
    fd = open("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data.txt"), "w") # IMU 1 uncal
    os.chmod("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data.txt"), 0777)
    fd2 = open("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data_cal.txt"), "w") # IMU 1 cal
    os.chmod("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data_cal.txt"), 0777)
```

```
    # Output uncal and cal data to txt files for IMU2
    fd3 = open("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data2.txt"), "w") # IMU 2 uncal
    os.chmod("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data2.txt"), 0777)
    fd4 = open("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data2_cal.txt"), "w") # IMU 2 cal
    os.chmod("../../data/%s_dynamic/%s" % (sample_name, sample_name +
"_dynamic_data2_cal.txt"), 0777)

    # Import calibration matrix for each IMU
    bias_a_inv = np.loadtxt("../../data/%s/%s_data_Bias_&_A_inv.txt" %
(sample_name, sample_name))
    bias_a_inv2 = np.loadtxt("../../data/%s/%s_data2_Bias_&_A_inv.txt" %
(sample_name, sample_name))
    gain = 0.667
    x_bias = bias_a_inv[0,0]
    y_bias = bias_a_inv[0,1]
    z_bias = bias_a_inv[0,2]
    bias = [x_bias,y_bias,z_bias] # Hard iron offset of IMU 1
    A_1 = [[bias_a_inv[1,0], bias_a_inv[1,1], bias_a_inv[1,2]], # A-1
matrix for IMU 1
           [bias_a_inv[2,0], bias_a_inv[2,1], bias_a_inv[2,2]],
           [bias_a_inv[3,0], bias_a_inv[3,1], bias_a_inv[3,2]]]

    x_bias2 = bias_a_inv2[0,0]
    y_bias2 = bias_a_inv2[0,1]
    z_bias2 = bias_a_inv2[0,2]
    bias2 = [x_bias2,y_bias2,z_bias2] # Hard Iron offset of IMU 2
    A_12 = [[bias_a_inv2[1,0], bias_a_inv2[1,1], bias_a_inv2[1,2]], # A-1
matrix for IMU 2
            [bias_a_inv2[2,0], bias_a_inv2[2,1], bias_a_inv2[2,2]],
            [bias_a_inv2[3,0], bias_a_inv2[3,1], bias_a_inv2[3,2]]]

    while True:
        # Read Serial data
        try:
            c = ser.read(1)
            data.append(c)
            if c == "\n":
                line = ''.join(data)
                data = []
                try:
                    line = line.rstrip('\n').rstrip('\r') # unit of
readings is in mG
                    ID = int(line.split(' ')[0]) # Support multiple imu;
associate them different ID
                    x = float(line.split(' ')[1]) # x sensor reading
                    y = float(line.split(' ')[2]) # y sensor reading
                    z = float(line.split(' ')[3]) # z sensor reading
                    mag_xyz = [ID, x, y, z]
                    np.set_printoptions(suppress=True) # turn off
scientific notation

                    if (ID == 1):
                        # if sensor 1 reads data
                        fd.write("%.5f %.5f %.5f\n" % (mag_xyz[1],
mag_xyz[2], mag_xyz[3]))
                        mag_xyz_1 = [x, y, z] # mG
                        # Calibrate readings from IMU 1
```

```
                        mag_xyz_cal_1 =
np.around((np.array(np.dot((np.subtract((np.array(mag_xyz_1)*gain),bias)),A
_1))/gain), decimals=3) # set decimal number to 5, they are with no unit
                        fd2.write("%.5f %.5f %.5f\n" % (mag_xyz_cal_1[0],
mag_xyz_cal_1[1], mag_xyz_cal_1[2])) # Write Calibrated data to txt file
                    elif (ID == 2):
                        # if sensor 2 reads data
                        fd3.write("%.5f %.5f %.5f\n" % (mag_xyz[1],
mag_xyz[2], mag_xyz[3]))
                        mag_xyz_2 = [x, y, z] # mG
                        # Calibrate readings from IMU 2
                        mag_xyz_cal_2 =
np.around((np.array(np.dot((np.subtract((np.array(mag_xyz_2)*gain),bias2)),
A_12))/gain), decimals=3) # set decimal number to 5, they are with no unit
                        fd4.write("%.5f %.5f %.5f\n" % (mag_xyz_cal_2[0],
mag_xyz_cal_2[1], mag_xyz_cal_2[2])) # Write Calibrated data to txt file
                    else:
                        print "Invalid IMU ID"
                        raise

