Freescale Semiconductor
Application Note

Document Number: AN4248
Rev. 4.0, 11/2015

Implementing a Tilt-Compensated eCompass using Accelerometer and Magnetometer Sensors

by: Talat Ozyagcilar
Applications Engineer

1 Introduction

This technical note provides the mathematics, reference source code and guidance for engineers implementing a tilt-compensated electronic compass (eCompass).

The eCompass uses a three-axis accelerometer and three-axis magnetometer. The accelerometer measures the components of the earth's gravity and the magnetometer measures the components of earth's magnetic field (the geomagnetic field). Since both the accelerometer and magnetometer are fixed on the Printed Circuit Board (PCB), their readings change according to the orientation of the PCB.

If the PCB remains flat, then the compass heading could be computed from the arctangent of the ratio of the two horizontal magnetic field components. Since, in general, the PCB will have an arbitrary orientation, the compass heading is a function of all three accelerometer readings and all three magnetometer readings.

The tilt-compensated eCompass algorithm actually calculates all three angles (pitch, roll, and yaw or compass heading) that define the PCB orientation. The eCompass algorithms can therefore also be used to create a 3D Pointer with the pointing direction defined by the yaw and pitch angles.

Contents

1 Introduction 1
1.1 Related Information 2
1.2 Key Words 2
1.3 Summary 2
2 Coordinate System and Package Alignment 3
3 Accelerometer and Magnetometer Outputs as a Function of Phone Orientation 5
4 Tilt-Compensation Algorithm 7
5 Estimation of the Hard-Iron Offset V 9
6 Visualization Using Experimental Data 10
7 Software Implementation 15
7.1 eCompass C# Source Code 15
7.2 Modulo Arithmetic Low Pass Filter for Angles C# Source Code 16
7.3 Sine and Cosine Calculation C# Source Code . . . 17
7.4 ATAN2 Calculation C# Source Code 19
7.5 ATAN Calculation C# Source Code 19
7.6 Integer Division C# Source Code 20

© 2013, 2015 Freescale Semiconductor, Inc. All rights reserved.

The accuracy of an eCompass is highly dependent on the calculation and subtraction in software of stray magnetic fields both within, and in the vicinity of, the magnetometer on the PCB. By convention, these fields are divided into those that are fixed (termed Hard-Iron effects) and those that are induced by the geomagnetic field (termed Soft-Iron effects). Any zero field offset in the magnetometer is normally included with the PCB's Hard-Iron effects and is calibrated at the same time.

This document describes a simple three-element model to compensate for Hard-Iron effects. This three-element model should suffice for many situations. Please contact your Freescale sales representative for details of a full 10-element model which compensates for both Hard and Soft-Iron effects.

The C# language source code listed within this document contains cross-references to the equations used. These listings contain all the code needed to return the yaw, pitch and roll angles from the magnetometer and accelerometer sensor readings.

For convenience, the remainder of this document assumes that the eCompass will be implemented within a mobile phone.

1.1 Related Information

C source code and additional documentation are available for download at www.freescale.com/sensorfusion.

1.2 Key Words

Accelerometer, Magnetometer, Tilt angles, eCompass, 3D Pointer, Tilt Compensation, Tilt Correction, Hard Iron, Soft Iron, Geomagnetism

1.3 Summary

1. A tilt-compensated electronic compass (eCompass) is implemented using the combination of a three-axis accelerometer and a three-axis magnetometer.
2. The accelerometer readings provide pitch and roll angle information which is used to correct the magnetometer data. This allows for accurate calculation of the yaw or compass heading when the eCompass is not held flat.
3. The pitch and roll angles are computed on the assumption that the accelerometer readings result entirely from the eCompass orientation in the earth's gravitational field. The tilt-compensated eCompass will not operate under freefall or low-g conditions at one extreme nor high-g accelerations at the other.
4. A 3D Pointer can be implemented using the yaw (compass heading) and pitch angles from the eCompass algorithms.
5. The magnetometer readings must be corrected for Hard-Iron and Soft-Iron effects.
6. A simple three-parameter Hard-Iron correction algorithm is described. Please contact your Freescale sales representative for details of Freescale's complete 10-parameter Hard and Soft-Iron correction algorithms.
7. Reference C# code is provided at the end of this document for the full tilt-compensated eCompass with Hard-Iron compensation.
8. Demonstration eCompass platforms are available that show Freescale's latest sensors. Please contact your Freescale sales representative for details.

2 Coordinate System and Package Alignment

This application note uses the industry standard "NED" (North, East, Down) coordinate system to label axes on the mobile phone. The *x-axis* of the phone is the eCompass pointing direction, the *y-axis* points to the right and the *z-axis* points downward. (see Figure 1).

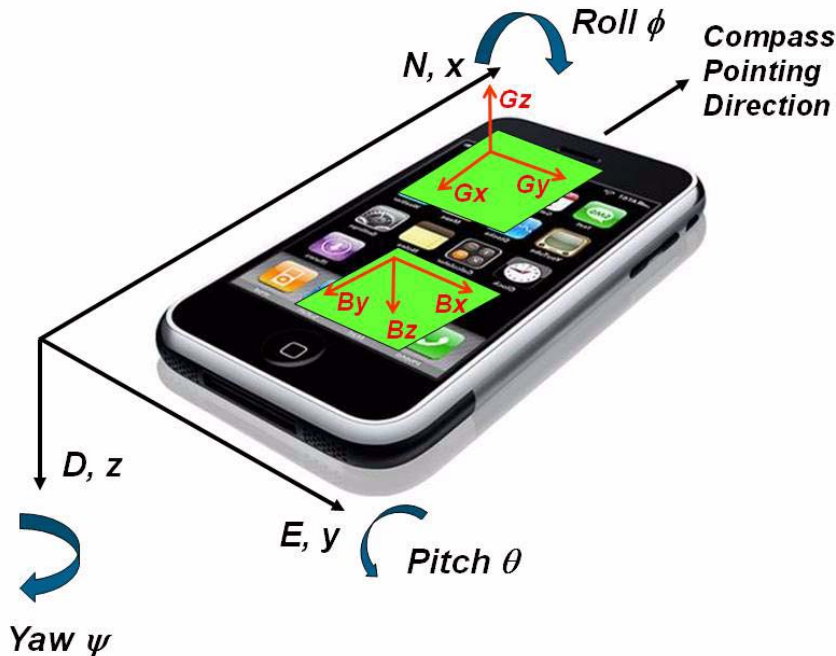


Figure 1. Coordinate System

A positive yaw angle ψ is defined to be a clockwise rotation about the positive *z-axis*. Similarly, a positive pitch angle θ and positive roll angle ϕ are defined as clockwise rotations about the positive *y*- and positive *x-axes* respectively.

It is crucial that the accelerometer and magnetometer outputs are aligned with the phone coordinate system. Different PCB layouts may have different orientations of the accelerometer and magnetometer packages and even the same PCB may be mounted in different orientations within the final product.

For example, in Figure 1, the accelerometer *y-axis* output G_y, is correctly aligned, but the *x-axis* G_x and *z-axis* G_z signals are inverted in sign. Also in Figure 1, the magnetometer output B_z is correct, but the *y-axis* signal should be set to B_x and the *x-axis* signal should be set to $-B_y$.

