Freescale Semiconductor
Application Note

Document Number:AN4685
Rev. 1.0, 4/2013

Tilt-Compensated eCompass in Aerospace, Android and Windows 8 Coordinate Systems

by: Mark Pedley

Contents

1 Introduction

This Application Note documents the tilt-compensated eCompass functions in Freescale's Xtrinsic eCompass and magnetic calibration software provided under license 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 Mathematical glossary

B	Geomagnetic field strength
$\boldsymbol{B}_{NED,c}, \boldsymbol{B}_{Android,c}, \boldsymbol{B}_{Win8,c}$	Calibrated magnetometer readings under Aerospace/NED, Android and Windows 8
$\boldsymbol{G}_{NED,p}, \boldsymbol{G}_{Android,p}, \boldsymbol{G}_{Win8,p}$	Accelerometer readings under Aerospace/ NED, Android and Windows 8
$\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z$	Rotation matrices about x, y and z axes
$\boldsymbol{R}_{NED}(\phi, \theta, \psi)$	Rotation matrix in the Aerospace/NED coordinate system
$\boldsymbol{R}_{Android}(\theta, \phi, \psi)$	Rotation matrix in the Android coordinate system
$\boldsymbol{R}_{Win8}(\phi, \theta, \psi)$	Rotation matrix in the Windows 8 coordinate system
δ	Geomagnetic inclination angle

Table continues on the next page...

ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle

2 Aerospace/NED Coordinate System

2.1 Rotation matrix

The Aerospace/NED rotation matrix $\boldsymbol{R}_{NED}(\phi,\theta,\psi)$ for rotation of the smartphone coordinate system by yaw angle ψ about the z axis, then pitch angle θ about the y axis and finally by roll angle ϕ about the x axis is:

$$\boldsymbol{R}_{NED}(\phi,\theta,\psi)=\boldsymbol{R}_{x}(\phi)\boldsymbol{R}_{\boldsymbol{y}}(\theta)\boldsymbol{R}_{\boldsymbol{z}}(\psi)=\begin{pmatrix}1&0&0\\0&cos\phi&sin\phi\\0&-sin\phi&cos\phi\end{pmatrix}\begin{pmatrix}cos\theta&0&-sin\theta\\0&1&0\\sin\theta&0&cos\theta\end{pmatrix}\begin{pmatrix}cos\psi&sin\psi&0\\-sin\psi&cos\psi&0\\0&0&1\end{pmatrix} \quad (1)$$

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

2.2 Ranges of Euler angles

By direct evaluation, the matrix $\boldsymbol{R}_{NED}(\phi+\pi,\pi-\theta,\psi+\pi)$ is equal to $\boldsymbol{R}_{NED}(\phi,\theta,\psi)$:

$$\boldsymbol{R}_{NED}(\phi+\pi,\pi-\theta,\psi+\pi)$$

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

$$=\begin{pmatrix}cos\,\theta\,cos\,\psi & cos\,\theta\,sin\psi & -sin\theta\\ cos\,\psi\,sin\theta sin\phi-cos\,\phi\,sin\psi & cos\,\phi\,cos\,\psi+sin\theta sin\phi sin\psi & cos\,\theta\,sin\phi\\ cos\phi\,cos\,\psi\,sin\theta+sin\phi sin\psi & cos\,\phi\,sin\theta sin\psi-cos\,\psi\,sin\phi & cos\,\theta\,cos\phi\end{pmatrix} \quad (4)$$

$$\Rightarrow \boldsymbol{R}_{NED}(\phi+\pi,\pi-\theta,\psi+\pi)=\boldsymbol{R}_{NED}(\phi,\theta,\psi) \quad (5)$$

As a consequence, there are, in general, two solutions for the Euler angles for any Aerospace/NED rotation matrix if the Euler angles are allowed to have values over the full 360º range. The Aerospace/NED coordinate system therefore restricts the pitch angle θ to the range –90º ≤ θ < 90º deg to remove one of these two solutions. The permitted ranges of the Euler angles in the Aerospace/NED coordinate system are:

$$-180\ deg \le \phi < 180\ deg \quad (6)$$

$$-90\ deg \le \theta < 90\ deg \quad (7)$$

$$0\ deg \le \psi < 360\ deg \quad (8)$$

Figure 1, Figure 2, and Figure 3 plot the Euler angles in the Aerospace/NED coordinate system as the smartphone is rotated through 360º in roll, pitch and yaw respectively with, in each case, the other two Euler angles set to zero. The roll ϕ and yaw ψ plots are smooth, modulo 360º, since –180º is equivalent to +180º and since 0º is equivalent to 360º. The pitch angle plot, however, shows the discontinuities in roll and yaw angles at –90º and 90º pitch angles as defined by equation **(5)** .

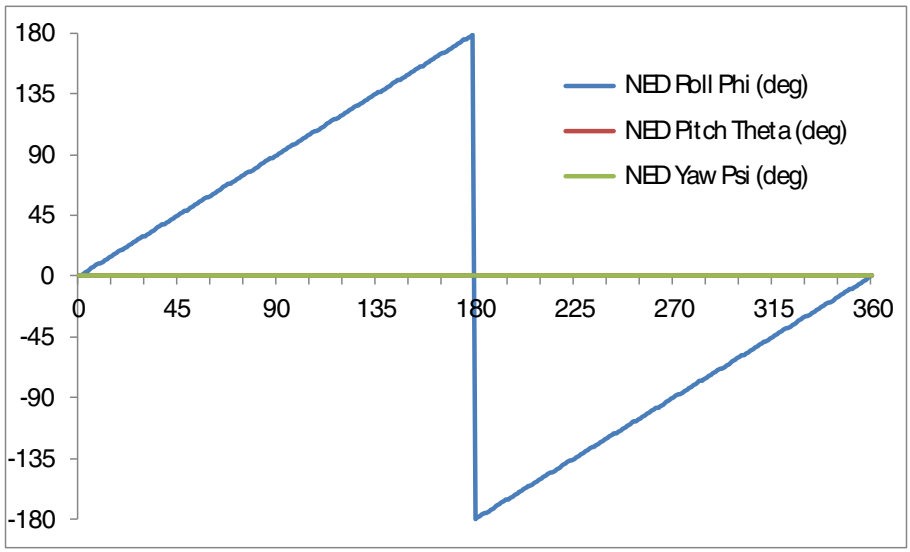

Figure 1. 360º roll rotation at zero pitch and yaw in the Aerospace/NED coordinate system

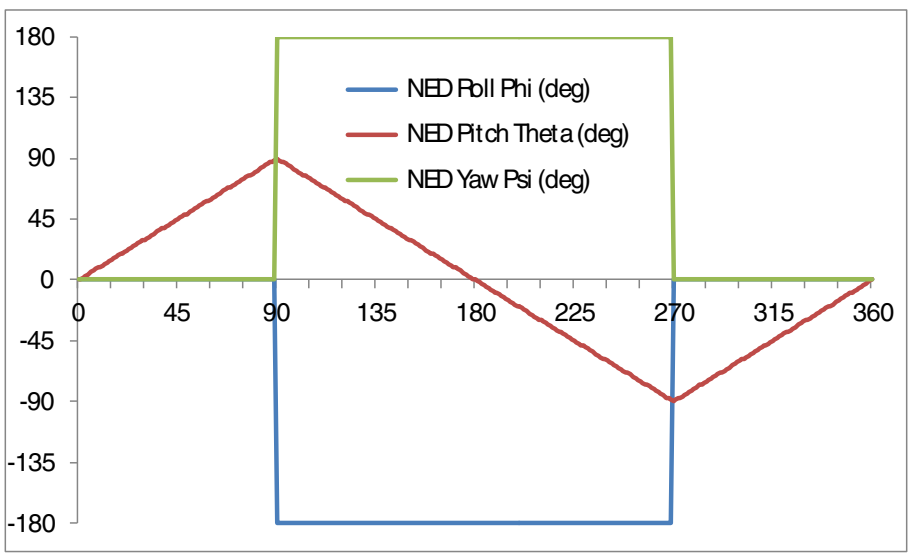

Figure 2. 360º pitch rotation at zero roll and yaw in the Aerospace/NED coordinate system

