Freescale Semiconductor
Application Note

Document Number:AN4696
Rev. 1.0, 4/2013

Accelerometer and Magnetometer Sensor Simulation for Tilt-Compensated eCompass

by: Mark Pedley

Contents

1 Introduction

This Application Note documents the accelerometer and magnetometer sensor simulation function in Freescale's Xtrinsic eCompass and magnetic calibration software at www.freescale.com/ecompass. This Application Note is part of the technical documentation for that software and its use and distribution are controlled by the license agreement.

1.1 Background

The function of the sensor simulation software is to allow validation of eCompass and magnetic calibration algorithms in the Aerospace, Android and Windows 8 coordinate systems for specified hard and soft iron distortion before interfacing to real sensors. Sensor noise is not modeled since this is sensor specific and adds an additional layer of complexity which is not relevant to algorithm verification.

All the Freescale Xtrinsic software available for download has, however, been fully verified with real sensors in real-time systems and closely matches the software used in Freescale's sensor toolbox eCompass software versions 6.0.0 and higher.

2 Accelerometer and Magnetometer Simulation

2.1 Coordinate systems

The coordinate systems and rotation matrices under the Aerospace/NED, Android and Windows 8 standards are discussed in equations (1) through (3).

The rotation matrices are defined in the sense of mapping a vector defined in the earth's reference frame to the smartphone reference frame due to a rotation of the smartphone coordinate system by Euler angles ϕ, θ, and ψ in roll, pitch and yaw. The reference orientation corresponding to zero roll, pitch and yaw angles corresponds to the smartphone lying flat, face up and pointing towards magnetic north.

2.1.1 Aerospace/NED

The Aerospace/NED rotation matrix is:

$$\boldsymbol{R}_{NED} = \begin{pmatrix} cos\,\theta\,cos\,\psi & cos\,\theta\,sin\psi & -sin\theta \\ cos\,\psi\,sin\theta sin\phi - cos\,\phi\,sin\psi & cos\,\phi\,cos\,\psi + sin\theta sin\phi sin\psi & cos\,\theta\,sin\phi \\ cos\phi\,cos\,\psi\,sin\theta + sin\phi sin\psi & cos\,\phi\,sin\theta sin\psi - cos\,\psi\,sin\phi & cos\,\theta\,cos\phi \end{pmatrix} \quad (1)$$

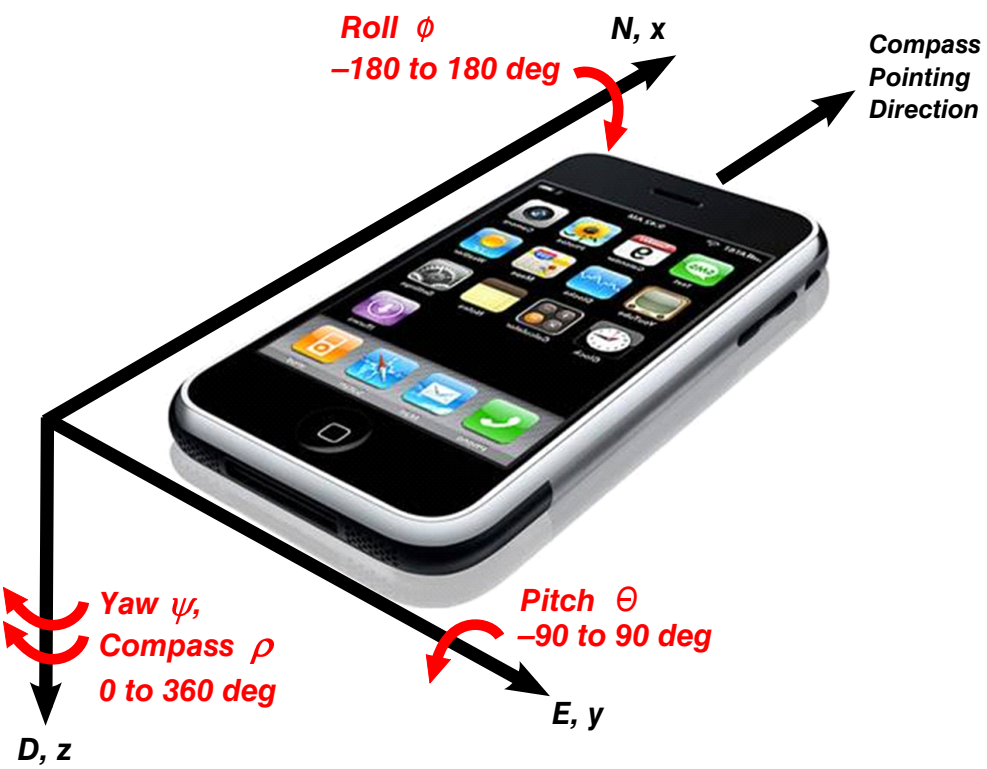

Figure 1. Aerospace/NED coordinate system

2.1.2 Android

The Android rotation matrix is:

$$\boldsymbol{R}_{Android} = \begin{pmatrix} cos\,\phi\,cos\,\psi & -cos\phi sin\psi & sin\phi \\ cos\,\theta\,sin\psi + cos\,\psi\,sin\phi sin\theta & cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta & -cos\,\phi\,sin\theta \\ -cos\,\psi\,cos\,\theta\,sin\phi + sin\psi sin\theta & cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \quad (2)$$