Once the package rotations and reflections are applied in software, a final check should be made while watching the raw accelerometer and magnetometer data from the PCB:

1. Place the PCB flat on the table. The *z-axis* accelerometer should read +1g and the *x* and *y* axes negligible values. Invert the PCB so that the *z-axis* points upwards and verify that the *z-axis* accelerometer now indicates -1g. Repeat with the *y-axis* pointing downwards and then upwards to check that the *y-axis* reports 1g and then reports -1g. Repeat once more with the *x-axis* pointing downwards and then upwards to check that the *x-axis* reports 1g and then -1g.
2. The horizontal component of the geomagnetic field always points to the magnetic north pole. In the northern hemisphere, the vertical component also points downward with the precise angle being dependent on location. When the PCB *x-axis* is pointed northward and downward, it should be possible to find a maximum value of the measured *x* component of the magnetic field. It should also be possible to find a minimum value when the PCB is aligned in the reverse direction. Repeat the measurements with the PCB *y-* and *z-axes* aligned first with, and then against, the geomagnetic field which should result in maximum and minimum readings in the *y-* and then *z-axes*.

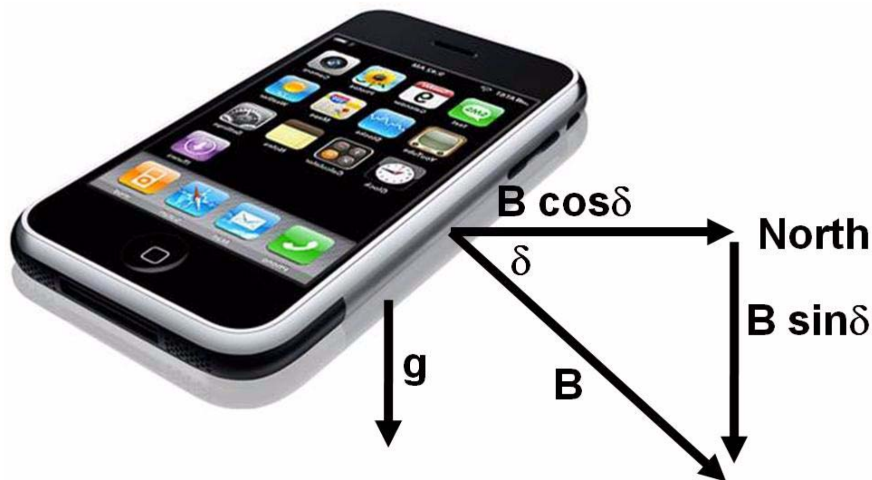


Figure 2. Gravitational and Magnetic Field Vectors

3 Accelerometer and Magnetometer Outputs as a Function of Phone Orientation

Any orientation of the phone can be modeled as resulting from rotations in yaw, pitch and roll applied to a starting position with the phone flat and pointing northwards. The accelerometer, $\boldsymbol{G}_r$, and magnetometer, $\boldsymbol{B}_r$, readings in this starting reference position are (see Figure 2):

$$\boldsymbol{G}_r = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad \text{Eqn. 1}$$

$$\boldsymbol{B}_r = B \begin{pmatrix} \cos\delta \\ 0 \\ \sin\delta \end{pmatrix} \quad \text{Eqn. 2}$$

The acceleration due to gravity is $g = 9.81\ \mathrm{ms}^{-2}$. B is the geomagnetic field strength which varies over the earth's surface from a minimum of 22 μT over South America to a maximum of 67 μT south of Australia. δ is the angle of inclination of the geomagnetic field measured downwards from horizontal and varies over the earth's surface from -90° at the south magnetic pole, through zero near the equator to +90° at the north magnetic pole. Detailed geomagnetic field maps are available from the *World Data Center for Geomagnetism* at http://wdc.kugi.kyoto-u.ac.jp/igrf/.

There is no requirement to know the details of the geomagnetic field strength nor inclination angle in order for the eCompass software to function since these cancel in the angle calculations (see Equations 20, 21 and 22).

The phone accelerometer, $\boldsymbol{G}_p$, and magnetometer, $\boldsymbol{B}_p$, readings measured after the three rotations $\boldsymbol{R}_z(\psi)$ then $\boldsymbol{R}_y(\theta)$ and finally $\boldsymbol{R}_x(\phi)$ are described by the equations:

$$\boldsymbol{G}_p = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)\boldsymbol{G}_r = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad \text{Eqn. 3}$$

$$\boldsymbol{B}_p = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)\boldsymbol{B}_r = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)B\begin{pmatrix} \cos\delta \\ 0 \\ \sin\delta \end{pmatrix} \quad \text{Eqn. 4}$$

The three rotation matrices referred to in Equations 3 and 4 are:

$$\boldsymbol{R}_x(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \quad \text{Eqn. 5}$$

$$\boldsymbol{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \quad \textbf{Eqn. 6}$$

$$\boldsymbol{R}_z(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \textbf{Eqn. 7}$$

Equation 3 assumes that the phone is not undergoing any linear acceleration and that the accelerometer signal $\boldsymbol{G}_p$ is a function of gravity and the phone orientation only. A tilt-compensated eCompass will give erroneous readings if it is subjected to any linear acceleration.

Equation 4 ignores any stray magnetic fields from Hard and Soft-Iron effects. The standard way of modeling the Hard-Iron effect is as an additive magnetic vector, $\boldsymbol{V}$, which rotates with the phone PCB and is therefore independent of phone orientation. Since any magnetometer sensor zero flux offset is also independent of phone orientation, it simply adds to the PCB Hard-Iron component and is calibrated and removed at the same time.

Equation 4 then becomes:

$$\boldsymbol{B}_p = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)B\begin{pmatrix} \cos\delta \\ 0 \\ \sin\delta \end{pmatrix} + \boldsymbol{V} = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)B\begin{pmatrix} \cos\delta \\ 0 \\ \sin\delta \end{pmatrix} + \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} \quad \textbf{Eqn. 8}$$

where V_x, V_y, and V_z, are the components of the Hard-Iron vector. Equation 8 does not model Soft-Iron effects. Please contact your Freescale sales representative for details of Freescale's full Hard-Iron and Soft-Iron calibration model and calibration source code.

4 Tilt-Compensation Algorithm

The tilt-compensated eCompass algorithm first calculates the roll and pitch angles ϕ and θ from the accelerometer reading by pre-multiplying Equation 3 by the inverse roll and pitch rotation matrices giving:

$$\boldsymbol{R}_y(-\theta)\boldsymbol{R}_x(-\phi)\boldsymbol{G}_p = \boldsymbol{R}_y(-\theta)\boldsymbol{R}_x(-\phi)\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \boldsymbol{R}_z(\psi)\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad \text{Eqn. 9}$$

where the vector $\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix}$ contains the three components of gravity measured by the accelerometer.

