Progress Report for GPS/IMU Project
Final Report

Abhijit
Vishal
Ranjith
Professor Elarabi

September 15, 2015

1 Introduction

Commonly navigation uses GPS signals to determine location. In addition, GPS data can be collected over time to determine path taken. In spite of the ubiquity of GPS signals on the earth's surface, there are certain locations where GPS signals are not available, such as inside a buildings or tunnels. Therefore, to determine accurate positioning in areas where GPS signals are unavailable, an Intertial Measurement Unit (IMU) can be used in conjunction with GPS data. The IMU is an electronic device that can measure velocity, orientation, gravitational forces and acceleration as well as magnetic fields with the help of different MEMS sensors (Microelectromechanical systems, a term referring to micro-scale devices integrated with microelectronics). Since an iPhone contains an accelerometer, gyroscope, and magnetometer, it is used as IMU device in this project. The sensors employed in iPhone are manufactured using MEMS technology.

The project deals with the integration of GPS data with the information collected from iPhone MEMS sensors. It is possible to use data collected from the iPhone sensors to estimate the missing GPS signals. The embedded accelerometers, gyroscopes, and magnetometers are used to find the translational and angular acceleration, which can be integrated to determine the translational and angular velocities, which can then be subsequently integrated to gel angular orientation and position from an initial known point.

The project proposes to estimate the missing GPS signal by sensor integration of iPhone data. This research work would enhance the GPS navigation to determine exact positions even in case of signal failures. During navigation, when the GPS as well as IMU data is available, the position error can be minimized by using either a Kalman or complementary filter. The GPS is used to correct the IMU data. When signal is lost, the IMU can be used to track the position of the device until GPS signal is regained. During the period when GPS signals are unavailable, the IMU is used for path tracking.

2 Problem Statement

This research project aims at increasing accuracy and efficiency of the GPS navigation system in hard-to-reach areas where GPS signals may be weaker by integrating Inertial Measurement Unit based on smartphone sensors. The objectives are as follows:

Orientation detection
: Design of an algorithm to detect smartphone orientation considering gyroscope and accelerometer measurements

Server implementation
: Implementation of algorithm on server to facilitate remote access to algorithm

3 Background

This section presents background information on the MEMS sensors.

3.1 Accelerometers

An accelerometer is a common MEMS device. In the IMU, it is used to determine translational acceleration. In addition, it can be used to determine the yaw and pitch as well, to correct the gyroscope data, which tends to drift. Note that this acceleration needs to be corrected for the gravitational influence. The iPhone employs a 3-axis accelerometer. The accelerometer detects a force acting in the opposite direction to the acceleration vector as determined by the net acceleration. The raw data from the smartphone can be converted to readable data through calibration.

3.2 Gyroscope

A gyroscope measures the rate of angular velocity. In the IMU, the gyroscope is used to determine the orientation. Since the gyroscope has a high rate of error, its data drifts over time. Therefore, the gyroscope's data needs to be corrected with the accelerometer. Note that this can be done with either a Kalman filter or a complementary filter. Using the Kalman filter requires knowledge of the data specifications of the MEMS sensors to estimate the sensor errors.

3.3 Magnetometer

The magnetometer is a sensor that detects the strength of magnetic fields. It is commonly used to determine heading by using the Earth's magnetic field. As such, the magnetometer can be used as a digital compass to get NSEW headings. The magnetometer present in the iPhone is an accurate device that functions properly only when the device is is a planar position, i.e. there should only be rotation about the local z-axis in the phone.

4 Project Progress

We have followed project implementation according to the flowchart given in Figure 1.

A brief description of each step is given:

Linux server A Linux server is used for remote access of the algorithm. We have installed MATLAB on the server and have tested its access.

Project background The mathematical models for the algorithmic back-end were developed. The capabilities of the accelerometer, gyroscope, and magnetometer were studied an modeled. The mathematical models we have developed rely on some prior work on orientation detection and accelerometer integration to reduce gyroscope errors.

Sensor fusion The developed algorithms were used to combine the calibrated data from the sensors to determine translation and angular position, the former used for motion tracking (dead reckoning) during lost GPS signals and the latter used to determine orientation in order to correct the local acceleration values.