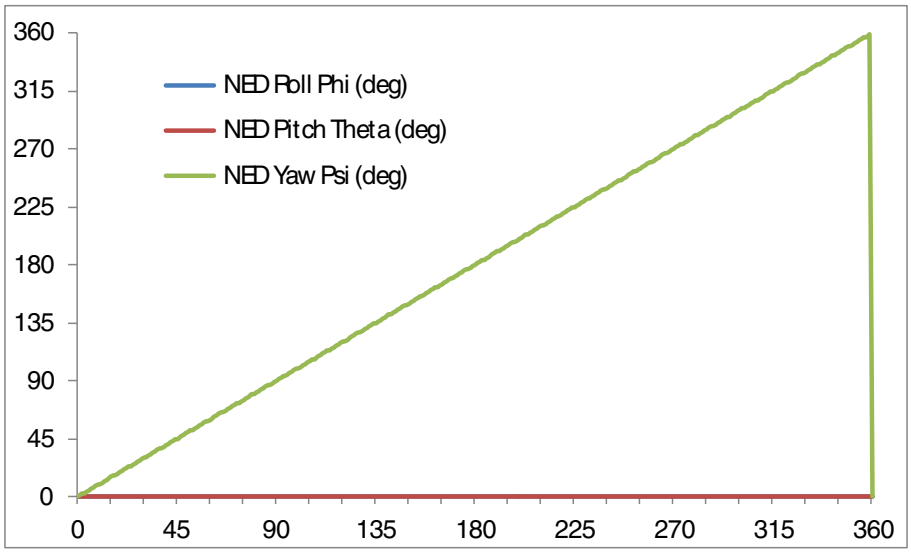

Figure 3. 360° yaw rotation at zero roll and pitch in the Aerospace/NED coordinate system

2.3 Sensor measurements

The accelerometer reading $\boldsymbol{G}_{NED,p}$, in units of g, after rotation of the smartphone from its initial position (flat and pointing to magnetic north) by angles θ in pitch and ϕ in roll is:

$$\boldsymbol{G}_{NED,p} = \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \boldsymbol{R}_{NED} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \boldsymbol{R}_{\boldsymbol{x}}(\phi)\boldsymbol{R}_{\boldsymbol{y}}(\theta)\boldsymbol{R}_{\boldsymbol{z}}(\psi) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{9}$$

$$= \begin{pmatrix} cos\,\theta\, cos\,\psi & cos\,\theta\, sin\psi & -sin\theta \\ cos\,\psi\, sin\theta sin\phi - cos\,\phi\, sin\psi & cos\,\phi\, cos\,\psi + sin\theta sin\phi sin\psi & cos\,\theta\, sin\phi \\ cos\phi\, cos\,\psi\, sin\theta + sin\phi sin\psi & cos\,\phi\, sin\theta sin\psi - cos\,\psi\, sin\phi & cos\,\theta\, cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -sin\theta \\ cos\theta sin\phi \\ cos\theta cos\phi \end{pmatrix} \tag{10}$$

The magnetometer reading, after removal of hard and soft iron distortion and after rotation by angles ψ in yaw, θ in pitch and ϕ in roll, is:

$$\boldsymbol{B}_{NED,c} = \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \boldsymbol{R}_{NED} B \begin{pmatrix} cos\delta \\ 0 \\ sin\delta \end{pmatrix} = \boldsymbol{R}_{\boldsymbol{x}}(\phi)\boldsymbol{R}_{\boldsymbol{y}}(\theta)\boldsymbol{R}_{\boldsymbol{z}}(\psi) B \begin{pmatrix} cos\delta \\ 0 \\ sin\delta \end{pmatrix} \tag{11}$$

$$= \begin{pmatrix} cos\,\theta\, cos\,\psi & cos\,\theta\, sin\psi & -sin\theta \\ cos\,\psi\, sin\theta sin\phi - cos\,\phi\, sin\psi & cos\,\phi\, cos\,\psi + sin\theta sin\phi sin\psi & cos\,\theta\, sin\phi \\ cos\phi\, cos\,\psi\, sin\theta + sin\phi sin\psi & cos\,\phi\, sin\theta sin\psi - cos\,\psi\, sin\phi & cos\,\theta\, cos\phi \end{pmatrix} B \begin{pmatrix} cos\delta \\ 0 \\ sin\delta \end{pmatrix} \tag{12}$$

$$= B \begin{pmatrix} cos\,\theta\, cos\,\psi\, cos\delta - sin\theta sin\delta \\ (cos\,\psi\, sin\theta sin\phi - cos\,\phi\, sin\psi) cos\delta + cos\,\theta\, sin\phi sin\delta \\ (cos\,\psi\, sin\theta cos\phi + sin\phi sin\psi) cos\delta + cos\,\theta\, cos\phi sin\delta \end{pmatrix} \tag{13}$$

2.4 Solution for Euler angles and rotation matrix

The function of the Aerospace/NED eCompass function is to compute the rotation matrix, Euler angles and quaternion from the accelerometer and calibrated magnetometer measurements. This is most elegantly done by computing the rotation matrix directly from the measurements and then computing the Euler angles and quaternion from the rotation matrix using the functions documented in Application Note AN4676.

Comparison of equations (2) and (10) shows that the third *z* column of the Aerospace/NED rotation matrix is simply the normalized accelerometer reading:

$$\begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} = \frac{1}{|\boldsymbol{G}_{NED,p}|}\begin{pmatrix} G_{NED,px} \\ G_{NED,py} \\ G_{NED,pz} \end{pmatrix} \tag{14}$$

Simple algebra on equations (2), (10) and (12) gives the expression for the second *y* column of the rotation matrix to be the normalized vector product of the accelerometer and calibrated magnetometer sensor readings:

$$\begin{pmatrix} R_{01} \\ R_{11} \\ R_{21} \end{pmatrix} = \frac{\begin{pmatrix} G_{NED,px} \\ G_{NED,py} \\ G_{NED,pz} \end{pmatrix} \times \begin{pmatrix} B_{NED,cx} \\ B_{NED,cy} \\ B_{NED,cz} \end{pmatrix}}{\left|\begin{pmatrix} G_{NED,px} \\ G_{NED,py} \\ G_{NED,pz} \end{pmatrix} \times \begin{pmatrix} B_{NED,cx} \\ B_{NED,cy} \\ B_{NED,cz} \end{pmatrix}\right|} \tag{15}$$

Finally, since any rotation matrix has orthonormal column vectors, the first *x* column of the rotation matrix is simply the normalized vector product of the *y* and *z* columns:

$$\begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix} = \frac{\begin{pmatrix} R_{01} \\ R_{11} \\ R_{21} \end{pmatrix} \times \begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix}}{\left|\begin{pmatrix} R_{01} \\ R_{11} \\ R_{21} \end{pmatrix} \times \begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix}\right|} \tag{16}$$

2.5 Inclination angle

The scalar product of the accelerometer and magnetometer vectors is invariant under rotation giving a simple expression for the magnetic inclination angle δ from measurements in the Aerospace/NED coordinate system.

$$\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} \cdot \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdot B \begin{pmatrix} cos\delta \\ 0 \\ sin\delta \end{pmatrix} = Bsin\delta \Longrightarrow sin\delta = \frac{G_{px}B_{cx} + G_{py}B_{cy} + G_{pz}B_{cz}}{|\boldsymbol{G}_p||\boldsymbol{B}_c|} \tag{17}$$

2.6 Gimbal lock

The rotation matrix generally has three degrees of freedom through being a function of the three independent rotations in roll, pitch and yaw. The phenomenon of gimbal lock occurs when the second of the sequence of the three Euler angle rotation matrices happens to align the rotation axes of the first and third rotation matrices so decreasing the number of degrees of freedom from three to two. When gimbal lock occurs, there is no unique solution for all three Euler angles and two of the Euler angles can oscillate uncontrollably.

It must be emphasized that gimbal lock is a mathematical instability afflicting the Euler angle representation of orientation. Rotation matrices, rotation quaternion and rotation vector representations of orientation do not experience any instability at gimbal lock orientations.