                    # Put sensor data to queue
                    mag_cal = (mag_xyz_cal_1, mag_xyz_cal_2)
                    q.put(mag_cal) # for animation
                    q2.put(mag_cal) # for scatter plot
                    print "Queuing: "  + str(mag_cal[0]) + str(mag_cal[1])
                    time.sleep(0.01)

                # Exception handler
                except KeyboardInterrupt:
                    raise

                except:
                    print "Invalid data"
                    pass

        except serial.serialutil.SerialException:
            print "Serial read fail, trying again"
            time.sleep(0.2)
            pass

        except KeyboardInterrupt:
            fd.close()
            fd2.close()
            fd3.close()
            fd4.close()
            sys.exit(1)


"""
 Name:      generate stickman and compass objects
 Purpose:   To animate/visulize real time data from queue
 Param:     q - Queue handle
"""
def serial_data_annimate(q):
    size = [midx * 2, midy * 2] # Define window size of the animation
    screen = pygame.display.set_mode(size) # Acquire screen size from
computer
    pygame.display.set_caption("Stickman_version")
    done = False
    clock = pygame.time.Clock()
    while not done:
```

```
        if not q.empty():
            data = q.get() # Get calibrated data from queue
            x1 = data[0][0] # x cal IMU1
            y1 = data[0][1] # y cal IMU1
            z1 = data[0][2] # z cal IMU1

            x2 = data[1][0] # x cal IMU2
            y2 = data[1][1] # y cal IMU2
            z2 = data[1][2] # z cal IMU2

            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    done = True

            # Defined compass and stickman objects
            screen.fill(white)
            pygame.draw.circle(screen, (0,0,255), (midx/2, midy), 300, 20)
            draw_compass_figure(screen, x1, y1, z1) # Compass
            draw_stick_figure(screen, x1, y1, z1, x2, y2, z2) # Stickman

            # Label N, S, W, E on compass
            myfont = pygame.font.SysFont("monospace", 30) # Font style
            label = myfont.render("W", 1, (255,0,0))
            screen.blit(label, (50, midy-10))
            label = myfont.render("E", 1, (255,0,0))
            screen.blit(label, (580, midy-10))
            label = myfont.render("S", 1, (255,0,0))
            screen.blit(label, (midx/2-10, 550))
            label = myfont.render("N", 1, (255,0,0))
            screen.blit(label, (midx/2-10, 30))
            pygame.draw.line(screen, blue, [700, 0], [700, 600], 5)
            pygame.display.flip()
            clock.tick(60)

    pygame.quit()


"""
 Name:      generate compass animation
 Purpose:   To animate/visulize real time data from queue
 Param:     q - Queue handle
"""
def draw_compass_figure(screen, x1, y1, z1): # Compass
    norm_x1 = 200*x1/np.sqrt(x1**2+y1**2)
    norm_y1 = 200*y1/np.sqrt(x1**2+y1**2)
    pygame.draw.circle(screen, black, [midx/2, midy], 10) # Draw the circle
    pygame.draw.line(screen, red, [midx/2, midy], [midx/2 + norm_x1, midy -
norm_y1], thickness) # Draw the niddle


"""
 Name:      generate stickman animation
 Purpose:   To animate/visulize real time data from queue
 Param:     q - Queue handle
"""
def draw_stick_figure(screen, x1, y1, z1, x2, y2, z2): # Stickman
    # Head (screen, colour, [horizontal, vertical], dia)
    pygame.draw.circle(screen, black, [midx+right, 120], 40)
    # Center Line
    pygame.draw.line(screen, black, [midx+right, 120], [midx+right, 320],
thickness)
```

```
    # Left Arm + forearm
    pygame.draw.line(screen, black, [midx+right, 220], [midx+right-100,
220+100], thickness)