User interface and kalman filter A MATLAB software platform is created to access datasets and analyze them to create an orientation and acceleration map. We have considered kalman filtering. However, with the lack of sensor specific information, such as error values, it is difficult to implement the filter.

System testing We have tested the system with some sample data obtained from on-line sources. For further testing, we need more data from a local source in order to verify the results.

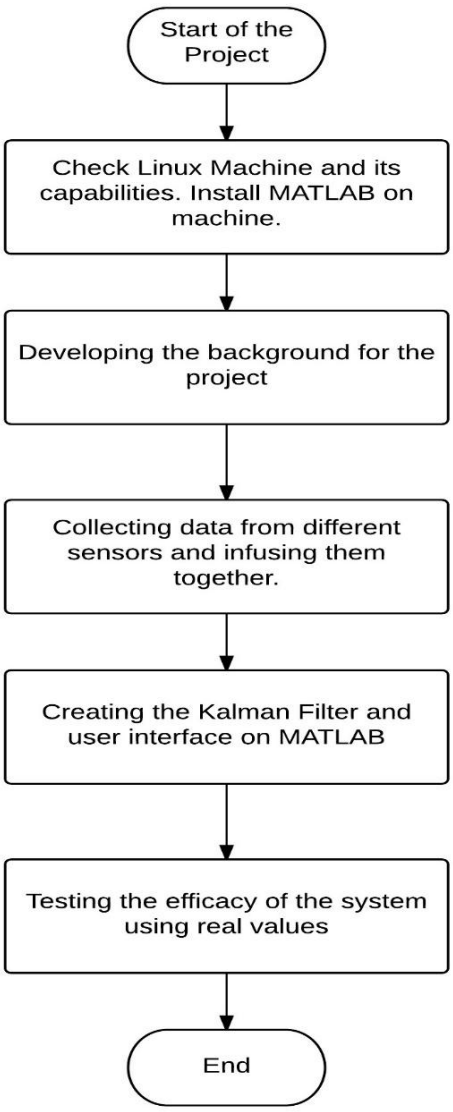

Figure 1: Project Schedule

5 Current Status

- The Linux server is built with a desktop installation of Ubuntu 14.0 LTS with remote access capabilities. A request is pending to obtain a valid license for MATLAB. Until as such time, we have installed Octave, an open source alternative to MATLAB that is compatible with most MATLAB commands.
- We have developed the mathematical models for the sensor fusion for position and orientation detection. We are still studying the Kalman Filter implementation.
- We have obtained data from an Android device (similar sensor functionalities) with an Androsensor app, which can log and send data wirelessly to a computer using UDP protocol in CSV format. However, not all data is obtained and the recording fidelity is questionable. As such, we need an iPhone or some sample data to test the system.
- The sensor fusion of accelerometer, magnetometer, and gyroscope has been completed. The calculation of orientation and displacement of iPhone has been completed using an algorithm implemented in MATLAB.

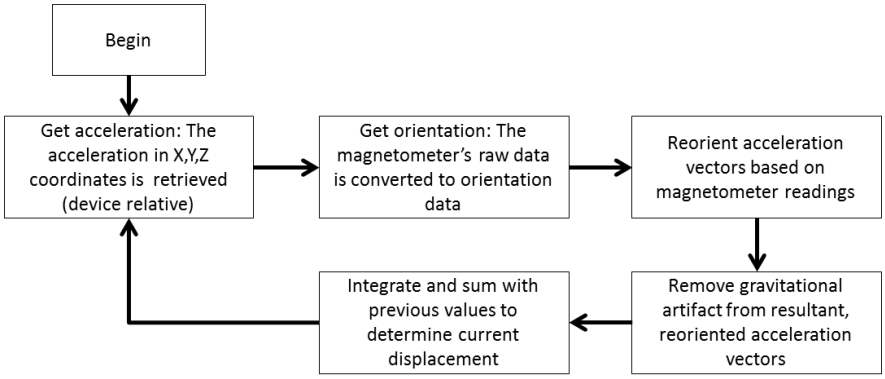

Figure 2: Algorithm - Orientation and Displacement detection process

6 Orientation and Displacement Detection with MATLAB

The implementation of GPS/IMU integration was undertaken by converting the iPhone to an IMU device. This is accomplished by integrating together data from the iPhone's accelerometer, gyroscope, and magnetometer. The implementation method is described as follows.