Expanding Equation 9 gives:

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad \text{Eqn. 10}$$

$$\Rightarrow \begin{pmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} \quad \text{Eqn. 11}$$

The y component of Equation 11 defines the roll angle ϕ as:

$$G_{py}\cos\phi - G_{pz}\sin\phi = 0 \quad \text{Eqn. 12}$$

$$\Rightarrow \tan(\phi) = \left(\frac{G_{py}}{G_{pz}}\right) \quad \text{Eqn. 13}$$

The x component of Equation 11 gives the pitch angle θ as:

$$G_{px}\cos\theta + G_{py}\sin\theta\sin\phi + G_{pz}\sin\theta\cos\phi = 0 \quad \text{Eqn. 14}$$

$$\Rightarrow \tan(\theta) = \left(\frac{-G_{px}}{G_{py}\sin\phi + G_{pz}\cos\phi}\right) \quad \text{Eqn. 15}$$

With the angles ϕ and θ known from the accelerometer, the magnetometer reading can be de-rotated to correct for the phone orientation using Equation 8:

$$\boldsymbol{R}_z(\psi)\begin{pmatrix} B\cos\delta \\ 0 \\ B\sin\delta \end{pmatrix} = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} B\cos\delta \\ 0 \\ B\sin\delta \end{pmatrix} = \boldsymbol{R}_y(-\theta)\boldsymbol{R}_x(-\phi)(\boldsymbol{B}_p - \boldsymbol{V}) \quad \text{Eqn. 16}$$

$$\Rightarrow \begin{pmatrix} \cos\psi B\cos\delta \\ -\sin\psi B\cos\delta \\ B\sin\delta \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} B_{px}-V_x \\ B_{py}-V_y \\ B_{pz}-V_z \end{pmatrix} \qquad \textbf{Eqn. 17}$$

$$= \begin{pmatrix} \cos\theta & \sin\theta\sin\phi & \sin\theta\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} B_{px}-V_x \\ B_{py}-V_y \\ B_{pz}-V_z \end{pmatrix} \qquad \textbf{Eqn. 18}$$

$$= \begin{pmatrix} (B_{px}-V_x)\cos\theta + (B_{py}-V_y)\sin\theta\sin\phi + (B_{pz}-V_z)\sin\theta\cos\phi \\ (B_{py}-V_y)\cos\phi - (B_{pz}-V_z)\sin\phi \\ -(B_{px}-V_x)\sin\theta + (B_{py}-V_y)\cos\theta\sin\phi + (B_{pz}-V_z)\cos\theta\cos\phi \end{pmatrix} = \begin{pmatrix} B_{fx} \\ B_{fy} \\ B_{fz} \end{pmatrix} \qquad \textbf{Eqn. 19}$$

The vector $\begin{pmatrix} B_{fx} \\ B_{fy} \\ B_{fz} \end{pmatrix}$ represent the components of the magnetometer sensor after correcting for the Hard-Iron offset and after de-rotating to the flat plane where $\theta = \phi = 0$.

The x and y components of Equation 19 give:

$$\cos\psi B\cos\delta = B_{fx} \qquad \textbf{Eqn. 20}$$

$$\sin\psi B\cos\delta = -B_{fy} \qquad \textbf{Eqn. 21}$$

$$\Rightarrow \tan(\psi) = \left(\frac{-B_{fy}}{B_{fx}}\right) = \left(\frac{(B_{pz}-V_z)\sin\phi - (B_{py}-V_y)\cos\phi}{(B_{px}-V_x)\cos\theta + (B_{py}-V_y)\sin\theta\sin\phi + (B_{pz}-V_z)\sin\theta\cos\phi}\right) \qquad \textbf{Eqn. 22}$$

Equation 22 allows solution for the yaw angle ψ where ψ is computed relative to magnetic north. The yaw angle ψ is therefore the required tilt-compensated eCompass heading.

Since Equations 13, 15 and 22 have an infinite number of solutions at multiples of 360°, it is standard convention to restrict the solutions for roll, pitch and yaw to the range -180° to 180°. A further constraint is imposed on the pitch angle to limit it to the range -90° to 90°. This ensures only one unique solution exists for the compass, pitch and roll angles for any phone orientation. Equations 13 and 22 are therefore computed with a software ATAN2 function (with output angle range -180° to 180°) and Equation 15 is computed with a software ATAN function (with output angle range -90° to 90°).

5 Estimation of the Hard-Iron Offset V

Equation 22 assumes knowledge of the Hard-Iron offset $\boldsymbol{V}$, which is a fixed magnetic offset adding to the true magnetometer sensor output. The Hard-Iron offset is the sum of any intrinsic zero field offset within the magnetometer sensor itself plus permanent magnetic fields within the PCB generated by magnetized ferromagnetic materials. It is quite normal for the Hard-Iron offset to greatly exceed the geomagnetic field. Therefore an accurate Hard-Iron estimation and subtraction are required to avoid Equation 22 jamming and returning compass angles within a limited range only. It is common practice for magnetometer sensors to be supplied without zero field offset calibration since the standard Hard-Iron estimation algorithms will compute the sum of both the magnetometer sensor zero field offset and the PCB Hard-Iron offset.

In the absence of any Hard-Iron effects, the locus of the magnetometer output under arbitrary phone orientation changes lies on the surface of a sphere in the space of B_{px}, B_{py} and B_{pz} with a radius equal to the magnitude of the geomagnetic field B. In the presence of Hard-Iron effects, the locus of the magnetic measurements is simply displaced by the Hard-Iron vector $\boldsymbol{V}$ so that the origin of the sphere is equal to the Hard-Iron offset V_x, V_y and V_z. The Hard-Iron offset $\boldsymbol{V}$ can then be computed by fitting the magnetometer measurements to the equation:

$$(\boldsymbol{B}_p - \boldsymbol{V})^T(\boldsymbol{B}_p - \boldsymbol{V}) = B^2 \quad \textbf{Eqn. 23}$$

The mathematics and algorithms for computing $\boldsymbol{V}$ using Equation 23 are documented in Freescale application note AN4246, “Calibrating an eCompass in the Presence of Hard and Soft-Iron Interference”.

6 Visualization Using Experimental Data

This section uses accelerometer and magnetometer measurements to visualize the transformations described mathematically in this document.

Figures 3 and 4 show scatter plots of accelerometer and magnetometer data taken as an eCompass PCB is rotated in yaw, pitch and roll angles. Each accelerometer measurement is paired with a magnetometer measurement taken at the same time.

As predicted by Equation 3, the accelerometer measurements in Figure 3 lie on the surface of a sphere with radius equal to the earth's gravitational field measured in mg. The slight deviation of the accelerometer measurements from the sphere is caused by handshake adding to the gravitational component during the measurement process.