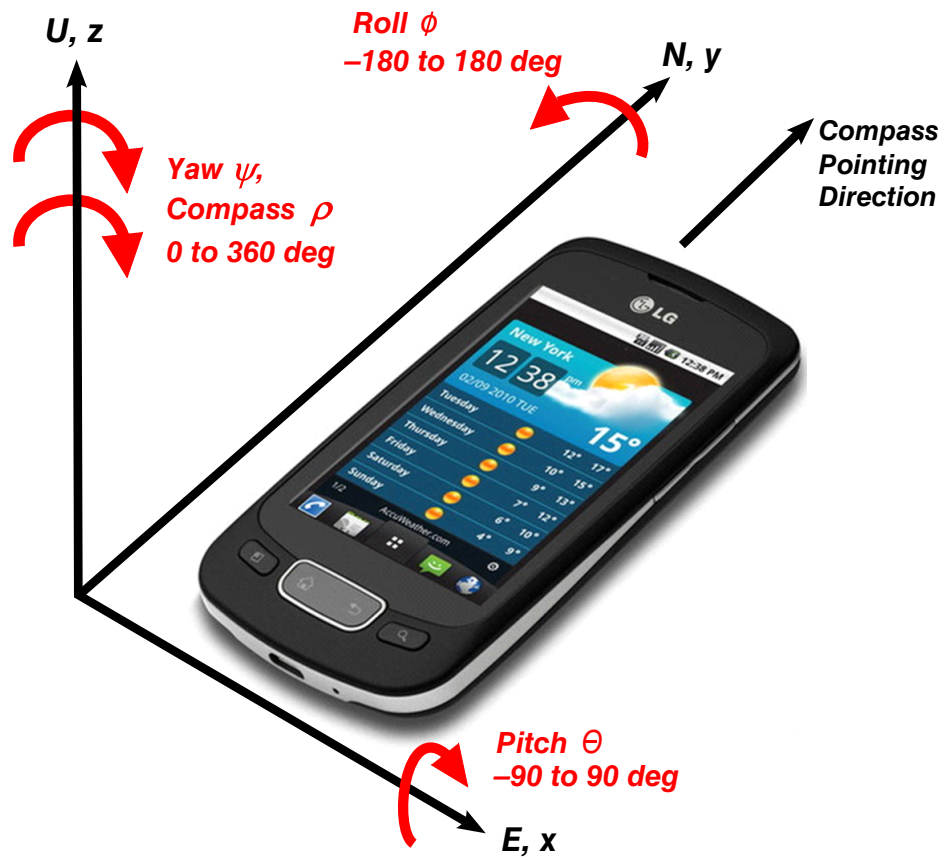

Figure 2. Android coordinate system

2.1.3 Windows 8

The Windows 8 rotation matrix is:

$$\boldsymbol{R}_{Win8} = \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -sin\phi\,cos\,\theta \\ -cos\,\theta\,sin\psi & cos\,\theta\,cos\,\psi & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \quad (3)$$