The iPhones accelerometer gives the device's acceleration in X, Y, and Z coordinates. This acceleration can be integrated over time to determine the device's velocity. The velocity can be integrated to determine the device's change in position. Since the accelerometer measures gravity as acceleration, the effects of gravity must be taken into account. When taken with an initial position from the GPS coordinates, the displacement data from the accelerometer can be used to determine accurate position via dead reckoning. However, the accelerometer's raw data is given in local device coordinates; thus, if the device's orientation is changed, the axes orientations also change. So, if the device is turned on its side, the Z-axis no longer points upwards; instead, it is also rotated.

Such a problem can be fixed with a gyroscope, which gives the change in angular position. This can be integrated to yield the angular displacement and, given an initial, known orientation, the devices angular position can be calculated at all times. Once the orientation is determined, the acceleration axes can be properly reoriented to correspond to a predetermined orientation so as to maintain constancy with the data, i.e. the virtual acceleration orientation can be normalized to follow an earth-relative orientation, rather than a device-relative orientation. However, the iPhones gyroscope can give erroneous data when jolted. Errors can accumulate extremely quickly, unless a Kalman filter is implemented. Furthermore, the gyroscope is unstable and lower angular velocities do not register on the sensor. Thus, if the orientation changes slowly over time, it would be impossible to verify.

Another solution is to use only the magnetometer. Since the magnetometer's readings are based on the earth's magnetic field, the raw data is considerably more accurate. The magnetometer can be used as a digital compass to determine the device's orientation around the Z-axis. In fact, this leads to a more appropriate earth-relative orientation system using geodetic format, where North corresponds to the X-axis and East corresponds to the Y-axis. A constraint is introduced in this system while the device can turn on the Z-axis, it cannot turn on the other two axes, i.e. it must remain level at all times, as the gyroscope is not extensively used and the magnetometer stops functioning properly when the device is tilted along the X- or Y- axis.

This implementation uses the accelerometer in conjunction with the magnetometer to determine the device‘s earth-relative position. The gyroscope is not extensively used due to its relative instability compared to the accelerometer and magnetometer. A flowchart of the programmatic process is given in Figure 2.

7 Algorithm for orientation detection

The iphone data consists of readings from the three-axis accelerometer, three-axis magnetometer, and three-axis gyroscope. Although the gyroscope is not extensively used at this time, its readings are still combined into the data for future use. The magnetometer data is an accurate representation of the readings as there is negligible fluctuation in the magnetic field. However, the accelerometer data must be corrected for fluctuations that may occur due to occasional bumps or shakes. This is accomplished by taking some data

with the device facing in the positive and negative X, Y, and Z axes and correcting any drifts with calibration equations, shown in Equation 1, where$m$ represents the slope of any drift.

$$Acc_{corrected} = m \cdot Acc_{uncorrected} \tag{1}$$

To maintain Earth-relative magnitudes, it is necessary to determine any rotation of the device around the Z axis and correct the vectors based on this rotation. The rotations are determined by considering the changes in heading produced by the magnetometer. The North and East vectors (X and Y) are corrected by rotating them to correlate with device orientation. Finally, the earth-frame acceleration in the X, Y, and Z axes are determined.

7.1 Determining Earth relative X, Y, and Z

The device frame axes, given in Figure **??**, are assumed to be fixed. Since the device may rotate, the data from these axes must be corrected relative to device orientation in the world frame. So, after the data are calibrated with Equation 1, the vectors are normalized and unit vectors obtained. Since the background acceleration is always downwards towards Earth, we can assume the acceleration vector **G** to be the gravity vector. Furthermore, the magnetometer vector **B** always points North because the magnetometer measures the magnetic field relative to the North pole. As a result, based on the Right hand Rule, the cross product of **G** and **B** yields the magnitude of the vector pointing East, or the **E** vector.

$$E = uG \times uB \tag{2}$$

Note that although the magnetometer vector **B** is considered the North vector, it cannot be used for calculation. Instead, the East vector **E** and **G** crossecd yield the true normalized North vector.