    # Left Thigh + Shin
    norm_x1 = 100*x1/np.sqrt(x1**2+y1**2)
    norm_y1 = 100*y1/np.sqrt(x1**2+y1**2)
    norm_x2 = 100*x2/np.sqrt(x2**2+y2**2)
    norm_y2 = 100*y2/np.sqrt(x2**2+y2**2)
    pygame.draw.line(screen, red, [midx+right, 320], [midx+right + norm_x1,
320 + norm_y1], thickness)
    pygame.draw.line(screen, red, [midx+right + norm_x1, 320 + norm_y1],
[midx+right + norm_x1 + norm_x2, 320+norm_y1 + norm_y2], thickness)


"""
 Name:      serial_data_qtplot
 Purpose:   To plot real time data from queue.
 Param:     q2 - Queue handle
"""
def serial_data_qtplot(q2):
    kelowna_mag_raw = 367.5
    gain = 0.667
    app = QtGui.QApplication([])
    w = gl.GLViewWidget()
    w.opts['distance'] = 3000 # This is the initial zoom of the plot
    w.show()
    w.setWindowTitle('Sample magnetometer plots')

    # Define grids properties
    g = gl.GLGridItem()
    g.scale(500, 500, 500)
    w.addItem(g)

    # Define axes properties
    ax = gl.GLAxisItem()
    ax.setSize(1000, 1000, 1000)
    w.addItem(ax)

    # Draw reference sphere
    md = gl.MeshData.sphere(rows=20, cols=20, radius=kelowna_mag_raw/gain)
    m4 = gl.GLMeshItem(meshdata=md, smooth=False, drawFaces=False,
drawEdges=True, edgeColor=(1,1,1,0.2))
    m4.translate(0,10,0)
    w.addItem(m4)

    # Maximum depth of queue. Increase this number to plot more data
    max_index = 10000
    index = 0
    p = np.zeros((max_index, 3))

    # Draw scatter plot; White (1,1,1), Red(255,0,0), Blue(0,0,255),
Green(0,128,0)
    sp = gl.GLScatterPlotItem(pos=p, color=(0, 0, 255, 0.6), size=50.0,
pxMode=False)
    w.addItem(sp)
    wait = 0
    while True:
        if not q2.empty():
            wait = 0
```

```
            pos = q2.get()
            p[index] = (pos[0][0], pos[0][1], pos[0][2]) # the calibrated
readings from IMU1
            index += 1
            if index > max_index - 1:
                index = 0
                print "index reset"
            sp.setData(pos=p)
            app.processEvents()
        # If queue is empty,wait for data in queue. Exit if nothing
        else:
            if (wait > 20):
                break
            else:
                time.sleep(0.5)
                wait += 1
    print "No more data to plot, Exiting"
    sys.exit(app.exec_())


"""
 Name:      main
 Purpose:   Initialize threads and wait for valid data to appear,
            then start 2nd process to start plotting real time data in
            queue
 Param:     None
"""
def main():
    print "\n"
    print "#-------------------------------------------------------#"
    print "This script should be executed after the finish of static
calibration."
    print "It will show you real-time uncalibrated and calibrated
plotting."
    print "The combined bias and the A-1 matrix will be used to calibrate
the dynamic magnetometer readings."
    print "Press ctrl + c to stop and save the readings.\nPress ctrl + z to
exit the program."
    print "The file name you enter should be identical to the one you named
for the static calibration."
    print "#-------------------------------------------------------#"
    sample_name = raw_input('Enter file name to store data
("name_dynamic"):\n')
    if not os.path.exists("../../data/%s_dynamic" % sample_name):
        os.makedirs("../../data/%s_dynamic" % sample_name)
        os.chmod("../../data/%s_dynamic" % sample_name, 0777)

    q = multiprocessing.Queue()
    q2 = multiprocessing.Queue()
    p1 = multiprocessing.Process(target=serial_data_get, args=(q, q2,
sample_name))
    p2 = multiprocessing.Process(target=serial_data_annimate, args=(q,))
    p3 = multiprocessing.Process(target=serial_data_qtplot, args=(q2,))
    p1.start()
    while q.empty():
        print "Queue is empty"
        time.sleep(1)
    p2.start()
    p3.start()
if __name__ == '__main__':
    main()
```