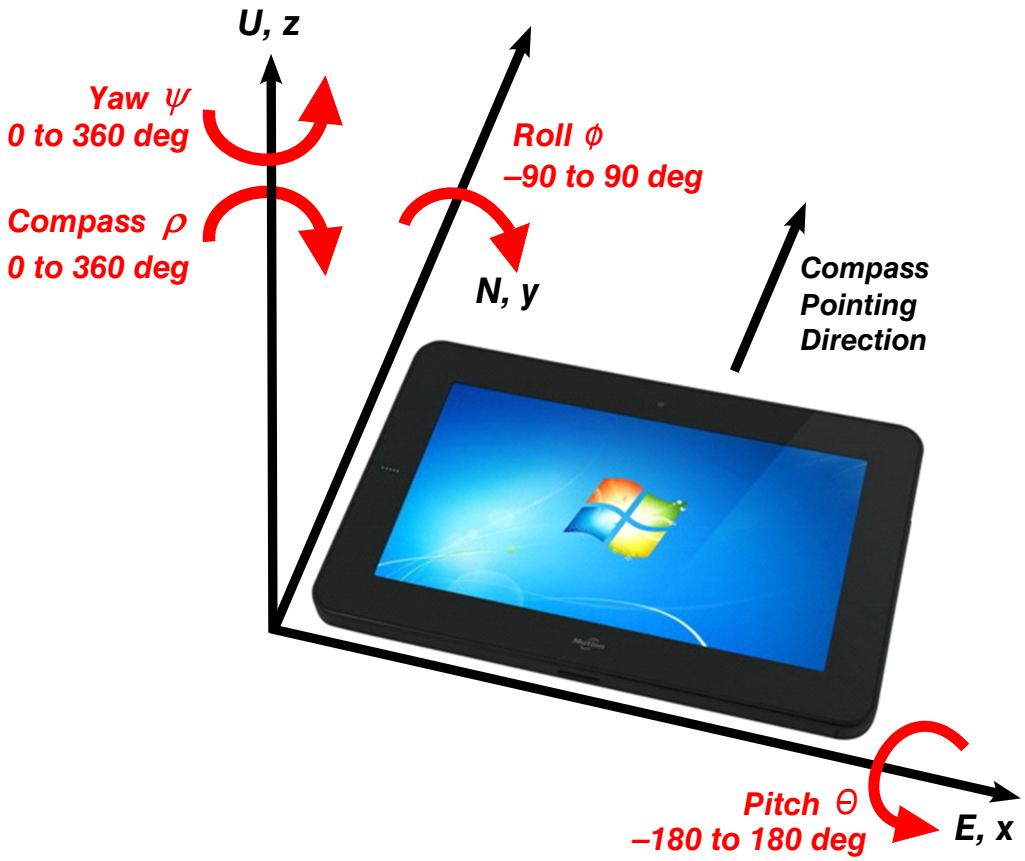

Figure 3. Windows 8 coordinate system

2.1.4 Key feature summary of coordinate systems

Table 1 summarizes key features of the three coordinate systems which will be referred to in later text.

Table 1. Key features of aerospace, Android and Windows 8 coordinate systems

Feature	Aerospace/NED	Android	Windows 8
Accelerometer output	Gravity-Acceleration *g*	Acceleration-Gravity *g*	Gravity-Acceleration *g*
Axes alignment	xyz = North-East-Down (NED)	xyz = East-North-Up (ENU)	xyz = East-North-Up (ENU)
Accelerometer z axis reading when flat	+1 *g*	+1 *g*	–1 *g*
Direction of Euler angles	Clockwise sense	Anti-clockwise sense	Clockwise sense
Ranges of Euler angles	0º ≤ Yaw ψ (*z*) < 360º –90º ≤ Pitch θ (*y*) < 90º –180º ≤ Roll ϕ (*x*) < 180º	0º ≤ Yaw ψ (*z*) < 360º –90º ≤ Roll ϕ (*y*) < 90º –180º ≤ Pitch θ (*x*) < 180º	0º ≤ Yaw ψ (*z*) < 360º –180º ≤ Pitch θ (*x*) < 180º –90º ≤ Roll ϕ (*y*) < 90º
Sequence of Euler angles	Yaw then pitch then roll $\boldsymbol{R} = \boldsymbol{R}_x(\phi)\boldsymbol{R}_y(\theta)\boldsymbol{R}_z(\psi)$	Yaw then roll then pitch $\boldsymbol{R} = \boldsymbol{R}_x(\theta)\boldsymbol{R}_y(\phi)\boldsymbol{R}_z(\psi)$	Yaw then pitch then roll $\boldsymbol{R} = \boldsymbol{R}_y(\phi)\boldsymbol{R}_x(\theta)\boldsymbol{R}_z(\psi)$
Compass heading ρ	$\rho = \psi$	$\rho = \psi$	ρ = 360º – ψ

2.2 Gravity and geomagnetic vectors in the earth's reference frame

Figure 4 shows the gravitational vector $\boldsymbol{g}$ and the geomagnetic vector $\boldsymbol{B}$ in the earth's reference frame. The gravitational vector always points downwards. The geomagnetic vector points towards magnetic north and (in the northern hemisphere) downwards from horizontal by the inclination angle δ.

Figure 4. Gravity and geomagnetic vectors in the earth's reference frame

Accelerometer sensors natively measure the sum of linear acceleration and the acceleration equivalent to the earth's gravitational field. The physics of being at rest in a 1 g gravitational field pointing downwards is equivalent to acceleration of 1 g in the opposite direction upwards (Einstein's equivalence principle). Accelerometer sensors therefore measure linear acceleration minus gravity and their x, y and z package axes are defined in this sense. Platform coordinate systems are at liberty to choose either of these two definitions which differ only in sign.

In addition, the mathematics of the tilt-compensated eCompass assume that there is no linear acceleration from handshake or other sources and that the accelerometer output is dependent on the smartphone orientation.

2.2.1 Aerospace/NED

The Aerospace/NED standard is most commonly applied as a gravity standard (which negates the accelerometer output) and because the gravity vector in the earth's reference frame is aligned along the Aerospace z axis, the accelerometer reading $\boldsymbol{G}_{NED,r}$ in the absence of any rotation is, in units of g:

$$\boldsymbol{G}_{NED,r} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (4)$$

The geomagnetic vector in the Aerospace/NED coordinate system in the absence of any rotation is, in units of g:

$$\boldsymbol{B}_{NED,r} = B\begin{pmatrix} cos\delta \\ 0 \\ sin\delta \end{pmatrix} \quad (5)$$

2.2.2 Android

Android is an acceleration positive standard (which therefore matches the accelerometer's native output once the accelerometer axes are aligned to the Android platform axes). The downwards pointing gravity vector is equivalent to 1g acceleration upwards and this equivalent acceleration is aligned along Android's positive z axis. The accelerometer reading $\boldsymbol{G}_{Android,r}$ in the Android coordinate system in the absence of rotation is:

$$\boldsymbol{G}_{Android,r} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (6)$$

The x and y axes in Android are swapped relative to the Aerospace/NED standard and the z axis is reversed. The geomagnetic vector in Android coordinate system in the absence of any rotation is, in units of g:

$$\boldsymbol{B}_{Android,r} = B\begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \quad (7)$$

2.2.3 Windows 8

Windows 8 is a gravity positive standard but since the Windows 8 z axis points upwards, the accelerometer reading $\boldsymbol{G}_{Win8,r}$ in the absence of rotation is:

$$\boldsymbol{G}_{Win8,r} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \quad (8)$$