$$N = uE \times uG \tag{3}$$

7.2 Roll and Pitch calculation

The roll and pitch values allow correction of the acceleration values. Note that the device must be pointing to the Earth-relative X axis, i.e. the North pole, for orientation detection to function properly. The pitch and roll are defined in Equations 4 and 5, respectively.

$$Pitch = -\arcsin uG_x \tag{4}$$

$$Roll = -\arcsin uG_y \tag{5}$$

7.3 Rotation Matrix

Finally, a rotation matrix is created to aid in orientation tracking and correcting the accelerometer readings. The rotation matrix gives all components in the North, East, and Z directions, as shown in Equation 6.

$$M = \begin{bmatrix} uNx & uNy & uNz \\ uEx & uEy & uEz \\ uGx & uGy & uGz \end{bmatrix} \tag{6}$$

The rotation angle can be derived from the rotation matrix M with the following equation:

$$\theta = \arccos(\frac{uNx + uEy + uGz - 1}{2}) \tag{7}$$

This is the net rotation angle. The components of this angle are given by Equation 8.

$$W = \frac{1}{2\sin\theta} \begin{bmatrix} uGy - uEz \\ uNz - uGz \\ uEx - uNy \end{bmatrix} \tag{8}$$

8 Earth Frame Rotation Matrix for Acceleration

8.1 Earth Frame Accelerations

The net accelerations in the body-frame coordinates are given by subtracting the gravitational acceleration from the components of the body frame acceleration. Finally, the rotation matrix and the rotation angles obtained from Equations 6 and 8 are used to determine Earth framre accelerations. This Earth Frame acceleration is given by Equation 9.

$$Acc_{Earth} = Acc_{Body}\cos(\theta) + (W \times Acc_{Body})\sin(\theta) + (W \times (W \cdot Acc_{Body}))(1 - \cos(\theta)) \tag{9}$$

8.2 Earth Frame Rotation Matrix

The Earth Frame Rotation Matrix (EFRM) is different from the Body Frame Rotation Matrix **W** presented in Equation 6. The EFRM, denoted by B_M, is the rotation of the device with respect to the Earth frame geodetic system. Note that for correct usage of the rotation matrices, it is necessary that the device's axes are initially oriented with the Earth Frame. In the presence pof GPS data, this can be corrected without initial orientation. Currently, we lack access to sample GPS data, and so cannot determine initial orientation.

The rotation matrix B_M is determined as follows:

$$B_M = \begin{bmatrix} 0 & -G_z\Delta t & G_y\Delta t \\ G_z\Delta t & 0 & -G_x\Delta t \\ -G_y\Delta t & -G_x\Delta t & 0 \end{bmatrix} \tag{10}$$

Note that a Body Frame Rotation Matrix determined from B_M can be more accurate than the matrix **W** determined from purely acceleration vectors. As such, a new rotation matrix G_M is determined as shown in Equation 11, where M_{unit} is the unit matrix (3 dimensions) and σ is the magnitude of the gyroscope's rotation vector ($\sigma = \sqrt{G_x^2 + G_y^2 + G_z^2}$).

$$G_M = M_{unit} + B_M \cdot \frac{\sin(\sigma)}{\sigma} + B_M^2 \cdot \frac{1 - \cos(\sigma)}{\sigma^2} \tag{11}$$

The projections of the accelerations in the X, Y, and Z along the normalized geodetic frame is determined with the gyroscope-based rotation matrix. Note that the vectors **E**, **N**, and **G** denote the East (x-axis), North (y-axis), and height (z-axis) in the geodetic frame. The gravitational constant is $9.8\frac{m}{s^2}$.

$$X_{acc} = G_{const} * (X_{corrected} * E_x + Y_{corrected} * E_y + Z_{corrected} * E_z); \tag{12}$$

$$Y_{acc} = G_{const} * (X_{corrected} * N_x + Y_{corrected} * N_y + Z_{corrected} * N_z); \tag{13}$$

$$Z_{acc} = G_{const} * (X_{corrected} * G_x + Y_{corrected} * G_y + Z_{corrected} * G_z); \tag{14}$$

9 Earth Frame Velocities and Displacement Determination

The Earth frame velocities are determined by integration of the acceleration vectors. A second integration yields the Earth frame displacement along each axis.

$$X_{displacement} = \iint_{\Delta t} X_{acc}; \tag{15}$$

10 Results

An excerpt of the data sample is given in Table 1.