Gimbal lock occurs in the Aerospace/NED coordinate system at pitch angles θ of –90º and +90º. At these two pitch angles θ, the final rotation in roll angle ϕ is about the same vertical axis as the initial yaw rotation angle ψ.

At θ=–90º, the Aerospace/NED rotation matrix simplifies to a function of the sum $\phi + \psi$ indicating that only the sum of the roll and yaw angles can be determined.

$$\boldsymbol{R}_{NED}(\phi, \theta = -90deg, \psi) = \begin{pmatrix} 0 & 0 & 1 \\ -cos\,\psi\, sin\phi - cos\,\phi\, sin\psi & cos\,\phi\, cos\,\psi - sin\phi sin\psi & 0 \\ -cos\phi\, cos\,\psi + sin\phi sin\psi & -cos\,\phi\, sin\psi - cos\,\psi\, sin\phi & 0 \end{pmatrix} \quad \textbf{(18)}$$

$$= \begin{pmatrix} 0 & 0 & 1 \\ -sin(\phi + \psi) & cos(\phi + \psi) & 0 \\ -cos(\phi + \psi) & -sin(\phi + \psi) & 0 \end{pmatrix} \quad \textbf{(19)}$$

Similarly, at θ=90º, the Aerospace/NED rotation matrix simplifies to a function of the difference $\phi - \psi$ indicating that only the difference of the roll and yaw angles can be determined.

$$\boldsymbol{R}_{NED}(\phi, \theta = 90deg, \psi) = \begin{pmatrix} 0 & 0 & -1 \\ cos\,\psi\, sin\phi - cos\,\phi\, sin\psi & cos\,\phi\, cos\,\psi + sin\phi sin\psi & 0 \\ cos\phi\, cos\,\psi + sin\phi sin\psi & cos\,\phi\, sin\psi - cos\,\psi\, sin\phi & 0 \end{pmatrix} \quad \textbf{(20)}$$

$$= \begin{pmatrix} 0 & 0 & -1 \\ sin(\phi - \psi) & cos(\phi - \psi) & 0 \\ cos(\phi - \psi) & -sin(\phi - \psi) & 0 \end{pmatrix} \quad \textbf{(21)}$$

2.7 eCompass C source code

```
// NED: Direct tilt-compensated e-Compass function
void feCompassDirectNED(struct SV6DOF *pthisSV6DOF, struct MagSensor *pthisMag,
struct AccelSensor *pthisAccel)
{
  // local variables
  float fmodB, fmodG, fmodx, fmody;
  float fsinDelta;

  // place the un-normalized gravity and geomagnetic vectors into
  // the rotation matrix z and x axes
  pthisSV6DOF->fR6DOFn[0][2] = pthisAccel->fGpx;
  pthisSV6DOF->fR6DOFn[1][2] = pthisAccel->fGpy;
  pthisSV6DOF->fR6DOFn[2][2] = pthisAccel->fGpz;
  pthisSV6DOF->fR6DOFn[0][0] = pthisMag->fBcx;
  pthisSV6DOF->fR6DOFn[1][0] = pthisMag->fBcy;
```

```
  pthisSV6DOF->fR6DOFn[2][0] = pthisMag->fBcz;

  // set y vector to vector product of z and x vectors
  pthisSV6DOF->fR6DOFn[0][1] = pthisSV6DOF->fR6DOFn[1][2] * pthisSV6DOF->fR6DOFn[2][0] -
    pthisSV6DOF->fR6DOFn[2][2] * pthisSV6DOF->fR6DOFn[1][0];
  pthisSV6DOF->fR6DOFn[1][1] = pthisSV6DOF->fR6DOFn[2][2] * pthisSV6DOF->fR6DOFn[0][0] -
    pthisSV6DOF->fR6DOFn[0][2] * pthisSV6DOF->fR6DOFn[2][0];
  pthisSV6DOF->fR6DOFn[2][1] = pthisSV6DOF->fR6DOFn[0][2] * pthisSV6DOF->fR6DOFn[1][0] -
    pthisSV6DOF->fR6DOFn[1][2] * pthisSV6DOF->fR6DOFn[0][0];

  // set x vector to vector product of y and z vectors
  pthisSV6DOF->fR6DOFn[0][0] = pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[2][2] -
    pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[1][2];
  pthisSV6DOF->fR6DOFn[1][0] = pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[0][2] -
    pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[2][2];
  pthisSV6DOF->fR6DOFn[2][0] = pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[1][2] -
    pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[0][2];

  // calculate the vector moduli
  fmodG =  (float) sqrt(pthisAccel->fGpx * pthisAccel->fGpx +
    pthisAccel->fGpy * pthisAccel->fGpy +
    pthisAccel->fGpz * pthisAccel->fGpz);
  fmodB =  (float) sqrt(pthisMag->fBcx * pthisMag->fBcx +
    pthisMag->fBcy * pthisMag->fBcy +
    pthisMag->fBcz * pthisMag->fBcz);
  fmodx = (float) sqrt(pthisSV6DOF->fR6DOFn[0][0] * pthisSV6DOF->fR6DOFn[0][0] +
        pthisSV6DOF->fR6DOFn[1][0] * pthisSV6DOF->fR6DOFn[1][0] +
        pthisSV6DOF->fR6DOFn[2][0] * pthisSV6DOF->fR6DOFn[2][0]);
  fmody = (float) sqrt(pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[0][1] +
        pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[1][1] +
        pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[2][1]);

  // normalize the rotation matrix
  if (!((fmodx == 0.0F) || (fmody == 0.0F) || (fmodG == 0.0F)))
  {
    // normalize x axis
    pthisSV6DOF->fR6DOFn[0][0] /= fmodx;
    pthisSV6DOF->fR6DOFn[1][0] /= fmodx;
    pthisSV6DOF->fR6DOFn[2][0] /= fmodx;
    // normalize y axis
    pthisSV6DOF->fR6DOFn[0][1] /= fmody;
    pthisSV6DOF->fR6DOFn[1][1] /= fmody;
    pthisSV6DOF->fR6DOFn[2][1] /= fmody;
    // normalize z axis
    pthisSV6DOF->fR6DOFn[0][2] /= fmodG;
    pthisSV6DOF->fR6DOFn[1][2] /= fmodG;
    pthisSV6DOF->fR6DOFn[2][2] /= fmodG;

    // estimate the declination angle Delta (90 minus angle between the vectors)
    fsinDelta = (pthisAccel->fGpx * pthisMag->fBcx + pthisAccel->fGpy * pthisMag->fBcy +
      pthisAccel->fGpz * pthisMag->fBcz) / fmodG / fmodB;
    pthisSV6DOF->fDelta6DOFn = (float) asin(fsinDelta) * FRADTODEG;
  }
  else
  // no solution is possible to set rotation to identity matrix and delta to 0 degrees
  {
    fmatrixAeqI(pthisSV6DOF->fR6DOFn, 3);
    pthisSV6DOF->fDelta6DOFn = 0.0F;
  }

  // extract the roll, pitch and yaw angles from the rotation matrix fR6DOFn
  fNEDAnglesDegFromRotationMatrix(pthisSV6DOF->fR6DOFn,
    &(pthisSV6DOF->fPhi6DOF), &(pthisSV6DOF->fThe6DOF),
    &(pthisSV6DOF->fPsi6DOF), &(pthisSV6DOF->fRho6DOF));

  // get the instantaneous orientation quaternion
  fQuaternionFromRotationMatrix(pthisSV6DOF->fR6DOFn, &(pthisSV6DOF->fq6DOFn));

  return;
}
```

3 Android Coordinate System

3.1 Rotation matrix

The Android rotation matrix $\boldsymbol{R}_{Android}(\theta,\phi,\psi)$ for rotation of the smartphone coordinate system by yaw angle ψ about the z axis, then roll angle ϕ about the y axis and finally by pitch angle θ about the x axis is:

$$\boldsymbol{R}_{Android}(\theta,\phi,\psi)=\boldsymbol{R}_{\boldsymbol{x}}(\theta)\boldsymbol{R}_{\boldsymbol{y}}(\phi)\boldsymbol{R}_{\boldsymbol{z}}(\psi)=\begin{pmatrix}1&0&0\\0&cos\,\theta&-sin\theta\\0&sin\theta&cos\theta\end{pmatrix}\begin{pmatrix}cos\,\phi&0&sin\phi\\0&1&0\\-sin\phi&0&cos\phi\end{pmatrix}\begin{pmatrix}cos\,\psi&-sin\psi&0\\sin\psi&cos\,\psi&0\\0&0&1\end{pmatrix} \tag{22}$$