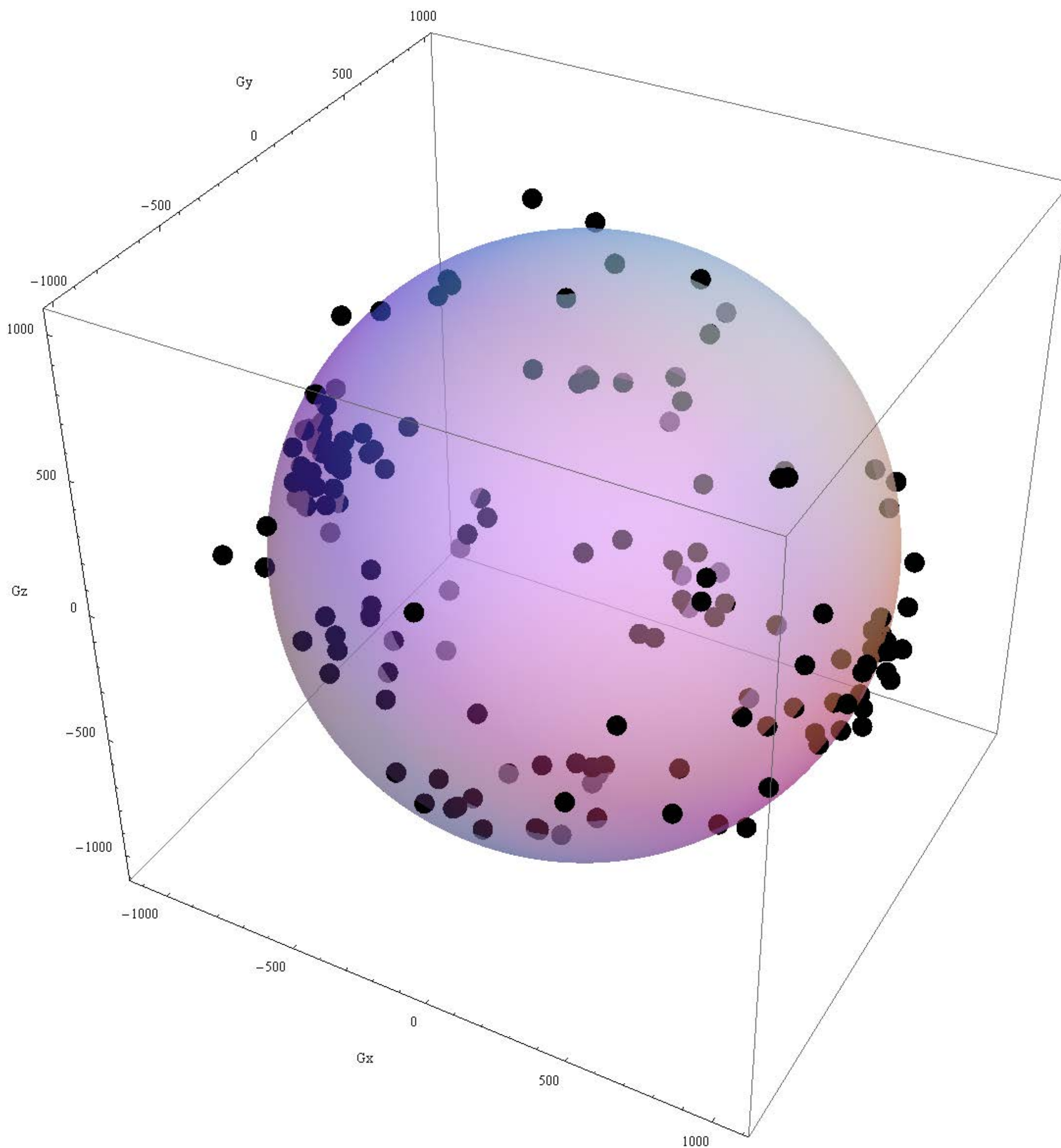


Figure 3. Scatter plot of accelerometer readings taken over a variety of orientation angles.

Similarly, as predicted by Equation 8, the magnetometer measurements in Figure 4 lie on the surface of a sphere with radius equal to the geomagnetic field strength B centered at the hard-Iron offset $\boldsymbol{V}$.

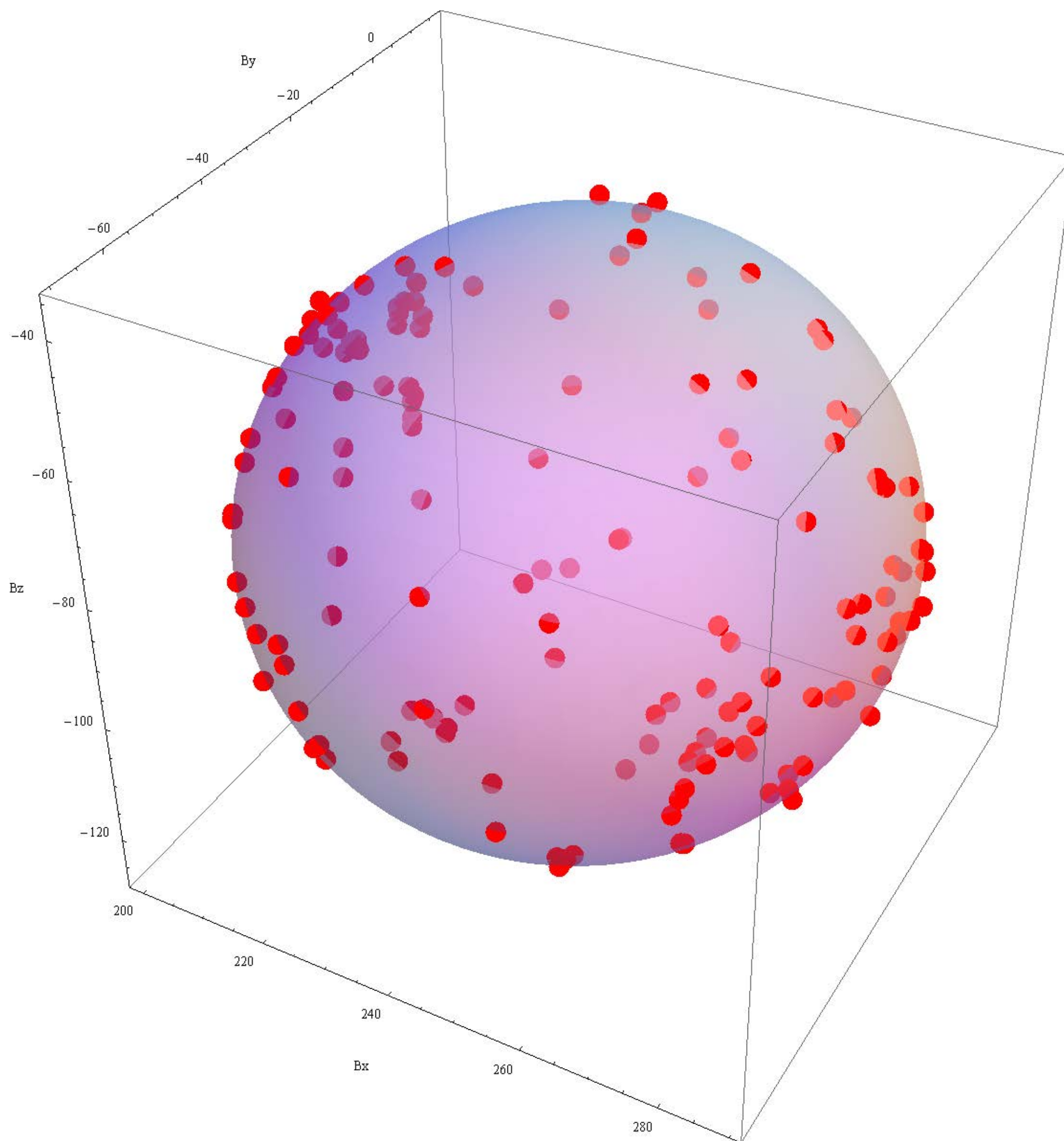


Figure 4. Scatter plot of raw magnetometer readings taken in parallel with the accelerometer readings of Figure 3.

Figure 5 shows the magnetometer readings of Figure 4 after correction for the hard-Iron offset using the simple algorithm of Equation 23. As predicted by Equation 8, these corrected readings $\boldsymbol{B_p}$-$\boldsymbol{V}$ lie on the surface of a sphere with radius equal to the geomagnetic field strength B centered at the origin.