The sample contains the individual accelerations and the magnetometer data. The gyroscope data given here is used to determine the rotation matrices. The algorithms and equations as outlined in Section 7 and in Section 8 are applied to this data. The accelerations in the North (X acis), East (Y direction) and Z axes

Table 1: Excerpt of sample data

Time	AccelX	AccelY	AccelZ	GyroX	GyroY	GyroZ	MagX	MagY	MagZ
1.700	-0.025	-0.156	-0.973	0.031	-0.042	0.004	4.117	12.434	-35.769
1.711	-0.017	-0.155	-0.985	0.019	-0.007	-0.002	4.117	12.434	-35.769
1.721	-0.018	-0.153	-0.989	-0.008	0.021	-0.016	4.529	12.572	-36.200
1.731	-0.010	-0.153	-0.994	-0.008	0.021	-0.016	4.529	12.572	-36.200
1.741	-0.011	-0.150	-0.988	-0.015	0.054	0.005	4.873	12.503	-36.128
1.751	-0.027	-0.151	-0.981	-0.008	0.047	0.010	4.873	12.503	-36.128
1.761	-0.040	-0.158	-0.979	0.014	-0.005	0.004	4.873	12.503	-36.128

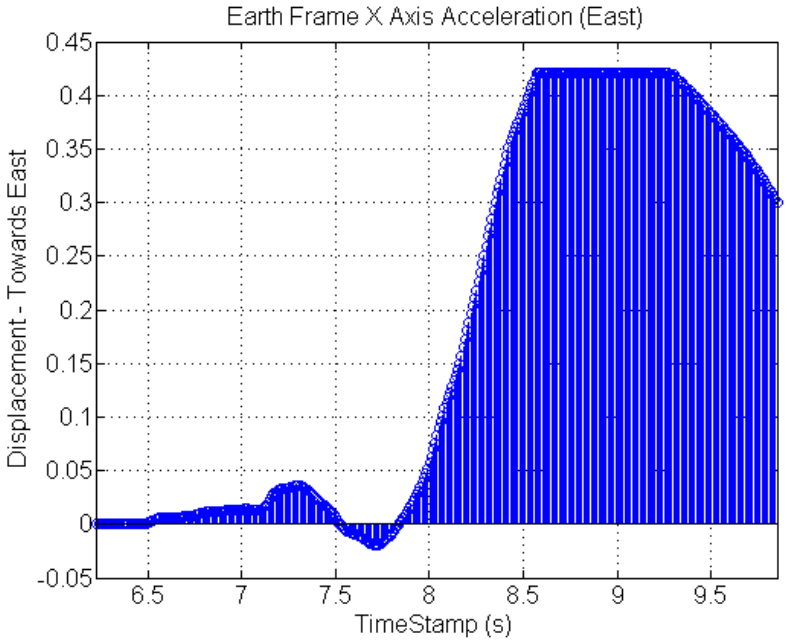

Figure 3: X Axis Net Acceleration

are determined. The resultant values are passed through a Savitzky-Golay FIR smoothing filter to remove noisy elements. The graphs of each component of the acceleration are given in Figures 3, 4, and 5.

It is possible to see the movement of the device over time in each of the graphs. In the X axis acceleration, it is clear that the device moves towards the North between 2 and 3 seconds. The data at 4 seconds shows positive and negative accelerations. This indicates a bump and a reverse acceleration. The integration of this acceleration will actually yield zero average velocity. The Savitzky-Golay filter has reduced the error at 4 seconds to a more appropriate value as well. In the Y axis acceleration, the device moves East quickly before stopping between 2 and 3 seconds. The bump in the X axis acceleration is also present in the Y axis acceleration. Finally, the Z axis acceleration indicates the device moved up from approximately 2.6 seconds to 3 seconds and subsequently moved down in small step increments between 3 and 4 seconds.

This data shows the net acceleration for the device in Earth frame coordinates. The data obtained can be compared with GPS data to correct any present drift errors. In situations where GPS signals are lost, the accelerometer, gyroscope, and magnetometer data can be used to determine the path taken by the device.