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

3.2 Ranges of Euler angles

By direct evaluation, the matrix $\boldsymbol{R}_{Android}(\theta+\pi,\pi-\phi,\psi+\pi)$ is equal to $\boldsymbol{R}_{Android}(\theta,\phi,\psi)$:

$$\boldsymbol{R}_{Android}(\theta+\pi,\pi-\phi,\psi+\pi)$$

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

$$=\begin{pmatrix}cos\,\phi\,cos\,\psi&-cos\phi sin\psi&sin\phi\\cos\,\theta\,sin\psi+cos\,\psi\,sin\phi sin\theta&cos\,\psi\,cos\,\theta-sin\phi sin\psi sin\theta&-cos\,\phi\,sin\theta\\-cos\,\psi\,cos\,\theta\,sin\phi+sin\psi sin\theta&cos\,\theta\,sin\phi sin\psi+cos\,\psi\,sin\theta&cos\,\phi\,cos\,\theta\end{pmatrix} \tag{25}$$

$$\Rightarrow \boldsymbol{R}_{Android}(\theta+\pi,\pi-\phi,\psi+\pi)=\boldsymbol{R}_{Android}(\theta,\phi,\psi) \tag{26}$$

There are, therefore, two solutions for the Euler angles for any Android rotation matrix if the Euler angles are allowed to have values over the full 360º range. The Android coordinate system therefore restricts the roll angle ϕ to the range –90º ≤ ϕ < 90º deg to remove one of these two solutions. The permitted ranges of the Euler angles in the Android coordinate system are:

$$-90\ deg \le \phi < 90\ deg \tag{27}$$

$$-180\ deg \le \theta < 180\ deg \tag{28}$$

$$0\ deg \le \psi < 360\ deg \tag{29}$$

Figure 4, Figure 5, and Figure 6 plot the Euler angles in the Android coordinate system as the smartphone is rotated through 360º in roll, pitch and yaw respectively with, in each case, the other two Euler angles set to zero. The pitch θ and yaw ψ plots are smooth, modulo 360º, since –180º is equivalent to +180º and since 0º is equivalent to 360º. The roll angle plot, however, shows the discontinuities in pitch and yaw angles at –90º and 90º roll angles as defined by equation (26).

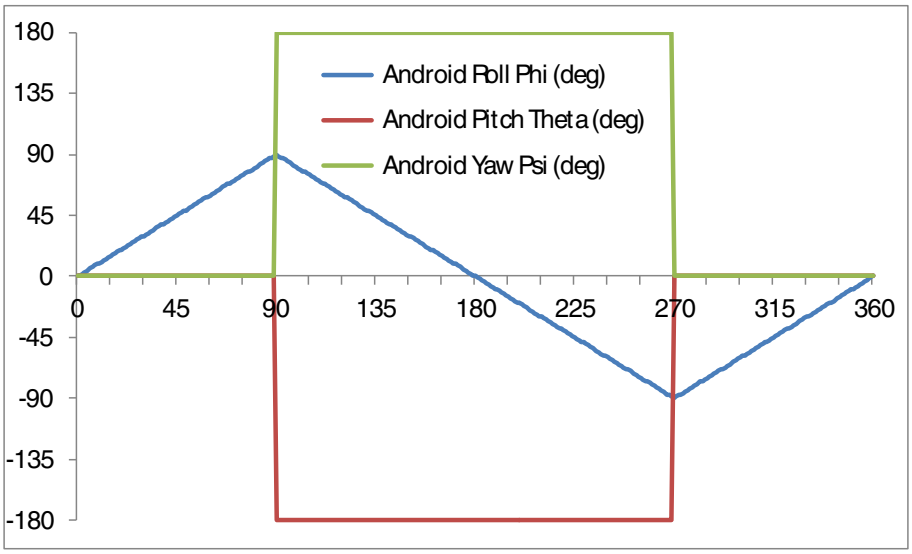

Figure 4. 360° roll rotation at zero pitch and yaw in the Android coordinate system

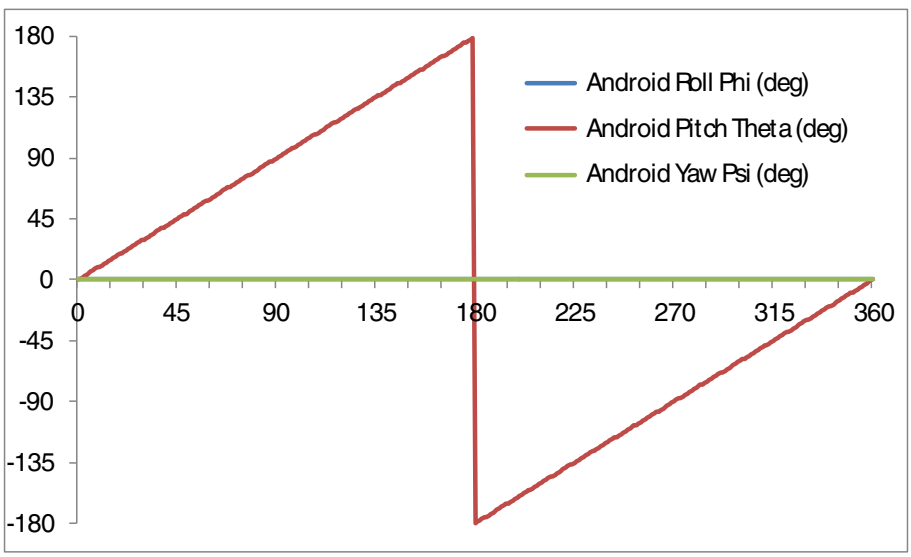

Figure 5. 360° pitch rotation at zero roll and yaw in the Android coordinate system

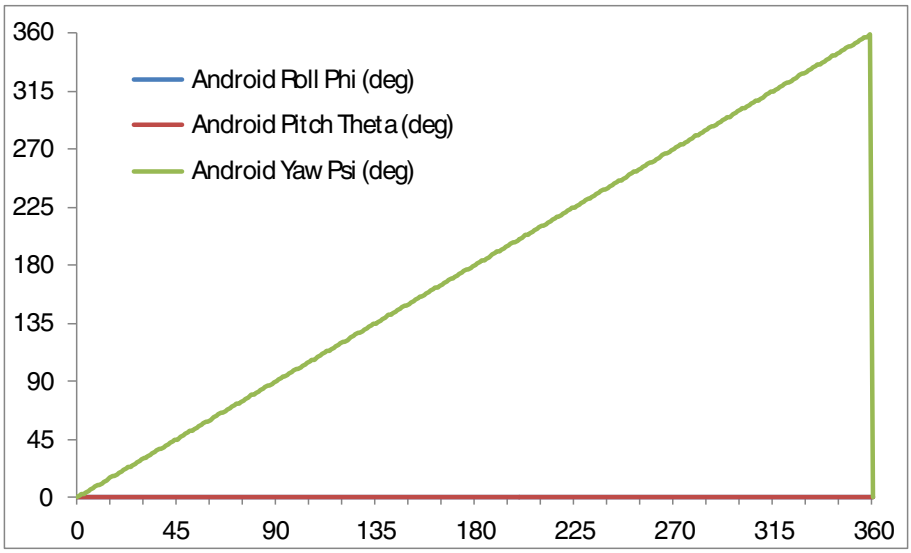

Figure 6. 360° yaw rotation at zero roll and pitch in the Android coordinate system