The geomagnetic vector in Windows 8 coordinate system matches that for Android since the two coordinate systems are the same. In the absence of any rotation the geomagnetic vector has value:

$$\boldsymbol{B}_{Win8,r} = B\begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \quad (9)$$

2.3 Accelerometer sensor simulation

2.3.1 Aerospace/NED

The Aerospace/NED coordinate system is a gravity positive standard with the z axis in the initial orientation pointed downwards and aligned with gravity giving +1 g in the z axis. The Aerospace/NED accelerometer output $\boldsymbol{G}_p$ after smartphone rotation is then given by (in units of g):

$$\boldsymbol{G}_{NED,p} = \boldsymbol{R}_{NED}\boldsymbol{G}_{NED,r} \tag{10}$$

$$= \begin{pmatrix} cos\,\theta\,cos\,\psi & cos\,\theta\,sin\psi & -sin\theta \\ cos\,\psi\,sin\theta sin\phi - cos\,\phi\,sin\psi & cos\,\phi\,cos\,\psi + sin\theta sin\phi sin\psi & cos\,\theta\,sin\phi \\ cos\phi\,cos\,\psi\,sin\theta + sin\phi sin\psi & cos\,\phi\,sin\theta sin\psi - cos\,\psi\,sin\phi & cos\,\theta\,cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{pmatrix} \tag{11}$$

2.3.2 Android

The Android coordinate system is an acceleration positive, gravity negative, standard with the z axis in the initial orientation aligned upwards and in the same direction as the equivalent upwards acceleration to downwards gravity. The accelerometer reading in the initial orientation is therefore +1 g in the z axis. The Android accelerometer output $\boldsymbol{G}_p$ after smartphone rotation is then given by (in units of g):

$$\boldsymbol{G}_{Android,p} = \boldsymbol{R}_{Android}\boldsymbol{G}_{Android,r} \tag{12}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi & -cos\phi sin\psi & sin\phi \\ cos\,\theta\,sin\psi + cos\,\psi\,sin\phi sin\theta & cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta & -cos\,\phi\,sin\theta \\ -cos\,\psi\,cos\,\theta\,sin\phi + sin\psi sin\theta & cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} sin\phi \\ -cos\phi\,sin\,\theta \\ cos\phi\,cos\,\theta \end{pmatrix} \tag{13}$$

2.3.3 Windows 8

The Windows 8 coordinate system is a gravity positive standard with the z axis in the initial orientation aligned upwards and in the opposite direction to gravity. The accelerometer reading in the initial orientation is therefore –1 g in the z axis. The Windows 8 accelerometer output $\boldsymbol{G}_p$ after smartphone rotation is then given by (in units of g):

$$\boldsymbol{G}_{Win8,p} = \boldsymbol{R}_{Win8}\boldsymbol{G}_{Win8,r} \tag{14}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -cos\,\theta\,sin\phi \\ -cos\,\theta\,sin\psi & cos\,\psi\,cos\,\theta & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} cos\,\theta\,sin\phi \\ -sin\,\theta \\ -cos\,\theta\,cos\phi \end{pmatrix} \tag{15}$$

2.4 Magnetometer sensor simulation

2.4.1 Aerospace / NED

Ignoring any hard iron and soft iron effects resulting from the magnetometer sensor or the smartphone circuit board, the geomagnetic field $\boldsymbol{B}_c$ measured by the smartphone in the Aerospace / NED coordinate system will be:

$$\boldsymbol{B}_{NED,c} = \boldsymbol{R}_{NED}\boldsymbol{B}_{NED,r} \tag{16}$$

$$= \begin{pmatrix} cos\,\theta\,cos\,\psi & cos\,\theta\,sin\psi & -sin\theta \\ cos\,\psi\,sin\theta sin\phi - cos\,\phi\,sin\psi & cos\,\phi\,cos\,\psi + sin\theta sin\phi sin\psi & cos\,\theta\,sin\phi \\ cos\phi\,cos\,\psi\,sin\theta + sin\phi sin\psi & cos\,\phi\,sin\theta sin\psi - cos\,\psi\,sin\phi & cos\,\theta\,cos\phi \end{pmatrix} B \begin{pmatrix} cos\delta \\ 0 \\ sin\delta \end{pmatrix} \tag{17}$$

$$= B \begin{pmatrix} cos\,\theta\,cos\,\psi\,cos\delta - sin\theta sin\delta \\ (cos\,\psi\,sin\theta sin\phi - cos\,\phi\,sin\psi)cos\delta + cos\,\theta\,sin\phi sin\delta \\ (cos\,\psi\,sin\theta cos\phi + sin\phi sin\psi)cos\delta + cos\,\theta\,cos\phi sin\delta \end{pmatrix} \tag{18}$$