11 Discussion and Conclusions

This research presents an orientation detection and subsequent displacement detection algorithm for use in GPS/IMU integration. The orientation detection was accomplished by using an iPhone's accelerometer and magnetometer to to determine device orientation relative to a fixed Earth frame, also known as the geodetic

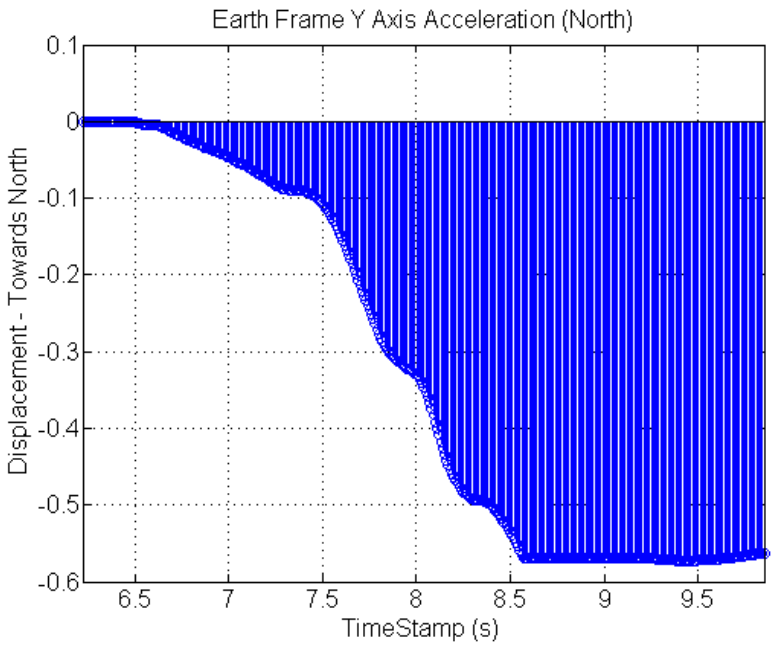

Figure 4: Y Axis Net Acceleration

frame. This orientation was used to derive a rotation matrix and a rotation quaternion. The rotation matrix can be used to determine the net acceleration in Earth Frame coordinates using Equation 9. The rotation quaternion, presented in the MATLAB code, can be used to implement the gyroscope readings and correct them with the accelerometer data-derived pitch and roll.

The results of this research show an implementation with sample data. The algorithm must be tested on more local sample data to determine efficiency. In addition, the drift of the measurements must be calculated to determine period of efficacy before the readings and subsequent dead reckoning become undependable.