3.3 Sensor measurements

The accelerometer reading $\boldsymbol{G}_{Android,p}$, in units of g, after rotation of the smartphone from its initial position (flat and pointing to magnetic north) by angles θ in pitch and ϕ in roll is:

$$\boldsymbol{G}_{Android,p} = \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \boldsymbol{R}_{Android} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \boldsymbol{R}_{\boldsymbol{x}}(\theta)\boldsymbol{R}_{\boldsymbol{y}}(\phi)\boldsymbol{R}_{\boldsymbol{z}}(\psi) \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{30}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi & -cos\phi sin\psi & sin\phi \\ cos\,\theta\,sin\psi + cos\,\psi\,sin\phi sin\theta & cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta & -cos\,\phi\,sin\theta \\ -cos\,\psi\,cos\,\theta\,sin\phi + sin\psi sin\theta & cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} sin\phi \\ -cos\phi\,sin\,\theta \\ cos\phi\,cos\,\theta \end{pmatrix} \tag{31}$$

The magnetometer reading, after removal of hard and soft iron distortion and after rotation by angles ψ in yaw, ϕ in roll and θ in pitch, is:

$$\boldsymbol{B}_{Android,c} = \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \boldsymbol{R}_{Android} B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} = \boldsymbol{R}_{\boldsymbol{x}}(\theta)\boldsymbol{R}_{\boldsymbol{y}}(\phi)\boldsymbol{R}_{\boldsymbol{z}}(\psi) B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \tag{32}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi & -cos\phi sin\psi & sin\phi \\ cos\,\theta\,sin\psi + cos\,\psi\,sin\phi sin\theta & cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta & -cos\,\phi\,sin\theta \\ -cos\,\psi\,cos\,\theta\,sin\phi + sin\psi sin\theta & cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \tag{33}$$

$$= B \begin{pmatrix} -sin\delta sin\phi - cos\delta cos\phi sin\psi \\ sin\delta\,cos\,\phi\,sin\theta + cos\delta(cos\,\psi\,cos\,\theta - sin\phi sin\psi sin\theta) \\ -sin\delta\,cos\,\phi\,cos\,\theta + cos\delta(cos\,\theta\,sin\phi sin\psi + cos\,\psi\,sin\theta) \end{pmatrix} \tag{34}$$

3.4 Solution for Euler angles and rotation matrix

The rotation matrix, Euler angles and quaternion are computed in the Android coordinate system using a very similar approach to described in Solution for Euler angles and rotation matrix for the Aerospace/NED coordinate system. The rotation matrix is first computed directly from the accelerometer and magnetometer measurements and then the Euler angles and quaternion computed from the rotation matrix using the functions documented in Application Note AN4676.

Comparison of equations **(25)** and **(31)** shows that the third z column of the Android rotation matrix is simply the normalized accelerometer reading:

$$\begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} = \frac{1}{|\vec{G}_{Android,p}|} \begin{pmatrix} G_{Android,px} \\ G_{Android,py} \\ G_{Android,pz} \end{pmatrix} \tag{35}$$

Simple algebra on equations **(23)**, **(31)** and **(34)** gives the expression for the first x column of the rotation matrix to be the normalized vector product of the calibrated magnetometer and accelerometer sensor readings:

$$\begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix} = \frac{\begin{pmatrix} B_{Android,cx} \\ B_{Android,cy} \\ B_{Android,cz} \end{pmatrix} \times \begin{pmatrix} G_{Android,px} \\ G_{Android,py} \\ G_{Android,pz} \end{pmatrix}}{\left| \begin{pmatrix} B_{Android,cx} \\ B_{Android,cy} \\ B_{Android,cz} \end{pmatrix} \times \begin{pmatrix} G_{Android,px} \\ G_{Android,py} \\ G_{Android,pz} \end{pmatrix} \right|} \tag{36}$$

Finally, since any rotation matrix has orthonormal column vectors, the second y column of the rotation matrix is simply the normalized vector product of the z and x columns:

$$\begin{pmatrix} R_{01} \\ R_{11} \\ R_{21} \end{pmatrix} = \frac{\begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} \times \begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix}}{\left| \begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} \times \begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix} \right|} \tag{37}$$

3.5 Inclination angle

The scalar product of the accelerometer and magnetometer vectors is invariant under rotation giving an expression for the magnetic inclination angle δ in the Android coordinate system.

$$\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} \cdot \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \cdot B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} = -Bsin\delta \Longrightarrow sin\delta = \frac{-(G_{px}B_{cx} + G_{py}B_{cy} + G_{pz}B_{cz})}{|\boldsymbol{G}_p||\boldsymbol{B}_c|} \tag{38}$$

3.6 Gimbal lock

The second rotation in the sequence of three Euler angle rotations in the Android coordinate system is about the *y* axis by roll angle ϕ. Gimbal lock occurs in the Android coordinate system at roll angles ϕ of –90º and +90º when the final rotation in roll angle θ is about the same vertical axis as the initial yaw rotation angle ψ.

At ϕ=–90º, the Android rotation matrix simplifies to a function of the difference $\phi - \psi$ indicating that only the difference of the pitch and yaw angles can be determined.

$$\boldsymbol{R}_{Android}(\theta, \phi = -90deg, \psi) = \begin{pmatrix} 0 & 0 & -1 \\ cos\,\theta\, sin\psi - cos\,\psi\, sin\theta & cos\,\psi\, cos\,\theta + sin\psi sin\theta & 0 \\ cos\,\psi\, cos\,\theta + sin\psi sin\theta & -cos\,\theta\, sin\psi + cos\,\psi\, sin\theta & 0 \end{pmatrix} \quad (39)$$

$$= \begin{pmatrix} 0 & 0 & -1 \\ -sin(\theta - \psi) & cos(\theta - \psi) & 0 \\ cos(\theta - \psi) & sin(\theta - \psi) & 0 \end{pmatrix} \quad (40)$$

At ϕ=90º, the Android rotation matrix simplifies to a function of the sum $\phi + \psi$ indicating that only the sum of the pitch and yaw angles can be determined.

$$\boldsymbol{R}_{Android}(\theta, \phi = 90deg, \psi) = \begin{pmatrix} 0 & 0 & 1 \\ cos\,\theta\, sin\psi + cos\,\psi\, sin\theta & cos\,\psi\, cos\,\theta - sin\psi sin\theta & 0 \\ -cos\,\psi\, cos\,\theta + sin\psi sin\theta & cos\,\theta\, sin\psi + cos\,\psi\, sin\theta & 0 \end{pmatrix} \quad (41)$$

$$= \begin{pmatrix} 0 & 0 & 1 \\ sin(\theta + \psi) & cos(\theta + \psi) & 0 \\ -cos(\theta + \psi) & sin(\theta + \psi) & 0 \end{pmatrix} \quad (42)$$