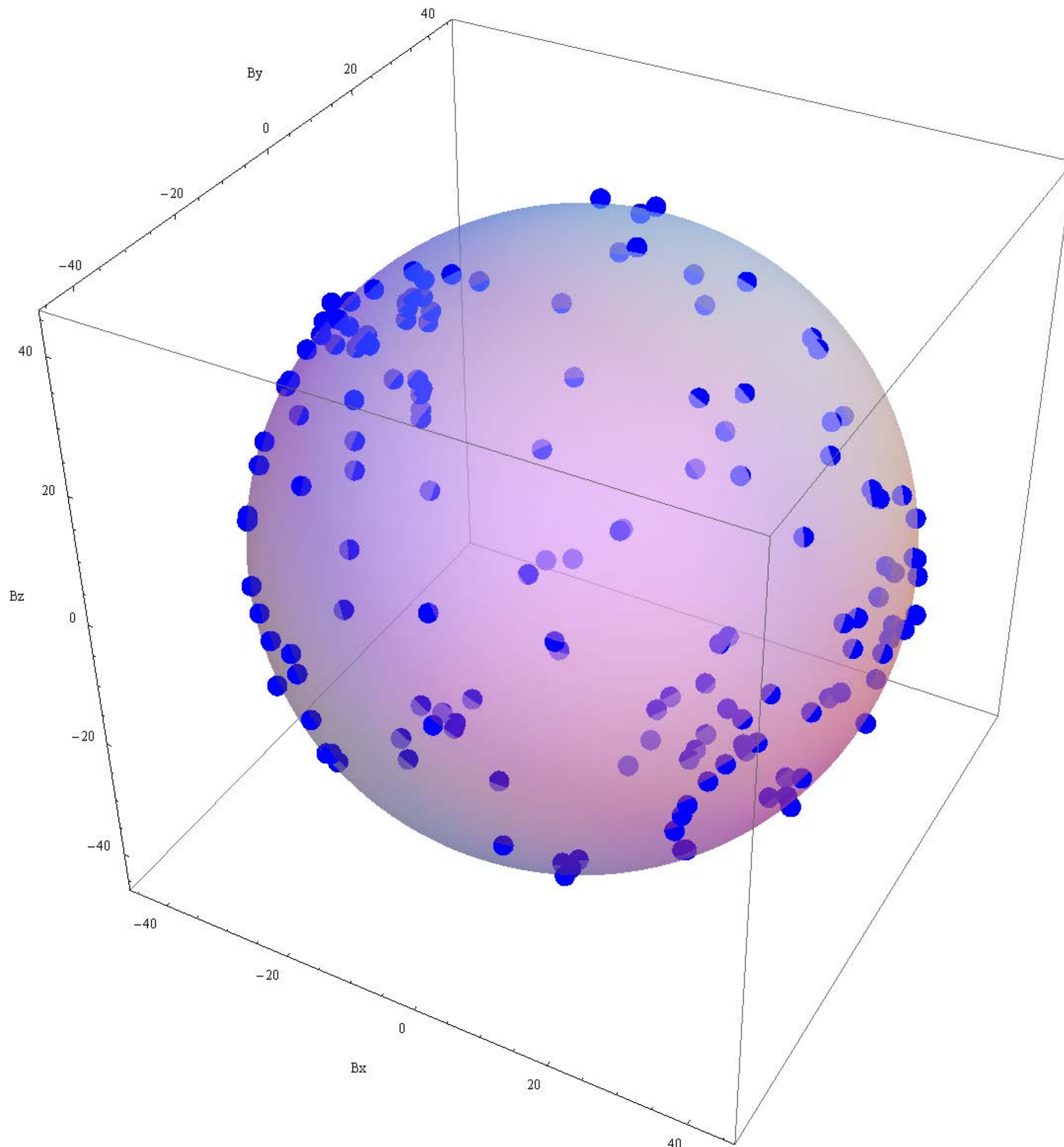


Figure 5. Scatter plot of calibrated magnetometer readings corrected for the hard-Iron offset.

Figure 6 shows the accelerometer readings corrected for roll and pitch angles using Equation 9. The corrected measurements, defined as $\boldsymbol{R}_y(-\theta)\boldsymbol{R}_x(-\phi)\boldsymbol{G}_p$, have zero x and y components and z component approximately equal to 1g. The slight variation in the z component results simply from noise or handshake during the measurement process.

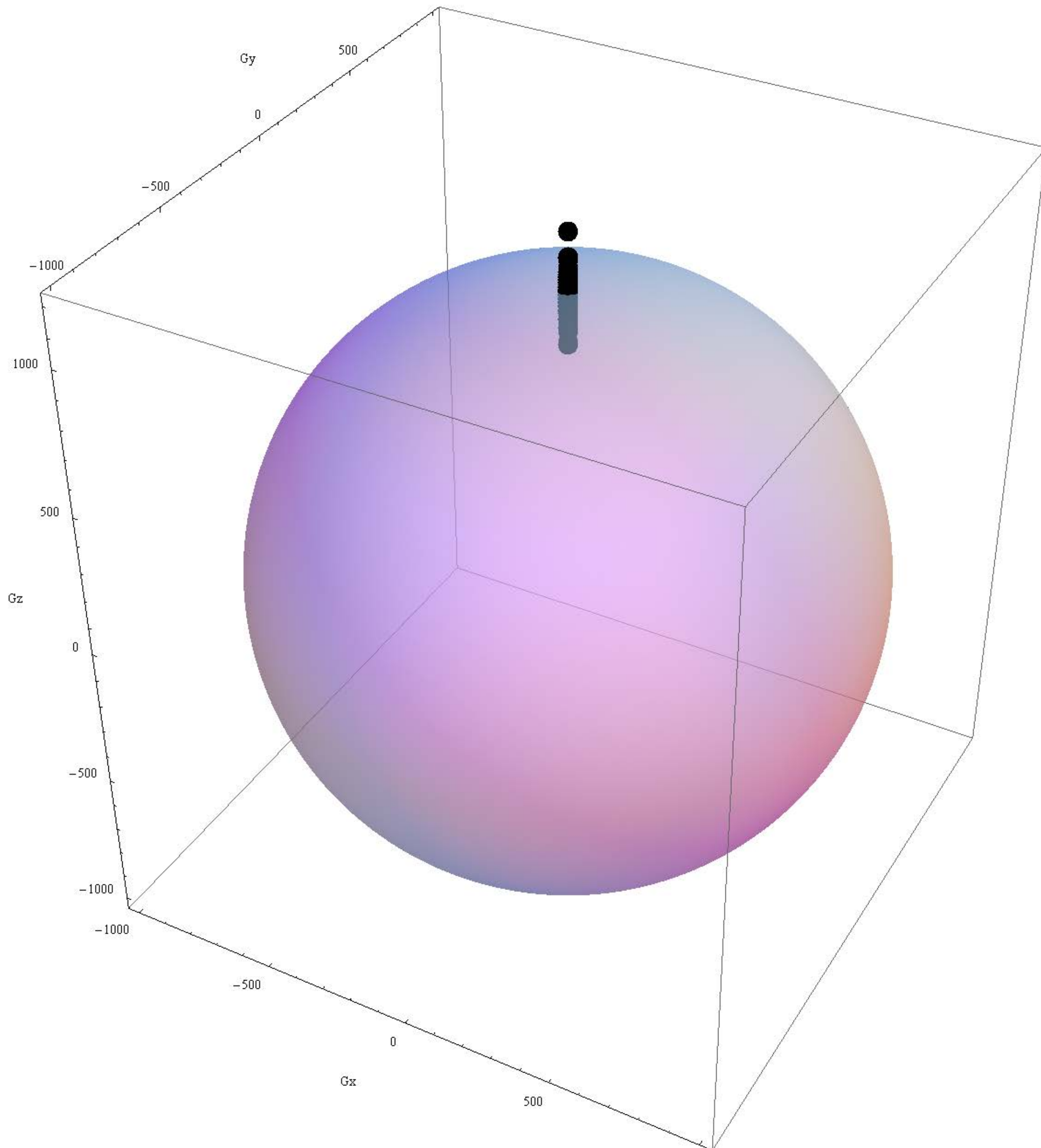


Figure 6. Scatter plot of accelerometer readings corrected for roll and pitch.