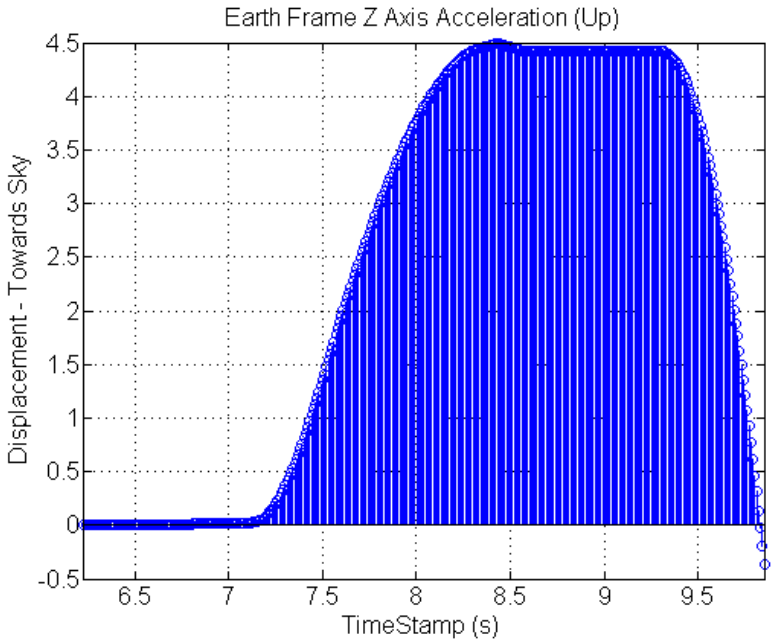

Figure 5: Z Axis Net Acceleration

12 Appendix - Selected Code

12.1 Data Correction

The following code shows the data correction using the calibration equation in Equation: 1. The acceleration data is corrected and both the acceleration and magnetometer data are normalized.

```
%%% Magnitude of acceleration and magnetometer vectors
accMagUncorrected = sqrt(accelX.^2 + accelY.^2 + accelZ.^2);
magMag = sqrt(magX.^2 + magY.^2 + magZ.^2);

%%% Vector correction, magnitude determination, normalization (acc)
accelXCorr = (xSlope*accelX) + xInter;
accelYCorr = (ySlope*accelY) + yInter;
accelZCorr = (zSlope*accelZ) + zInter;

accMagCorr = sqrt(accelXCorr.^2 + accelYCorr.^2 + accelZCorr.^2);

%%% Normalized acceleration

accXNorm = accelXCorr./accMagCorr;
accYNorm = accelYCorr./accMagCorr;
accZNorm = accelZCorr./accMagCorr;

%%% Magnetometer normalized

magXNorm = magX./magMag;
magYNorm = magY./magMag;
magZNorm = magZ./magMag;
```

12.2 North and East Determination

The East direction is determined by crossing the **G** and the **B** vectors, as given in Section 7. The North direction magnitude is determined by crossing the **E** and **G** vectors.

```
%%% East determination (accNorm cross mag) and normalization

eastX = accYNorm.*magZ - accZNorm.*magY;
eastY = accXNorm.*magZ - accZNorm.*magX;
eastZ = accXNorm.*magY - accYNorm.*magX;

eastMag = sqrt(eastX.^2 + eastY.^2 + eastZ.^2);

eastNormX = eastX./eastMag;
eastNormY = eastY./eastMag;
eastNormZ = eastZ./eastMag;

%%% North determination G cross B cross G i.e. E cross G and normalization

northX = (accXNorm.^2 + accYNorm.^2 + accZNorm.^2).*magX - ...
    (accXNorm.*magX + accYNorm.*magY + accZNorm.*magZ).*accXNorm;
northY = (accXNorm.^2 + accYNorm.^2 + accZNorm.^2).*magY - ...
    (accXNorm.*magX + accYNorm.*magY + accZNorm.*magZ).*accYNorm;
northZ = (accXNorm.^2 + accYNorm.^2 + accZNorm.^2).*magZ - ...
    (accXNorm.*magX + accYNorm.*magY + accZNorm.*magZ).*accZNorm;

northMag = sqrt(northX.^2 + northY.^2 + northZ.^2);
```

```
northNormX = northX./northMag;
northNormY = northY./northMag;
northNormZ = northZ./northMag;
```

12.3 Gyroscope Matrices

The rotation matrices are created with the following code.

```
%%% Matrix Representation of Gyroscope data

Br1a = zeros(length(timeStamp),1);
Br1b = -gyroZ.*delta;
Br1c = gyroY.*delta;

Br2a = gyroZ.*delta;
Br2b = zeros(length(timeStamp),1);
Br2c = -gyroX.*delta;

Br3a = -gyroY.*delta;
Br3b = gyroX.*delta;
Br3c = zeros(length(timeStamp),1);

%%% B squared
BsqR1a = Br1a.*Br1a + Br1b.*Br2a + Br1c.*Br3a;
BsqR1b = Br1a.*Br1b + Br1b.*Br2b + Br1c.*Br3b;
BsqR1c = Br1a.*Br1c + Br1b.*Br2c + Br1c.*Br3c;

BsqR2a = Br2a.*Br1a + Br2b.*Br2a + Br2c.*Br3a;
BsqR2b = Br2a.*Br1b + Br2b.*Br2b + Br2c.*Br3b;
BsqR2c = Br2a.*Br1c + Br2b.*Br2c + Br2c.*Br3c;

BsqR3a = Br3a.*Br1a + Br3b.*Br2a + Br3c.*Br3a;
BsqR3b = Br3a.*Br1b + Br3b.*Br2b + Br3c.*Br3b;
BsqR3c = Br3a.*Br1c + Br3b.*Br2c + Br3c.*Br3c;

%%% Gyroscope Rotation Matarix

gyroRotMatr1a = 1 + sincMagRotVector.*Br1a + cosMagRotVector.*BsqR1a;
gyroRotMatr1b = sincMagRotVector.*Br1b + cosMagRotVector.*BsqR1b;
gyroRotMatr1c = sincMagRotVector.*Br1c + cosMagRotVector.*BsqR1c;

gyroRotMatr2a = sincMagRotVector.*Br2a + cosMagRotVector.*BsqR2a;
gyroRotMatr2b = 1 + sincMagRotVector.*Br2b + cosMagRotVector.*BsqR2b;
gyroRotMatr2c = sincMagRotVector.*Br2c + cosMagRotVector.*BsqR2c;

gyroRotMatr3a = sincMagRotVector.*Br3a + cosMagRotVector.*BsqR3a;
gyroRotMatr3b = sincMagRotVector.*Br3b + cosMagRotVector.*BsqR3b;
gyroRotMatr3c = 1 + sincMagRotVector.*Br3c + cosMagRotVector.*BsqR3c;

netMagRot = sqrt(gyroX.^2 + gyroY.^2 + gyroZ.^2);
```