3.7 eCompass C source code

```
// Android: Direct tilt-compensated e-Compass function
void feCompassDirectAndroid(struct SV6DOF *pthisSV6DOF, struct MagSensor *pthisMag,
  struct AccelSensor *pthisAccel)
{
  // local variables
  float fmodB, fmodG, fmodx, fmody;
  float fsinDelta;

  // place the un-normalized gravity and geomagnetic vectors into
  // the rotation matrix z and y axes
  pthisSV6DOF->fR6DOFn[0][2] = pthisAccel->fGpx;
  pthisSV6DOF->fR6DOFn[1][2] = pthisAccel->fGpy;
  pthisSV6DOF->fR6DOFn[2][2] = pthisAccel->fGpz;
  pthisSV6DOF->fR6DOFn[0][1] = pthisMag->fBcx;
  pthisSV6DOF->fR6DOFn[1][1] = pthisMag->fBcy;
  pthisSV6DOF->fR6DOFn[2][1] = pthisMag->fBcz;

  // set x vector to vector product of y and z vectors
  pthisSV6DOF->fR6DOFn[0][0] = pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[2][2] -
    pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[1][2];
  pthisSV6DOF->fR6DOFn[1][0] = pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[0][2] -
    pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[2][2];
  pthisSV6DOF->fR6DOFn[2][0] = pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[1][2] -
    pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[0][2];

  // set y vector to vector product of z and x vectors
  pthisSV6DOF->fR6DOFn[0][1] = pthisSV6DOF->fR6DOFn[1][2] * pthisSV6DOF->fR6DOFn[2][0] -
    pthisSV6DOF->fR6DOFn[2][2] * pthisSV6DOF->fR6DOFn[1][0];
```

```
  pthisSV6DOF->fR6DOFn[1][1] = pthisSV6DOF->fR6DOFn[2][2] * pthisSV6DOF->fR6DOFn[0][0] -
    pthisSV6DOF->fR6DOFn[0][2] * pthisSV6DOF->fR6DOFn[2][0];
  pthisSV6DOF->fR6DOFn[2][1] = pthisSV6DOF->fR6DOFn[0][2] * pthisSV6DOF->fR6DOFn[1][0] -
    pthisSV6DOF->fR6DOFn[1][2] * pthisSV6DOF->fR6DOFn[0][0];

  // calculate the vector moduli
  fmodG =  (float) sqrt(pthisAccel->fGpx * pthisAccel->fGpx +
    pthisAccel->fGpy * pthisAccel->fGpy +
    pthisAccel->fGpz * pthisAccel->fGpz);
  fmodB =  (float) sqrt(pthisMag->fBcx * pthisMag->fBcx +
    pthisMag->fBcy * pthisMag->fBcy +
    pthisMag->fBcz * pthisMag->fBcz);
  fmodx = (float) sqrt(pthisSV6DOF->fR6DOFn[0][0] * pthisSV6DOF->fR6DOFn[0][0] +
        pthisSV6DOF->fR6DOFn[1][0] * pthisSV6DOF->fR6DOFn[1][0] +
        pthisSV6DOF->fR6DOFn[2][0] * pthisSV6DOF->fR6DOFn[2][0]);
  fmody = (float) sqrt(pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[0][1] +
        pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[1][1] +
        pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[2][1]);

  // normalize the rotation matrix
  if (!((fmodx == 0.0F) || (fmody == 0.0F) || (fmodG == 0.0F)))
  {
    // normalize x axis
    pthisSV6DOF->fR6DOFn[0][0] /= fmodx;
    pthisSV6DOF->fR6DOFn[1][0] /= fmodx;
    pthisSV6DOF->fR6DOFn[2][0] /= fmodx;
    // normalize y axis
    pthisSV6DOF->fR6DOFn[0][1] /= fmody;
    pthisSV6DOF->fR6DOFn[1][1] /= fmody;
    pthisSV6DOF->fR6DOFn[2][1] /= fmody;
    // normalize z axis
    pthisSV6DOF->fR6DOFn[0][2] /= fmodG;
    pthisSV6DOF->fR6DOFn[1][2] /= fmodG;
    pthisSV6DOF->fR6DOFn[2][2] /= fmodG;

    // estimate the declination angle Delta (90 minus angle between the vectors)
    fsinDelta = -(pthisAccel->fGpx * pthisMag->fBcx + pthisAccel->fGpy * pthisMag->fBcy +
      pthisAccel->fGpz * pthisMag->fBcz) / fmodG / fmodB;
    pthisSV6DOF->fDelta6DOFn = (float) asin(fsinDelta) * FRADTODEG;
  }
  else
  // no solution is possible to set rotation to identity matrix and delta to 0 degrees
  {
    fmatrixAeqI(pthisSV6DOF->fR6DOFn, 3);
    pthisSV6DOF->fDelta6DOFn = 0.0F;
  }

  // extract the roll, pitch and yaw angles from the rotation matrix fR6DOFn
  fAndroidAnglesDegFromRotationMatrix(pthisSV6DOF->fR6DOFn,
    &(pthisSV6DOF->fPhi6DOF), &(pthisSV6DOF->fThe6DOF),
    &(pthisSV6DOF->fPsi6DOF), &(pthisSV6DOF->fRho6DOF));

  // get the instantaneous orientation quaternion
  fQuaternionFromRotationMatrix(pthisSV6DOF->fR6DOFn, &(pthisSV6DOF->fq6DOFn));

  return;
}
```

4 Windows 8 Coordinate System

4.1 Rotation matrix

The Windows 8 rotation matrix for rotation of the smartphone coordinate system by yaw angle ψ about the z axis, then pitch angle θ about the x axis and finally by roll angle ϕ about the y axis is:

$$\boldsymbol{R}_{Win8}(\phi,\theta,\psi) = \boldsymbol{R}_{\boldsymbol{y}}(\phi)\boldsymbol{R}_{\boldsymbol{x}}(\theta)\boldsymbol{R}_{\boldsymbol{z}}(\psi) = \begin{pmatrix} cos\,\phi & 0 & -sin\phi \\ 0 & 1 & 0 \\ sin\phi & 0 & cos\phi \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\,\theta & sin\theta \\ 0 & -sin\theta & cos\theta \end{pmatrix}\begin{pmatrix} cos\,\psi & sin\psi & 0 \\ -sin\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (43)$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -sin\phi\,cos\,\theta \\ -cos\,\theta\,sin\psi & cos\,\theta\,cos\,\psi & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \quad (44)$$

4.2 Ranges of Euler angles

By direct evaluation, the matrix $\boldsymbol{R}_{Win8}(\phi+\pi,\pi-\theta,\psi+\pi)$ is equal to $\boldsymbol{R}_{Win8}(\phi,\theta,\psi)$:

$$\boldsymbol{R}_{Win8}(\phi+\pi,\pi-\theta,\psi+\pi)$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -cos\,\theta\,sin\phi \\ -cos\,\theta\,sin\psi & cos\,\psi\,cos\,\theta & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \quad (45)$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -cos\,\theta\,sin\phi \\ -cos\,\theta\,sin\psi & cos\,\psi\,cos\,\theta & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \quad (46)$$

$$\Rightarrow \boldsymbol{R}_{Win8}(\phi+\pi,\pi-\theta,\psi+\pi) = \boldsymbol{R}_{Win8}(\phi,\theta,\psi) \quad (47)$$

As a consequence, there are, in general, two sets of Euler angles for any rotation matrix if the Euler angles are allowed to have values over the full 360º range. The Windows 8 coordinate system therefore restricts the roll angle ϕ to the range $-90º \le \phi < 90º$ deg to remove one of these two solutions. The permitted ranges of Euler angles in the Windows 8 coordinate system are:

$$-90\ deg \le \phi < 90\ deg \quad (48)$$

$$-180\ deg \le \theta < 180\ deg \quad (49)$$

$$0\ deg \le \psi < 360\ deg \quad (50)$$

Figure 7, Figure 8,, and Figure 9 plot the Euler angles in the Windows 8 coordinate system as the smartphone is rotated through 360º in roll, pitch and yaw respectively with, in each case, the other two Euler angles set to zero. The pitch θ and yaw ψ plots are smooth, modulo 360º, since –180º is equivalent to +180º and since 0º is equivalent to 360º. The roll angle plot, however, shows the discontinuities in roll, pitch and yaw angles at –90º and 90º roll angles as defined by equation (47).

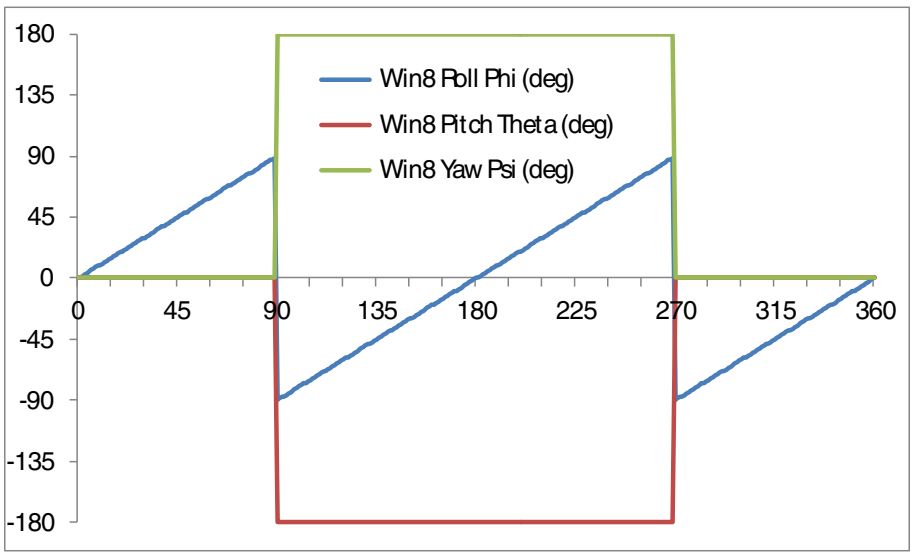

Figure 7. 360° roll rotation at zero pitch and yaw in the Windows 8 coordinate system