2.4.2 Android

Again ignoring hard and soft iron disturbance, the geomagnetic field $\boldsymbol{B}_c$ measured by the smartphone in the Android coordinate system will be:

$$\boldsymbol{B}_{Android,c} = \boldsymbol{R}_{Android}\boldsymbol{B}_{Android,r} \tag{19}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi & -cos\phi sin\psi & sin\phi \\ cos\,\theta\,sin\psi + cos\,\psi\,sin\phi sin\theta & cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta & -cos\,\phi\,sin\theta \\ -cos\,\psi\,cos\,\theta\,sin\phi + sin\psi sin\theta & cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \tag{20}$$

$$= B \begin{pmatrix} -sin\delta sin\phi - cos\delta cos\phi sin\psi \\ sin\delta\,cos\,\phi\,sin\theta + cos\delta(cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta) \\ -sin\delta\,cos\,\phi\,cos\,\theta + cos\delta(cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta) \end{pmatrix} \tag{21}$$

2.4.3 Windows 8

Again ignoring hard and soft iron disturbance, the geomagnetic field $\boldsymbol{B}_c$ measured by the smartphone in the Windows 8 coordinate system will be:

$$\boldsymbol{B}_{Win8,c} = \boldsymbol{R}_{Win8}\boldsymbol{B}_{Win8,r} \tag{22}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -sin\phi\,cos\,\theta \\ -cos\,\theta\,sin\psi & cos\,\theta\,cos\,\psi & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \tag{23}$$

$$= B \begin{pmatrix} cos\,\theta\,sin\phi sin\delta + (cos\,\psi\,sin\phi sin\theta + cos\,\phi\,sin\psi)cos\delta \\ cos\,\theta\,cos\,\psi\,cos\delta - sin\delta sin\theta \\ -cos\,\phi\,cos\,\theta\,sin\delta + (sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta)cos\delta \end{pmatrix} \tag{24}$$

2.5 Hard and soft iron effects

The most general linear model for incorporating hard and soft iron effects derives the magnetometer reading $\boldsymbol{B}_p$ from the geomagnetic field measured in the smartphone coordinate system $\boldsymbol{B}_c$ through multiplication by the soft iron matrix $\boldsymbol{W}$ and addition of a hard iron vector $\boldsymbol{V}$ as:

$$\boldsymbol{B}_p = \boldsymbol{W}\boldsymbol{B}_c + \boldsymbol{V} = \boldsymbol{W}\boldsymbol{R}\boldsymbol{B}_r + \boldsymbol{V} \quad (25)$$

2.6 Euler angles ranges and compass heading

In general, two solutions exist for the roll, pitch and yaw Euler angles consistent with a given rotation matrix and smartphone orientation. As a simple example, from a starting point with the smartphone flat, the same orientation results from applying either (a) a pitch change of 180º or (b) a yaw rotation of 180º and a roll rotation of 180º. Only a single solution results, however, if one of the Euler angles is restricted in range.

The Aerospace/NED coordinate system restricts the pitch angle to lie in the range –90º to 90º but allows the roll angle to span the full range of –180º to 180º. The Android and Windows 8 coordinate systems take the opposite approach of restricting the roll angle to lie in the range –90º to 90º and allowing the pitch angle to span the full range of –180º to 180º.

Examination of Figure 1 through Figure 3 shows that the compass heading angle ρ is equal to the yaw angle ψ for the Aerospace/NED and Android coordinate systems but is the negative of the yaw angle for the Windows 8 coordinate system.

3 Sensor Simulation C Function

3.1 Compile time constants

The function `fSixDOFSensorDrivers` references two hardware specific constants used to derive the accelerometer and magnetometer readings as 16-bit integers from the floating point values in units of μT and g.

```
// sensor specific: MAG3110 and FXOS8700 provide 10 counts / uT
#define FCOUNTSPERUT 10.0F
// sensor specific: MMA8451 and FXOS8700 provide 4096 counts / g in 2g mode
#define FCOUNTSPERG 4096.0F
```

3.2 Data structures

The `SimulationModel` data structure contains details of the hard and soft iron disturbance model, the local geomagnetic field and the current orientation Euler angles.

```
// simulation model structure definition
struct SimulationModel
{
  // hard iron offset
  float fVx;           // simulated x component of hard iron
```

```
  float fVy;           // simulated y component of hard iron
  float fVz;           // simulated z component of hard iron
  // soft iron matrix
  float *fW[3];
  // geomagnetic field
  float fB;            // simulated geomagnetic field strength (uT)
  float fDeltaDeg;     // simulated geomagnetic inclination angle (deg)
  // orientation Euler angles
  float fPhiDeg;       // simulated roll angle (deg)
  float fTheDeg;       // simulated pitch angle (deg)
  float fPsiDeg;       // simulated yaw angle (deg)
  float fRhoDeg;       // simulated compass angle (deg)
};
```

The `AccelSensor` data structure contains the simulated accelerometer sensor output in integer counts and floating point *g*.

```
// accelerometer sensor structure definition
struct AccelSensor
{
  int16 iGpx;          // accelerometer sensor output x (counts)
  int16 iGpy;          // accelerometer sensor output y (counts)
  int16 iGpz;          // accelerometer sensor output z (counts)
  float fGpx;          // accelerometer sensor output x (g)
  float fGpy;          // accelerometer sensor output y (g)
  float fGpz;          // accelerometer sensor output z (g)
};
```

The `MagSensor` data structure contains the simulated magnetometer sensor output in integer counts and floating point µT.

```
// magnetometer sensor structure definition
struct MagSensor
{
  int16 iBpx;          // magnetometer sensor output x (counts)
  int16 iBpy;          // magnetometer sensor output y (counts)
  int16 iBpz;          // magnetometer sensor output z (counts)
  float fBpx;          // magnetometer sensor output x (uT)
  float fBpy;          // magnetometer sensor output y (uT)
  float fBpz;          // magnetometer sensor output z (uT)
};
```