Figure 7 shows the magnetometer readings of Figure 3 further corrected for roll and pitch. As predicted by Equation 8, these measurements, defined as $\boldsymbol{R}_y(-\theta)\boldsymbol{R}_x(-\phi)(\boldsymbol{B}_p - \boldsymbol{V})$, have a circular distribution in the x and y axes and a constant z component equal to $B\sin\delta$.

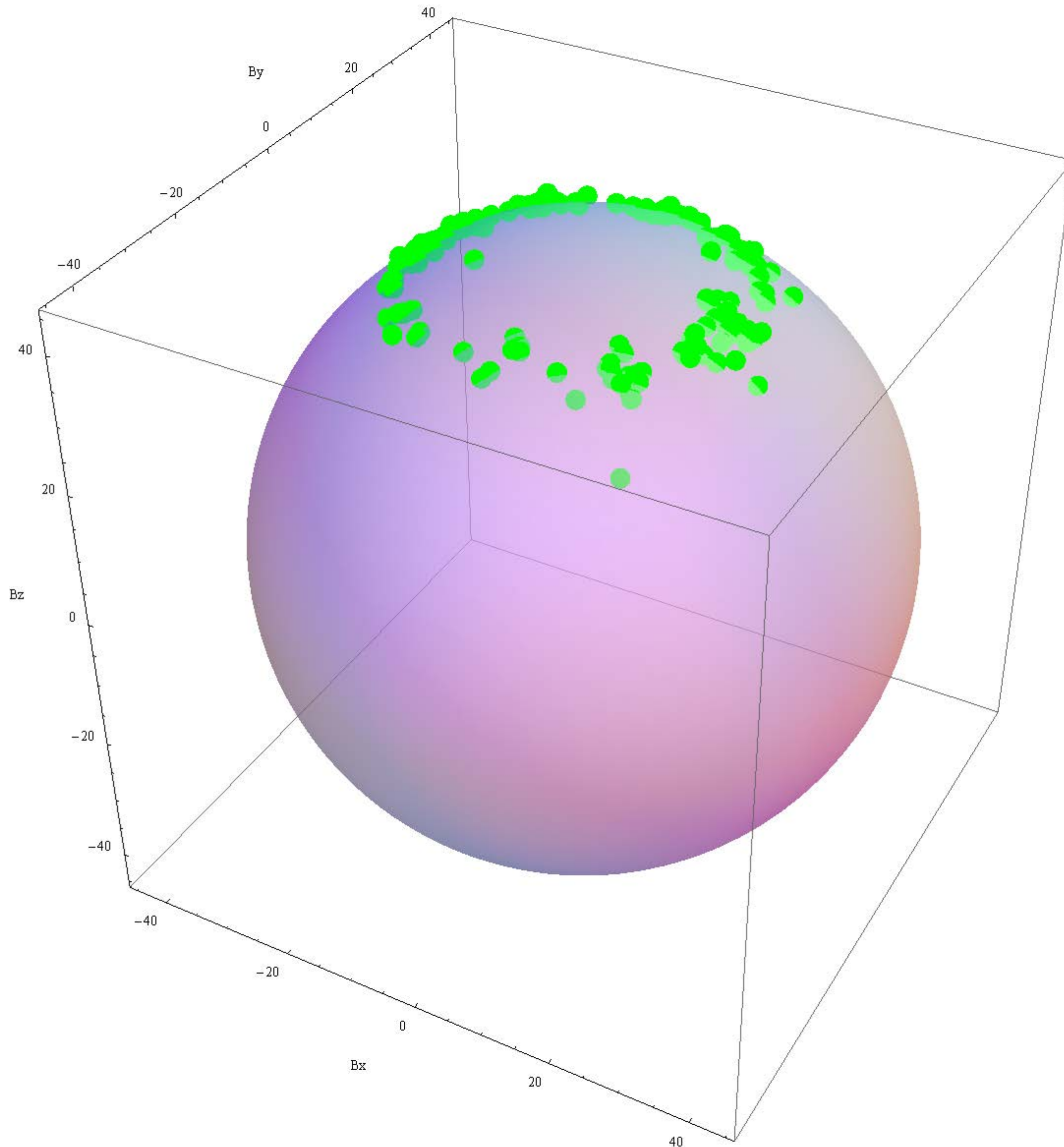


Figure 7. Scatter plot of calibrated magnetometer readings corrected for roll and pitch

7 Software Implementation

The reference C# code in this documentation uses integer operands only and makes no calls to any external mathematical libraries. Custom functions are provided in this document for all the trigonometric and numerical calculations required.

The accelerometer and magnetometer readings are assumed to fit within a signed 16-bit Int16 (since the most sensitive consumer accelerometers and magnetometers currently provide a maximum of 14 bits of data). All calculations are performed on the raw Int16 data read from the sensors without any need to convert to physical units of ms^{-2} or µT. It is, however, recommended that the user implement fixed multipliers to boost the accelerometer and magnetometer readings closer to the maximum range -32768 to +32767 to reduce quantization noise in the mathematical routines. For example, if the accelerometer data is signed 14-bit with range -2^{13} to $2^{13}-1$, then the multiplier should be 4x to maximize the dynamic range.

The trigonometric rotations used to rotate the sensor measurements use 16 x 16 bit multiplies into a 32-bit integer with the result returned as an Int16. Q15 fractional arithmetic is used for the sines and cosines where -32768 represents -1.00 and +32767 represents 0.999.

Sines and Cosines are computed directly from ratios of the accelerometer and magnetometer data rather than from angles. Angles are computed using a custom ATAN2 function which returns angles in degrees times 100 so 30° is output as 3000 decimal and -45° as -4500 decimal. This provides 0.01° angular resolution within the word length of an Int16.

7.1 eCompass C# Source Code

The tilt-compensated eCompass function is listed below. The function calls the trigonometric functions iTrig and iHundredAtan2Deg. The three angles computed should be low-pass filtered (see next section).

Global variables used by the function are listed immediately below.

```
/* roll pitch and yaw angles computed by iecompass */
static Int16 iPhi, iThe, iPsi;
/* magnetic field readings corrected for hard iron effects and PCB orientation */
static Int16 iBfx, iBfy, iBfz;
/* hard iron estimate */
static Int16 iVx, iVy, iVz;

/* tilt-compensated e-Compass code */
public static void iecompass(Int16 iBpx, Int16 iBpy, Int16 iBpz,
                             Int16 iGpx, Int16 iGpy, Int16 iGpz)
{
/* stack variables */
/* iBpx, iBpy, iBpz: the three components of the magnetometer sensor */
/* iGpx, iGpy, iGpz: the three components of the accelerometer sensor */

/* local variables */
Int16 iSin, iCos;          /* sine and cosine */

/* subtract the hard iron offset */
iBpx -= iVx;      /* see Eq 16 */
iBpy -= iVy;      /* see Eq 16 */
iBpz -= iVz;      /* see Eq 16 */
```

```
/* calculate current roll angle Phi */
iPhi = iHundredAtan2Deg(iGpy, iGpz);/* Eq 13 */
/* calculate sin and cosine of roll angle Phi */
iSin = iTrig(iGpy, iGpz);  /* Eq 13: sin = opposite / hypotenuse */
iCos = iTrig(iGpz, iGpy);  /* Eq 13: cos = adjacent / hypotenuse */
/* de-rotate by roll angle Phi */
iBfy = (Int16)((iBpy * iCos - iBpz * iSin) >> 15);/* Eq 19 y component */
iBpz = (Int16)((iBpy * iSin + iBpz * iCos) >> 15);/* Bpy*sin(Phi)+Bpz*cos(Phi)*/
iGpz = (Int16)((iGpy * iSin + iGpz * iCos) >> 15);/* Eq 15 denominator */

/* calculate current pitch angle Theta */
iThe = iHundredAtan2Deg((Int16)-iGpx, iGpz);/* Eq 15 */
/* restrict pitch angle to range -90 to 90 degrees */
if (iThe > 9000) iThe = (Int16) (18000 - iThe);
if (iThe < -9000) iThe = (Int16) (-18000 - iThe);
/* calculate sin and cosine of pitch angle Theta */
iSin = (Int16)-iTrig(iGpx, iGpz);    /* Eq 15: sin = opposite / hypotenuse */
iCos = iTrig(iGpz, iGpx);            /* Eq 15: cos = adjacent / hypotenuse */
/* correct cosine if pitch not in range -90 to 90 degrees */
if (iCos < 0) iCos = (Int16)-iCos;
/* de-rotate by pitch angle Theta */
iBfx = (Int16)((iBpx * iCos + iBpz * iSin) >> 15); /* Eq 19: x component */
iBfz = (Int16)((-iBpx * iSin + iBpz * iCos) >> 15);/* Eq 19: z component */

/* calculate current yaw = e-compass angle Psi */
iPsi = iHundredAtan2Deg((Int16)-iBfy, iBfx);  /* Eq 22 */
}
```