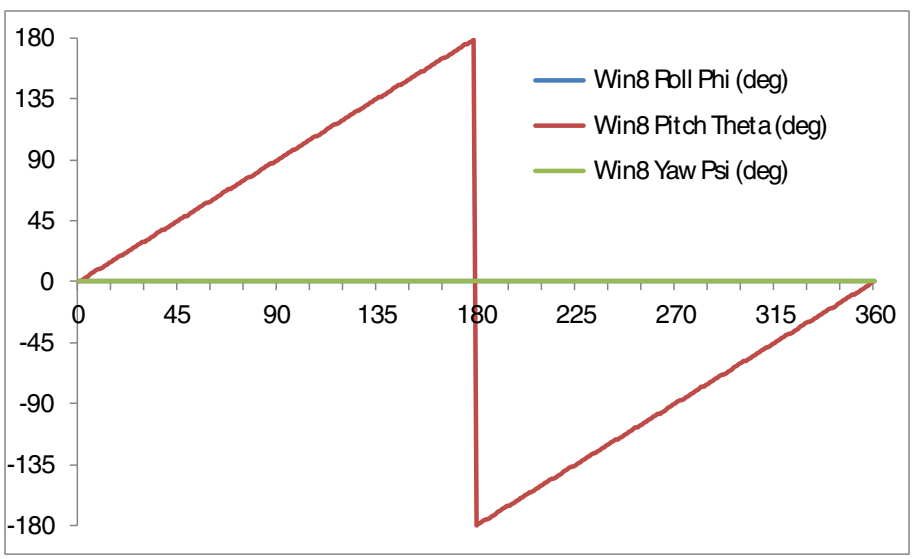

Figure 8. 360° pitch rotation at zero roll and yaw in the Windows 8 coordinate system

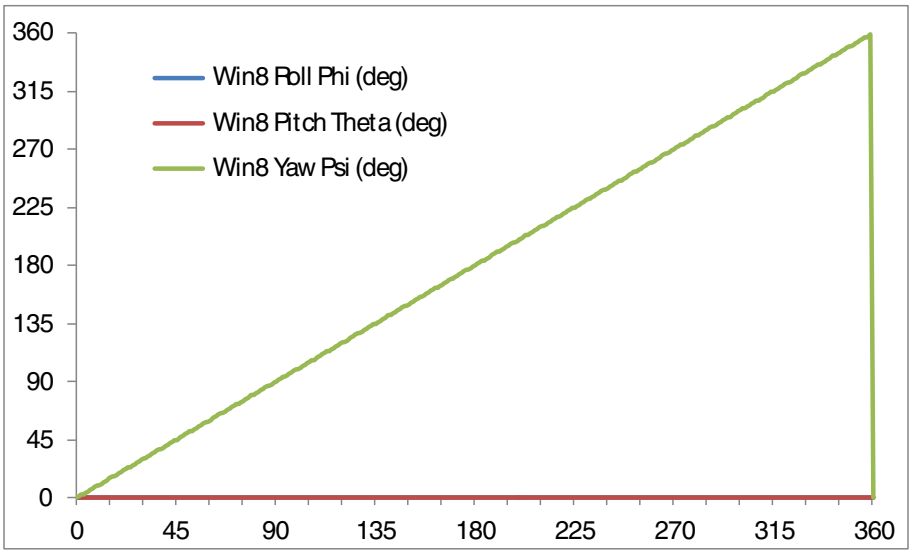

Figure 9. 360° yaw rotation at zero roll and pitch in the Windows 8 coordinate system

4.3 Sensor measurements

The accelerometer reading $\boldsymbol{G}_{Win8,p}$, in units of g, after rotation of the smartphone from its initial position (flat and pointing to magnetic north) by angles θ in pitch and ϕ in roll is:

$$\boldsymbol{G}_{Win8,p} = \begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \boldsymbol{R}_{Win8} \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} = \boldsymbol{R_y}(\phi)\boldsymbol{R_x}(\theta)\boldsymbol{R_z}(\psi) \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \tag{51}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -cos\,\theta\,sin\phi \\ -cos\,\theta\,sin\psi & cos\,\psi\,cos\,\theta & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} cos\,\theta\,sin\phi \\ -sin\,\theta \\ -cos\,\theta\,cos\phi \end{pmatrix} \tag{52}$$

The magnetometer reading, after removal of hard and soft iron distortion, after rotation by angles ψ in yaw, θ in pitch and ϕ in roll is:

$$\boldsymbol{B}_{Win8,c} = \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \boldsymbol{R}_{Win8} B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} = \boldsymbol{R_y}(\phi)\boldsymbol{R_x}(\theta)\boldsymbol{R_z}(\psi) B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \tag{53}$$

$$= \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\theta sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi sin\theta & -cos\,\theta\,sin\phi \\ -cos\,\theta\,sin\psi & cos\,\psi\,cos\,\theta & sin\theta \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi sin\theta & sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta & cos\,\phi\,cos\,\theta \end{pmatrix} B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} \tag{54}$$

$$= B \begin{pmatrix} cos\,\theta\,sin\phi sin\delta + (cos\,\psi\,sin\phi sin\theta + cos\,\phi\,sin\psi)cos\delta \\ cos\,\theta\,cos\,\psi\,cos\delta - sin\delta sin\theta \\ -cos\,\phi\,cos\,\theta\,sin\delta + (sin\phi sin\psi - cos\,\phi\,cos\,\psi\,sin\theta)cos\delta \end{pmatrix} \tag{55}$$

4.4 Solution for Euler Angles and Rotation Matrix

The rotation matrix, Euler angles and quaternion are computed in the Windows 8 coordinate system using a very similar approach to described in sections Solution for Euler angles and rotation matrix for the Aerospace/NED coordinate system and Solution for Euler angles and rotation matrix for the Android coordinate system. The rotation matrix is first computed directly from the accelerometer and magnetometer measurements and then the Euler angles and quaternion computed from the rotation matrix using the functions documented in Application Note AN4676.

Comparison of equations **(44)** and **(52)** shows that the third z column of the Windows 8 rotation matrix is simply the negated normalized accelerometer reading:

$$\begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} = \frac{-1}{|\boldsymbol{G}_{Win8,p}|} \begin{pmatrix} G_{Win8,px} \\ G_{Win8,py} \\ G_{Win8,pz} \end{pmatrix} \tag{56}$$

Simple algebra on equations **(43)**, **(52)** and **(55)** gives the expression for the first x column of the rotation matrix to be the negated normalized vector product of the calibrated magnetometer and accelerometer sensor readings:

$$\begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix} = \frac{-\begin{pmatrix} B_{Win8,cx} \\ B_{Win8,cy} \\ B_{Win8,cz} \end{pmatrix} \times \begin{pmatrix} G_{Win8,px} \\ G_{Win8,py} \\ G_{Win8,pz} \end{pmatrix}}{\left| \begin{pmatrix} B_{Win8,cx} \\ B_{Win8,cy} \\ B_{Win8,cz} \end{pmatrix} \times \begin{pmatrix} G_{Win8,px} \\ G_{Win8,py} \\ G_{Win8,pz} \end{pmatrix} \right|} \tag{57}$$

Finally, since any rotation matrix has orthonormal column vectors, the second y column of the rotation matrix is simply the normalized vector product of the z and x columns:

$$\begin{pmatrix} R_{01} \\ R_{11} \\ R_{21} \end{pmatrix} = \frac{\begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} \times \begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix}}{\left| \begin{pmatrix} R_{02} \\ R_{12} \\ R_{22} \end{pmatrix} \times \begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \end{pmatrix} \right|} \tag{58}$$

4.5 Inclination angle

The scalar product of the accelerometer and magnetometer vectors is invariant under rotation giving an expression for the magnetic inclination angle δ in the Windows 8 coordinate system.

$$\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} \cdot \begin{pmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \cdot B \begin{pmatrix} 0 \\ cos\delta \\ -sin\delta \end{pmatrix} = Bsin\delta \Longrightarrow sin\delta = \frac{G_{px}B_{cx} + G_{py}B_{cy} + G_{pz}B_{cz}}{|\boldsymbol{G}_p||\boldsymbol{B}_c|} \tag{59}$$

4.6 Gimbal lock

The rotation sequence under Windows 8 is the same yaw, then pitch and finally roll sequence as used in the Aerospace/NED coordinate system. Gimbal lock therefore occurs in the Windows 8 coordinate system at pitch angles θ of –90º and +90º when the final rotation in roll angle ϕ is about the same vertical axis as the initial yaw rotation angle ψ.