12.4 Body frame Projection Matrices

The body frame rotation matrices are created with the following code. This is used to determine the change in device orientation and derive the earth frame accelerations.

```
%%%Rotation matrices - East, North, and Z axis in Row 1, 2, 3

for count = 2:endPoint
efEastR1a(count) = efEastR1a(count-1)*gyroRotMatr1a(startFrame + ...
    count - 1) + efEastR1b(count-1)*gyroRotMatr2a(startFrame + ...
    count - 1) + efEastR1c(count-1)*gyroRotMatr3a(startFrame + count - 1);
efEastR1b(count) = efEastR1a(count-1)*gyroRotMatr1b(startFrame + ...
    count - 1) + efEastR1b(count-1)*gyroRotMatr2b(startFrame + ...
    count - 1) + efEastR1c(count-1)*gyroRotMatr3b(startFrame + count - 1);
efEastR1c(count) = efEastR1a(count-1)*gyroRotMatr1c(startFrame + ...
    count - 1) + efEastR1b(count-1)*gyroRotMatr2c(startFrame + ...
    count - 1) + efEastR1c(count-1)*gyroRotMatr3c(startFrame + count - 1);

efNorthR2a(count) = efNorthR2a(count-1)*gyroRotMatr1a(startFrame + ...
    count - 1) + efNorthR2b(count-1)*gyroRotMatr2a(startFrame + ...
    count - 1) + efNorthR2c(count-1)*gyroRotMatr3a(startFrame + count - 1);
efNorthR2b(count) = efNorthR2a(count-1)*gyroRotMatr1b(startFrame + ...
    count - 1) + efNorthR2b(count-1)*gyroRotMatr2b(startFrame + ...
    count - 1) + efNorthR2c(count-1)*gyroRotMatr3b(startFrame + count - 1);
efNorthR2c(count) = efNorthR2a(count-1)*gyroRotMatr1c(startFrame + ...
    count - 1) + efNorthR2b(count-1)*gyroRotMatr2c(startFrame + ...
    count - 1) + efNorthR2c(count-1)*gyroRotMatr3c(startFrame + count - 1);

efUpR3a(count) = efUpR3a(count-1)*gyroRotMatr1a(startFrame + ...
    count - 1) + efUpR3b(count-1)*gyroRotMatr2a(startFrame + ...
    count - 1) + efUpR3c(count-1)*gyroRotMatr3a(startFrame + count - 1);
efUpR3b(count) = efUpR3a(count-1)*gyroRotMatr1b(startFrame + ...
    count - 1) + efUpR3b(count-1)*gyroRotMatr2b(startFrame + ...
    count - 1) + efUpR3c(count-1)*gyroRotMatr3b(startFrame + count - 1);
efUpR3c(count) = efUpR3a(count-1)*gyroRotMatr1c(startFrame + ...
    count - 1) + efUpR3b(count-1)*gyroRotMatr2c(startFrame + ...
    count - 1) + efUpR3c(count-1)*gyroRotMatr3c(startFrame + count - 1);
end
```