7.2 Modulo Arithmetic Low Pass Filter for Angles C# Source Code

The code for a simple exponential low pass filter, modulo 360°, for the output angles is listed below.

The filter has a single pole on the real axis at $z = 1 - \alpha$ and has transfer function $H(z)$ given by:

$$H(z) = \left(\frac{\alpha}{1-(1-\alpha)z^{-1}}\right) \quad \textbf{Eqn. 24}$$

The difference equation filtering the input series $x[n]$ into output $y[n]$ is given by:

$$y[n] = (1-\alpha)y[n-1] + \alpha x[n] \quad \textbf{Eqn. 25}$$

or equivalently in software:

$$yn \mathrel{+}= \alpha * (xn - yn); \quad \textbf{Eqn. 26}$$

The time constant in samples is given by the reciprocal of the filter coefficient α.

The additional complexity of the code below is to implement the filter in modulo arithmetic so that a sample to sample angle change of 359° is correctly interpreted as a -1° change.

The code is written for filtering the yaw (compass) angle ψ but can also be used for the roll angle ϕ with changes to use iPhi instead of iPsi.

```
Int32 tmpAngle;          /* temporary angle*100 deg: range -36000 to 36000 */
static Int16 iLPPsi;     /* low pass filtered angle*100 deg: range -18000 to 18000 */
```

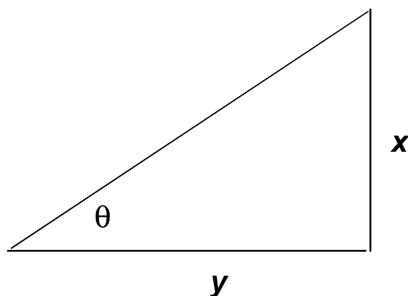
```
static UInt16 ANGLE_LPF;  /* low pass filter: set to 32768 / N for N samples averaging */

/* implement a modulo arithmetic exponential low pass filter on the yaw angle */
/* compute the change in angle modulo 360 degrees */
tmpAngle = (Int32)iPsi - (Int32)iLPPsi;
if (tmpAngle > 18000) tmpAngle -= 36000;
if (tmpAngle < -18000) tmpAngle += 36000;
/* calculate the new low pass filtered angle */
tmpAngle = (Int32)iLPPsi + ((ANGLE_LPF * tmpAngle) >> 15);
/* check that the angle remains in -180 to 180 deg bounds */
if (tmpAngle > 18000) tmpAngle -= 36000;
if (tmpAngle < -18000) tmpAngle += 36000;
/* store the correctly bounded low pass filtered angle */
iLPPsi = (Int16)tmpAngle;
```

For the pitch angle θ, which is restricted to the range -90° to 90°, the final bounds check should be changed to:

```
if (tmpAngle > 9000) tmpAngle = (Int16) (18000 - tmpAngle);
if (tmpAngle < -9000) tmpAngle = (Int16) (-18000 - tmpAngle);
```

7.3 Sine and Cosine Calculation C# Source Code

The function iTrig computes angle sines and cosines using the definitions:



$$\sin\theta = \frac{x}{\sqrt{x^2 + y^2}} \quad \textbf{Eqn. 27}$$

$$\cos\theta = \frac{y}{\sqrt{x^2 + y^2}} \quad \textbf{Eqn. 28}$$

The function uses a binary division algorithm to solve for r where:

$$r^2(x^2 + y^2) = x^2 \quad \textbf{Eqn. 29}$$

The accuracy is determined by the threshold MINDELTATRIG. The setting for maximum accuracy is MINDELTATRIG = 1.

```
const UInt16 MINDELTATRIG = 1;           /* final step size for iTrig */
```

```
/* function to calculate ir = ix / sqrt(ix*ix+iy*iy) using binary division */
static Int16 iTrig(Int16 ix, Int16 iy)
{
UInt32 itmp;      /* scratch */
UInt32 ixsq;      /* ix * ix */
Int16 isignx;     /* storage for sign of x. algorithm assumes x >= 0 then corrects later */
UInt32 ihypsq;    /* (ix * ix) + (iy * iy) */
Int16 ir;         /* result = ix / sqrt(ix*ix+iy*iy) range -1, 1 returned as signed Int16 */
Int16 idelta;     /* delta on candidate result dividing each stage by factor of 2 */

/* stack variables */
/* ix, iy: signed 16 bit integers representing sensor reading in range -32768 to 32767 */
/* function returns signed Int16 as signed fraction (ie +32767=0.99997, -32768=-1.0000) */
/* algorithm solves for ir*ir*(ix*ix+iy*iy)=ix*ix */

/* correct for pathological case: ix==iy==0 */
if ((ix == 0) && (iy == 0)) ix = iy = 1;

/* check for -32768 which is not handled correctly */
if (ix == -32768) ix = -32767;
if (iy == -32768) iy = -32767;

/* store the sign for later use. algorithm assumes x is positive for convenience */
isignx = 1;
if (ix < 0)
{
        ix = (Int16)-ix;
        isignx = -1;
}

/* for convenience in the boosting set iy to be positive as well as ix */
iy = (Int16)Math.Abs(iy);

/* to reduce quantization effects, boost ix and iy but keep below maximum signed 16 bit */
while ((ix < 16384) && (iy < 16384))
{
        ix = (Int16)(ix + ix);
        iy = (Int16)(iy + iy);
}

/* calculate ix*ix and the hypotenuse squared */
ixsq = (UInt32)(ix * ix);            /* ixsq=ix*ix: 0 to 32767^2 = 1073676289 */
ihypsq = (UInt32)(ixsq + iy * iy);   /* ihypsq=(ix*ix+iy*iy) 0 to 2*32767*32767=2147352578 */

/* set result r to zero and binary search step to 16384 = 0.5 */
ir = 0;
idelta = 16384;                      /* set as 2^14 = 0.5 */

/* loop over binary sub-division algorithm */
do
{
        /* generate new candidate solution for ir and test if we are too high or too low */
        /* itmp=(ir+delta)^2, range 0 to 32767*32767 = 2^30 = 1073676289 */
        itmp = (UInt32)((ir + idelta) * (ir + idelta));
        /* itmp=(ir+delta)^2*(ix*ix+iy*iy), range 0 to 2^31 = 2147221516 */
        itmp = (itmp >> 15) * (ihypsq >> 15);
```

```
        if (itmp <= ixsq) ir += idelta;
        idelta = (Int16)(idelta >> 1);       /* divide by 2 using right shift one bit */
} while (idelta >= MINDELTATRIG);     /* last loop is performed for idelta=MINDELTATRIG */

/* correct the sign before returning */
return (Int16)(ir * isignx);
}
```