3.3 Simulation function

```
void fSixDOFSensorDrivers(int32 iCoordSystem, struct SimulationModel *pthisSimulation,
  struct AccelSensor *pthisAccel, struct MagSensor *pthisMag)
{
  // this function simulates the accelerometer and magnetometer sensor outputs
  // for random Euler angles (roll, pitch and yaw) under the NED (Aerospace),
  // Android and Win8 coordinate systems for the purpose of validating the eCompass
  // and magnetic calibration software independent of physical sensors.

  // the accelerometer data is simulated in floating point and integer counts into
  // pthisAccel->fGpx, pthisAccel->fGpy, pthisAccel->fGpz and
  // pthisAccel->iGpx, pthisAccel->iGpy, pthisAccel->iGpz

  // the magnetometer data is simulated in floating point and integer counts into
  // pthisMag->fBpx, pthisMag->fBpy, pthisMag->fBpz and
  // pthisMag->iBpx, pthisMag->iBpy, pthisMag->iBpz

  // when confident that the eCompass and calibration algorithms are working
  // on simulated data, replace this function with your custom driver for the
  // specific sensors used and populate the same six elements of pthisAccel and
  // six elements of pthisMag.

  // since the native units of the accelerometer and magnetometer are integer
  // counts read over I2C or other bus, the integer members of pthisAccel and
```

```
  // pthisMag should be populated with the 16 bit sensor data and then the
  // floating point members computed from the integer data by scaling. the float
  // and integer members will be related by FCOUNTSPERG and FCOUNTSPERUT.

  // since this is a sensor simulation function, it is more convenient to compute
  // the floating point sensor values first and then cast these to 16 bit integer
  // values.

  // it is vital that the x, y and z outputs from the real sensors are correctly
  // aligned with the NED, Android or Windows 8 coordinate system selected. the
  // raw x, y and z sensor data read from the accelerometer and magnetometer
  // sensors are in the sensor coordinate frame. the sensors frames of reference
  // will in general be rotated relative to each other and to the coordinate
  // system of the final product as a result of PCB layout decisions taken.

  // local variables
  float sinPhi, cosPhi;      // sine and cosine of simulated roll angle Phi
  float sinThe, cosThe;      // sine and cosine of simulated pitch angle The
  float sinPsi, cosPsi;      // sine and cosine of simulated yaw angle Psi
  float sinDelta, cosDelta;    // sine and cosine of simulated inclination angle Delta
  float ftmpx, ftmpy, ftmpz;    // scratch variables

  // calculate random orientation Euler angles (deg)
  if (iCoordSystem == NED)
  {
    // -180 <= phi < 180 deg, -90 <= the < 90 deg, 0 <= psi < 360 deg
    pthisSimulation->fPhiDeg = (float)(((float) rand() / (RAND_MAX + 1) - 0.5F) * 360.0F);
    pthisSimulation->fTheDeg = (float)(((float) rand() / (RAND_MAX + 1) - 0.5F) * 180.0F);
    pthisSimulation->fPsiDeg = (float)((float) rand() / (RAND_MAX + 1) * 360.0F);
    pthisSimulation->fRhoDeg = pthisSimulation->fPsiDeg;
  }
  else if (iCoordSystem == ANDROID)
 {
   // -90 <= phi < 90 deg, -180 <= the < 180 deg, 0 <= psi < 360 deg
   pthisSimulation->fPhiDeg = (float)(((float) rand() / (RAND_MAX + 1) - 0.5F) * 180.0F);
   pthisSimulation->fTheDeg = (float)(((float) rand() / (RAND_MAX + 1) - 0.5F) *
360.0F);
   pthisSimulation->fPsiDeg = (float)((float) rand() / (RAND_MAX + 1) * 360.0F);
   pthisSimulation->fRhoDeg = pthisSimulation->fPsiDeg;
 }
  else // Win 8
 {
   // -90 <= phi < 90 deg, -180 <= the <180 deg, 0 <= psi < 360 deg