At θ=–90º, the Windows 8 rotation matrix simplifies to a function of the difference $\phi - \psi$ indicating that only the difference of the roll and yaw angles can be determined.

$$\boldsymbol{R}_{Win8}(\phi, \theta = -90deg, \psi) = \begin{pmatrix} cos\,\phi\,cos\,\psi + sin\phi sin\psi & cos\phi sin\psi - cos\,\psi\,sin\phi & 0 \\ 0 & 0 & -1 \\ cos\,\psi\,sin\phi - cos\,\phi\,sin\psi & sin\phi sin\psi + cos\,\phi\,cos\,\psi & 0 \end{pmatrix} \quad (60)$$

$$= \begin{pmatrix} cos(\phi - \psi) & -sin(\phi - \psi) & 0 \\ 0 & 0 & -1 \\ sin(\phi - \psi) & cos(\phi - \psi) & 0 \end{pmatrix} \quad (61)$$

At θ=90º, the Windows 8 rotation matrix simplifies to a function of the sum $\phi + \psi$ indicating that only the sum of the roll and yaw angles can be determined.

$$\boldsymbol{R}_{Win8}(\phi, \theta = 90deg, \psi) = \begin{pmatrix} cos\,\phi\,cos\,\psi - sin\phi sin\psi & cos\phi sin\psi + cos\,\psi\,sin\phi & 0 \\ 0 & 0 & 1 \\ cos\,\psi\,sin\phi + cos\,\phi\,sin\psi & sin\phi sin\psi - cos\,\phi\,cos\,\psi & 0 \end{pmatrix} \quad (62)$$

$$= \begin{pmatrix} cos(\phi + \psi) & sin(\phi + \psi) & 0 \\ 0 & 0 & 1 \\ sin(\phi + \psi) & -cos(\phi + \psi) & 0 \end{pmatrix} \quad (63)$$

4.7 eCompass C source code

```
// Win8: Direct tilt-compensated e-Compass function
void feCompassDirectWin8(struct SV6DOF *pthisSV6DOF, struct MagSensor *pthisMag,
  struct AccelSensor *pthisAccel)
{
  // local variables
  float fmodB, fmodG, fmodx, fmody;
  float fsinDelta;

  // place the negated un-normalized gravity and un-normalized geomagnetic vectors
  // into the rotation matrix z and y axes
  pthisSV6DOF->fR6DOFn[0][2] = -pthisAccel->fGpx;
  pthisSV6DOF->fR6DOFn[1][2] = -pthisAccel->fGpy;
  pthisSV6DOF->fR6DOFn[2][2] = -pthisAccel->fGpz;
  pthisSV6DOF->fR6DOFn[0][1] = pthisMag->fBcx;
  pthisSV6DOF->fR6DOFn[1][1] = pthisMag->fBcy;
  pthisSV6DOF->fR6DOFn[2][1] = pthisMag->fBcz;

  // set x vector to vector product of y and z vectors
  pthisSV6DOF->fR6DOFn[0][0] = pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[2][2] -
    pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[1][2];
  pthisSV6DOF->fR6DOFn[1][0] = pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[0][2] -
    pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[2][2];
  pthisSV6DOF->fR6DOFn[2][0] = pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[1][2] -
    pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[0][2];

  // set y vector to vector product of z and x vectors
  pthisSV6DOF->fR6DOFn[0][1] = pthisSV6DOF->fR6DOFn[1][2] * pthisSV6DOF->fR6DOFn[2][0] -
    pthisSV6DOF->fR6DOFn[2][2] * pthisSV6DOF->fR6DOFn[1][0];
```

```
  pthisSV6DOF->fR6DOFn[1][1] = pthisSV6DOF->fR6DOFn[2][2] * pthisSV6DOF->fR6DOFn[0][0] -
    pthisSV6DOF->fR6DOFn[0][2] * pthisSV6DOF->fR6DOFn[2][0];
  pthisSV6DOF->fR6DOFn[2][1] = pthisSV6DOF->fR6DOFn[0][2] * pthisSV6DOF->fR6DOFn[1][0] -
    pthisSV6DOF->fR6DOFn[1][2] * pthisSV6DOF->fR6DOFn[0][0];

  // calculate the vector moduli
  fmodG =  (float) sqrt(pthisAccel->fGpx * pthisAccel->fGpx +
    pthisAccel->fGpy * pthisAccel->fGpy +
    pthisAccel->fGpz * pthisAccel->fGpz);
  fmodB =  (float) sqrt(pthisMag->fBcx * pthisMag->fBcx +
    pthisMag->fBcy * pthisMag->fBcy +
    pthisMag->fBcz * pthisMag->fBcz);
  fmodx = (float) sqrt(pthisSV6DOF->fR6DOFn[0][0] * pthisSV6DOF->fR6DOFn[0][0] +
        pthisSV6DOF->fR6DOFn[1][0] * pthisSV6DOF->fR6DOFn[1][0] +
        pthisSV6DOF->fR6DOFn[2][0] * pthisSV6DOF->fR6DOFn[2][0]);
  fmody = (float) sqrt(pthisSV6DOF->fR6DOFn[0][1] * pthisSV6DOF->fR6DOFn[0][1] +
        pthisSV6DOF->fR6DOFn[1][1] * pthisSV6DOF->fR6DOFn[1][1] +
        pthisSV6DOF->fR6DOFn[2][1] * pthisSV6DOF->fR6DOFn[2][1]);

  // normalize the rotation matrix
  if (!((fmodx == 0.0F) || (fmody == 0.0F) || (fmodG == 0.0F)))
  {
    // normalize x axis
    pthisSV6DOF->fR6DOFn[0][0] /= fmodx;
    pthisSV6DOF->fR6DOFn[1][0] /= fmodx;
    pthisSV6DOF->fR6DOFn[2][0] /= fmodx;
    // normalize y axis
    pthisSV6DOF->fR6DOFn[0][1] /= fmody;
    pthisSV6DOF->fR6DOFn[1][1] /= fmody;
    pthisSV6DOF->fR6DOFn[2][1] /= fmody;
    // normalize z axis
    pthisSV6DOF->fR6DOFn[0][2] /= fmodG;
    pthisSV6DOF->fR6DOFn[1][2] /= fmodG;
    pthisSV6DOF->fR6DOFn[2][2] /= fmodG;

    // estimate the declination angle Delta (90 minus angle between the vectors)
    fsinDelta = (pthisAccel->fGpx * pthisMag->fBcx + pthisAccel->fGpy * pthisMag->fBcy +
      pthisAccel->fGpz * pthisMag->fBcz) / fmodG / fmodB;
    pthisSV6DOF->fDelta6DOFn = (float) asin(fsinDelta) * FRADTODEG;
  }
  else
  // no solution is possible to set rotation to identity matrix and delta to 0 degrees
  {
    fmatrixAeqI(pthisSV6DOF->fR6DOFn, 3);
    pthisSV6DOF->fDelta6DOFn = 0.0F;
  }

  // extract the roll, pitch and yaw angles from the rotation matrix fR6DOFn
  fWin8AnglesDegFromRotationMatrix(pthisSV6DOF->fR6DOFn,
    &(pthisSV6DOF->fPhi6DOF), &(pthisSV6DOF->fThe6DOF),
    &(pthisSV6DOF->fPsi6DOF), &(pthisSV6DOF->fRho6DOF));

  // get the instantaneous orientation quaternion
  fQuaternionFromRotationMatrix(pthisSV6DOF->fR6DOFn, &(pthisSV6DOF->fq6DOFn));

  return;
}
```

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: AN4685
Rev. 1.0, 4/2013