7.4 ATAN2 Calculation C# Source Code

The function iHundredAtan2Deg is a wrapper function which implements the ATAN2 function by assigning the results of an ATAN function to the correct quadrant. The result is the angle in degrees times 100.

```
/* calculates 100*atan2(iy/ix)=100*atan2(iy,ix) in deg for ix, iy in range -32768 to 32767 */
static Int16 iHundredAtan2Deg(Int16 iy, Int16 ix)
{
Int16 iResult;    /* angle in degrees times 100 */

/* check for -32768 which is not handled correctly */
if (ix == -32768) ix = -32767;
if (iy == -32768) iy = -32767;

/* check for quadrants */
if ((ix >= 0) && (iy >= 0))         /* range 0 to 90 degrees */
        iResult = iHundredAtanDeg(iy, ix);
else if ((ix <= 0) && (iy >= 0))    /* range 90 to 180 degrees */
        iResult = (Int16)(18000 - (Int16)iHundredAtanDeg(iy, (Int16)-ix));
else if ((ix <= 0) && (iy <= 0))    /* range -180 to -90 degrees */
        iResult = (Int16)((Int16)-18000 + iHundredAtanDeg((Int16)-iy, (Int16)-ix));
else                                /* ix >=0 and iy <= 0 giving range -90 to 0 degrees */
        iResult = (Int16)(-iHundredAtanDeg((Int16)-iy, ix));
return (iResult);
}
```

7.5 ATAN Calculation C# Source Code

The function iHundredAtanDeg computes the $ATAN\left(\frac{Y}{X}\right)$ function for X and Y in the range 0 to 32767 (interpreted as 0.0 to 0.9999695 in Q15 fractional arithmetic) outputting the angle in degrees * 100 in the range 0 to 9000 (0.0° to 90.0°).

For $Y \le X$ the output angle is in the range 0° to 45° and is computed using the polynomial approximation:

$$\text{Angle} * 100 = \left\{\frac{K1}{2^{15}}\left(\frac{Y}{X}\right) + \frac{K2}{2^{45}}\left(\frac{Y}{X}\right)^3 + \frac{K3}{2^{75}}\left(\frac{Y}{X}\right)^5\right\} \quad \textbf{Eqn. 30}$$

For $Y > X$, the identity is used (valid in degrees for positive x):

$$\operatorname{atan}(x) = 90 - \operatorname{atan}\left(\frac{1}{x}\right) \quad \textbf{Eqn. 31}$$

$$\text{Angle} * 100 = 9000 - \left\{ \frac{K1}{2^{15}}\left(\frac{X}{Y}\right) + \frac{K2}{2^{45}}\left(\frac{X}{Y}\right)^3 + \frac{K3}{2^{75}}\left(\frac{X}{Y}\right)^5 \right\} \quad \textbf{Eqn. 32}$$

K1, K2 and *K3* were computed by brute force optimization to minimize the maximum error.

```
/* fifth order of polynomial approximation giving 0.05 deg max error */
const Int16 K1 = 5701;
const Int16 K2 = -1645;
const Int16 K3 = 446;

/* calculates 100*atan(iy/ix) range 0 to 9000 for all ix, iy positive in range 0 to 32767 */
static Int16 iHundredAtanDeg(Int16 iy, Int16 ix)
{
Int32 iAngle;   /* angle in degrees times 100 */
Int16 iRatio;   /* ratio of iy / ix or vice versa */
Int32 iTmp;     /* temporary variable */

/* check for pathological cases */
if ((ix == 0) && (iy == 0)) return (0);
if ((ix == 0) && (iy != 0)) return (9000);

/* check for non-pathological cases */
if (iy <= ix)
         iRatio = iDivide(iy, ix); /* return a fraction in range 0. to 32767 = 0. to 1. */
else
         iRatio = iDivide(ix, iy); /* return a fraction in range 0. to 32767 = 0. to 1. */

/* first, third and fifth order polynomial approximation */
iAngle = (Int32) K1 * (Int32) iRatio;
iTmp = ((Int32) iRatio >> 5) * ((Int32) iRatio >> 5) * ((Int32) iRatio >> 5);
iAngle += (iTmp >> 15) * (Int32) K2;
iTmp = (iTmp >> 20) * ((Int32) iRatio >> 5) * ((Int32) iRatio >> 5)
iAngle += (iTmp >> 15) * (Int32) K3;
iAngle = iAngle >> 15;

/* check if above 45 degrees */
if (iy > ix) iAngle = (Int16)(9000 - iAngle);

/* for tidiness, limit result to range 0 to 9000 equals 0.0 to 90.0 degrees */
if (iAngle < 0) iAngle = 0;
if (iAngle > 9000) iAngle = 9000;

return ((Int16) iAngle);
}
```

7.6 Integer Division C# Source Code

The function iDivide is an accurate integer division function where it is given that both the numerator and denominator are non-negative, non-zero and where the denominator is greater than the numerator. The result is in the range 0 decimal to 32767 decimal which is interpreted in Q15 fractional arithmetic as the range 0.0 to 0.9999695.

The function solves for r where:

$$r = \left(\frac{y}{x}\right) \quad \text{Eqn. 33}$$

using a binary division algorithm to solve for:

$$rx = y \quad \text{Eqn. 34}$$

The accuracy is determined by the threshold MINDELTADIV. The setting for maximum accuracy is MINDELTADIV = 1.

```
const UInt16 MINDELTADIV = 1;              /* final step size for iDivide */

/* function to calculate ir = iy / ix with iy <= ix, and ix, iy both > 0 */
static Int16 iDivide(Int16 iy, Int16 ix)
{
Int16 itmp;                  /* scratch */
Int16 ir;                    /* result = iy / ix range 0., 1. returned in range 0 to 32767 */
Int16 idelta;                /* delta on candidate result dividing each stage by factor of 2 */

/* set result r to zero and binary search step to 16384 = 0.5 */
ir = 0;
idelta = 16384;                          /* set as 2^14 = 0.5 */

/* to reduce quantization effects, boost ix and iy to the maximum signed 16 bit value */
while ((ix < 16384) && (iy < 16384))
{
        ix = (Int16)(ix + ix);
        iy = (Int16)(iy + iy);
}

/* loop over binary sub-division algorithm solving for ir*ix = iy */
do
{
        /* generate new candidate solution for ir and test if we are too high or too low */
        itmp = (Int16)(ir + idelta);        /* itmp=ir+delta, the candidate solution */
        itmp = (Int16)((itmp * ix) >> 15);
        if (itmp <= iy) ir += idelta;
        idelta = (Int16)(idelta >> 1);      /* divide by 2 using right shift one bit */
} while (idelta >= MINDELTADIV);    /* last loop is performed for idelta=MINDELTADIV */

return (ir);
}
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: http://www.reg.net/v2/webservices/Freescale/Docs/TermsandConditions.htm.

Freescale and the Freescale logo are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.
© 2013, 2015 Freescale Semiconductor, Inc.

Document Number: AN4248
Rev. 4.0
11/2015

freescale™