   pthisSimulation->fPhiDeg = (float)(((float) rand() / (RAND_MAX + 1) - 0.5F) * 180.0F);
   pthisSimulation->fTheDeg = (float)(((float) rand() / (RAND_MAX + 1) - 0.5F) *
360.0F);
   pthisSimulation->fPsiDeg = (float)((float) rand() / (RAND_MAX + 1) * 360.0F);
   pthisSimulation->fRhoDeg = 360.0F - pthisSimulation->fPsiDeg;
   if (pthisSimulation->fRhoDeg == 360.0F)
     pthisSimulation->fRhoDeg = 0.0F;
 }

  // compute sines and cosines of the Euler angles
  sinPhi = (float) sin(pthisSimulation->fPhiDeg * FDEGTORAD);
  cosPhi = (float) cos(pthisSimulation->fPhiDeg * FDEGTORAD);
  sinThe = (float) sin(pthisSimulation->fTheDeg * FDEGTORAD);
  cosThe = (float) cos(pthisSimulation->fTheDeg * FDEGTORAD);
  sinPsi = (float) sin(pthisSimulation->fPsiDeg * FDEGTORAD);
  cosPsi = (float) cos(pthisSimulation->fPsiDeg * FDEGTORAD);
  sinDelta = (float) sin(pthisSimulation->fDeltaDeg * FDEGTORAD);
  cosDelta = (float) cos(pthisSimulation->fDeltaDeg * FDEGTORAD);

  // calculate the accelerometer output in units of g
  if (iCoordSystem == NED)
  {
    pthisAccel->fGpx = -sinThe;
    pthisAccel->fGpy = cosThe * sinPhi;
    pthisAccel->fGpz = cosThe * cosPhi;
  }
  else if (iCoordSystem == ANDROID)
```

```
  {
    pthisAccel->fGpx = sinPhi;
    pthisAccel->fGpy = -cosPhi * sinThe;
    pthisAccel->fGpz = cosPhi * cosThe;
  }
  else // Win8
  {
    pthisAccel->fGpx = cosThe * sinPhi;
    pthisAccel->fGpy = -sinThe;
    pthisAccel->fGpz = -cosThe * cosPhi;
  }

  // convert the accelerometer sensor reading into bit counts
  pthisAccel->iGpx = (int16) (pthisAccel->fGpx * FCOUNTSPERG);
  pthisAccel->iGpy = (int16) (pthisAccel->fGpy * FCOUNTSPERG);
  pthisAccel->iGpz = (int16) (pthisAccel->fGpz * FCOUNTSPERG);

  // calculate rotated geomagnetic component in units of uT
  if (iCoordSystem == NED)
  {
    ftmpx = pthisSimulation->fB * (cosThe * cosPsi * cosDelta - sinThe * sinDelta);
    ftmpy = pthisSimulation->fB * ((cosPsi * sinThe * sinPhi - cosPhi * sinPsi) * cosDelta +
      cosThe * sinPhi * sinDelta);
    ftmpz = pthisSimulation->fB * ((cosPsi * sinThe * cosPhi + sinPhi * sinPsi) * cosDelta +
      cosThe * cosPhi * sinDelta);
  }
  else if (iCoordSystem == ANDROID)
  {
   ftmpx = pthisSimulation->fB * (-sinDelta * sinPhi - cosDelta * cosPhi * sinPsi);
   ftmpy = pthisSimulation->fB * (sinDelta * cosPhi * sinThe + cosDelta * (cosPsi * cosThe -
     sinPhi * sinPsi * sinThe));
   ftmpz = pthisSimulation->fB * (-sinDelta * cosPhi * cosThe + cosDelta *
     (cosThe * sinPhi * sinPsi + cosPsi * sinThe));
  }
  else // Win 8
  {
   ftmpx = pthisSimulation->fB * ((cosPsi * sinThe * sinPhi + cosPhi * sinPsi) * cosDelta +
     cosThe * sinPhi * sinDelta);
   ftmpy = pthisSimulation->fB * (cosThe * cosPsi * cosDelta - sinThe * sinDelta);
   ftmpz = pthisSimulation->fB * ((-cosPsi * sinThe * cosPhi + sinPhi * sinPsi) * cosDelta -
     cosThe * cosPhi * sinDelta);
  }

  // apply the forward soft iron matrix and hard iron vector
  pthisMag->fBpx = pthisSimulation->fW[0][0] * ftmpx + pthisSimulation->fW[0][1] * ftmpy +
    pthisSimulation->fW[0][2] * ftmpz + pthisSimulation->fVx;
  pthisMag->fBpy = pthisSimulation->fW[1][0] * ftmpx + pthisSimulation->fW[1][1] * ftmpy +
    pthisSimulation->fW[1][2] * ftmpz + pthisSimulation->fVy;
  pthisMag->fBpz = pthisSimulation->fW[2][0] * ftmpx + pthisSimulation->fW[2][1] * ftmpy +
    pthisSimulation->fW[2][2] * ftmpz + pthisSimulation->fVz;

  // get magnetometer sensor reading iBpxyz in bit counts
  pthisMag->iBpx = (int16) (pthisMag->fBpx * FCOUNTSPERUT);
  pthisMag->iBpy = (int16) (pthisMag->fBpy * FCOUNTSPERUT);
  pthisMag->iBpz = (int16) (pthisMag->fBpz * FCOUNTSPERUT);

  return;
}
```

4 Drivers for Real Sensors

4.1 Summary

The drivers for real accelerometer and magnetometer sensors will replace the simulated driver once there is confidence that the ecompass and calibration software is running on the target processor. The real driver will be somewhat simpler since there is:

- No need to simulate orientations: the physical movement of the smartphone performs this.
- No need to simulate the accelerometer nor magnetometer sensor readings since this is replaced by simply reading the sensor output.
- No need to simulate hard and soft iron interference: the components on the PCB add this effect

The components of the real driver will therefore:

1. Read the raw the *x*, *y*, and *z* channel 16 bit accelerometer and magnetometer data over serial bus.
2. Apply a hardware abstraction layer (HAL) mapping to map these raw integer count readings onto the coordinate system being used (Aerospace, Android or Windows 8).
3. Convert the integer count accelerometer readings to floating point units in g and the magnetometer readings to floating point units of µT. The precise scaling constants depend on the sensors being used and their selected range.
4. Place this data in the data structures `pthisAccel` and `pthisMag`.

Step 1 is dependent on the sensors being used and so it is impossible to comment further except to note that typically to 8-bit reads will be required to obtain the 16-bit data in each channel and a union structure or similar approach used to place these two bytes into the 16-bit short integer operand.

Steps 3 and 4 are trivial.

Step 2 is discussed in the next sections. It is far simpler to perform the HAL mapping in software rather than attempt to force the PCB layout to align the channels on each sensor to each other and to the coordinate system being used. The HAL alignment should be no more complex than swapping a couple of channels and perhaps negating one or two. The HAL mappings for accelerometer and magnetometer need only be determined once for each circuit board and should then be fixed.

4.2 Accelerometer HAL mapping

4.2.1 Aerospace/NED

Refer to the Aerospace/NED coordinate system shown in Figure 1 for definitions of the *x*, *y* and *z* axes.

Place the smartphone with the display side facing upwards and the *z* axis pointing downwards. The accelerometer *z* axis should read approximately +1g.

Rotate the smartphone so that the *y* axis (right hand side) points downwards. The accelerometer *y* axis should read approximately +1g

Rotate the smartphone so that the *x* axis is pointing downwards. The accelerometer *x* axis should read approximately +1g.

In general, this will not be the case. Simply exchange and negate the accelerometer channels as needed until the channels are correctly aligned.

4.2.2 Android

Refer to the Android coordinate system shown in Figure 2 for definitions of the *x*, *y* and *z* axes. The Android coordinate system is an acceleration-positive system so it is simplest to explain the accelerometer alignment with channels aligned upwards which is against gravity but aligned with the equivalent acceleration in the opposite direction to gravity.

Place the smartphone with the display side and *z* axis facing upwards. The accelerometer *z* axis should read approximately +1g.

Rotate the smartphone so that the y axis points upwards. The accelerometer *y* axis should read approximately +1g

Rotate the smartphone so that the x axis (right hand side) is pointing upwards. The accelerometer *x* axis should read approximately +1g.

In general, this will not be the case. Simply exchange and negate the accelerometer channels as needed until the channels are correctly aligned.

4.2.3 Windows 8

Refer to the Windows 8 coordinate system shown in Figure 3 for definitions of the *x*, *y* and *z* axes. The Windows 8 coordinate system is a gravity-positive system so it is simplest to explain the accelerometer alignment with channels aligned downwards and aligned with gravity

Invert the smartphone so that the display side and *z* axis are facing downwards. The accelerometer *z* axis should read approximately +1g.

Rotate the smartphone so that the *y* axis points downwards. The accelerometer *y* axis should read approximately +1g

Rotate the smartphone so that the *x* axis (right hand side) is pointing downwards. The accelerometer *x* axis should read approximately +1g.

In general, this will not be the case. Simply exchange and negate the accelerometer channels as needed until the channels are correctly aligned.

4.3 Magnetometer HAL mapping

4.3.1 Aerospace/NED, Android and Windows 8

All three coordinate systems can be treated together for the magnetometer HAL alignment since there is no equivalent to the acceleration versus gravity ambiguity which complicates accelerometer alignment. The instructions here are for use in the northern hemisphere where the inclination angle is δ positive and the geomagnetic field lines point downwards and northwards. The magnetometer HAL alignment should be performed well away from magnets, steel filing cabinets or similar sources of magnetic disturbance.

Use Figure 1 to Figure 3 as reference for the coordinate system selected. Place the product flat with the z axis of the product pointed downwards (aligned with the downwards component of the geomagnetic field) and with the *x* axis pointing approximately to magnetic north and record the unaligned *x*, *y*, and *z* magnetometer readings. Rotate the product by 180º about the *x* axis so that the *z* axis is now pointing upwards (aligned against the downwards component of the geomagnetic field) and again record the unaligned *x*, *y*, and *z* magnetometer readings. If correctly aligned, the *z* axis reading should undergo a decrease of approximately 100 µT during the rotation. The *x* and *y* axes should change little since the *x* axis remains pointing northwards and since the *y* axis changes from east to west or vice-versa and, by definition, the eastwards component of the magnetic field is approximately zero.

Repeat with the product's y axis pointing downwards with either the *x* or *z* axis pointing northwards. Record the *x*, *y*, and *z* magnetometer readings and then rotate the product 180º about the northwards axis. If correctly aligned, the *y* axis reading should decrease by approximately 100 µT during the rotation and the *x* and *z* axis readings should remain relatively unchanged.

Finally repeat with the product's *x* axis pointing downwards and with either the *y* or *z* axis pointing northwards. Record the *x*, *y*, and *z* magnetometer readings and then rotate the product 180º about the northwards axis. If correctly aligned, the *x* axis reading should decrease by approximately 100 µT during the rotation and the *y* and *z* axis readings should remain relatively unchanged.

Swap and negate the sensor *x*, *y*, *z* readings as required until the magnetometer sensor axes are aligned with the product axes in the Aerospace/NED, Android or Windows 8 coordinate system required.

The instructions should be reversed for HAL alignment in the southern hemisphere where the geomagnetic field lines point upwards and northwards. The reading in each channel should now be larger when each axis is pointed upwards compared to when pointed downwards.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4696
Rev. 1.0, 4/2013