Freescale Semiconductor
Application Note

Document Number:AN4709
Rev 1.0, 5/2013

Tilt Estimation Using an Accelerometer

in Aerospace, Android and Windows 8 Coordinate Systems

by: Mark Pedley

Contents

1 Introduction

This Application Note is an addendum to Version 3 of the Xtrinsic eCompass software release provided under license at www.freescale.com/ecompass. It was written to address the topic of estimating tilt in the Aerospace (NED), Android and Windows 8 coordinate systems using an accelerometer sensor alone. The source code listings in this document are not therefore part of the Version 3 software release but are written to allow any licensee to easily add these additional functions to their own implementation with minimal effort.

The algorithms in this Application Note are classified as three degree of freedom (also termed 3DOF) in contrast to the eCompass software and application notes which are six degree of freedom (also termed 6DOF) algorithms. Because these algorithms only use an accelerometer sensor, the algorithms can compute roll and pitch (also termed tilt angles) from the horizontal but cannot compute a compass heading for which a magnetometer sensor is required.

This Application Note is part of the technical documentation for the Version 3 Xtrinsic software and its use and distribution are controlled by that license agreement.

2 The 3DOF Orientation Matrix

2.1 Introduction

Application Note AN4676 derives the orientation matrices in the Aerospace/NED, Android and Windows 8 coordinate systems after arbitrary roll, pitch and yaw angle rotations. Application Note AN4685 shows how this orientation matrix can be computed in a 6DOF system containing an accelerometer and magnetometer. The mathematics in this section are taken from these documents without re-derivation.

An accelerometer sensor is completely insensitive to rotations about the vertical gravity axis and is therefore, unsurprisingly, incapable of determining the yaw and compass heading angles. An accelerometer sensor can, however, determine the orientation matrix and quaternion for roll and pitch angles only assuming a zero yaw and compass heading angle. The orientation matrices in the three coordinate systems are derived in this section with this assumption.

2.2 Aerospace/NED

The Aerospace/NED rotation matrix for zero yaw angle ψ after rotation from the horizontal by pitch angle θ followed by roll angle ϕ is (see AN4676 equation 100):

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

The Aerospace/NED accelerometer reading at this orientation is, in units of *g* (see AN4685 equation 10):

$$\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \sin\phi\sqrt{1 - {G_{px}}^2} \\ \cos\phi\sqrt{1 - {G_{px}}^2} \end{pmatrix} \quad \textbf{(2)}$$

The positive square root for cosθ can always be taken in equation **(2)** because the pitch angle θ in the Aerospace/NED coordinate system varies between –90° and +90° giving a strictly non-negative value of cosθ.

Substituting equation **(2)** into equation **(1)** gives the 3DOF orientation matrix directly in terms of the accelerometer reading as:

$$\boldsymbol{R}_{NED}(\psi = 0) = \begin{pmatrix} \sqrt{1 - {G_{px}}^2} & 0 & G_{px} \\ \frac{-G_{px}G_{py}}{\sqrt{1 - {G_{px}}^2}} & \frac{G_{pz}}{\sqrt{1 - {G_{px}}^2}} & G_{py} \\ \frac{-G_{px}G_{pz}}{\sqrt{1 - {G_{px}}^2}} & \frac{-G_{py}}{\sqrt{1 - {G_{px}}^2}} & G_{pz} \end{pmatrix} = \begin{pmatrix} \sqrt{{G_{py}}^2 + {G_{pz}}^2} & 0 & G_{px} \\ \frac{-G_{px}G_{py}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & \frac{G_{pz}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & G_{py} \\ \frac{-G_{px}G_{pz}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & \frac{-G_{py}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & G_{pz} \end{pmatrix} \quad \textbf{(3)}$$

2.3 Android

The Android rotation matrix for zero yaw angle ψ after rotation from the horizontal by roll angle ϕ followed by pitch angle θ is (see AN4676 equation 116):

$$\boldsymbol{R}_{Android}(\psi = 0) = \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ \sin\phi\sin\theta & \cos\theta & -\cos\phi\sin\theta \\ -\cos\theta\sin\phi & \sin\theta & \cos\phi\cos\theta \end{pmatrix} \tag{4}$$

The Android accelerometer reading at this orientation is, in units of g (see AN4685 equation 31):

$$\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} \sin\phi \\ -\cos\phi\sin\theta \\ \cos\phi\cos\theta \end{pmatrix} = \begin{pmatrix} \sin\phi \\ -\sin\theta\sqrt{1 - {G_{px}}^2} \\ \cos\theta\sqrt{1 - {G_{px}}^2} \end{pmatrix} \tag{5}$$

The positive square root for cosϕ can always be taken in equation (5) because the roll angle ϕ in the Android coordinate system varies between –90° and +90° giving a strictly non-negative value of cosϕ.

Substituting equation (5) into equation (4) gives the 3DOF orientation matrix directly in terms of the accelerometer reading as:

$$\boldsymbol{R}_{Android}(\psi = 0) = \begin{pmatrix} \sqrt{1 - {G_{px}}^2} & 0 & G_{px} \\ \frac{-G_{px}G_{py}}{\sqrt{1 - {G_{px}}^2}} & \frac{G_{pz}}{\sqrt{1 - {G_{px}}^2}} & G_{py} \\ \frac{-G_{px}G_{pz}}{\sqrt{1 - {G_{px}}^2}} & \frac{-G_{py}}{\sqrt{1 - {G_{px}}^2}} & G_{pz} \end{pmatrix} = \begin{pmatrix} \sqrt{{G_{py}}^2 + {G_{pz}}^2} & 0 & G_{px} \\ \frac{-G_{px}G_{py}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & \frac{G_{pz}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & G_{py} \\ \frac{-G_{px}G_{pz}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & \frac{-G_{py}}{\sqrt{{G_{py}}^2 + {G_{pz}}^2}} & G_{pz} \end{pmatrix} \tag{6}$$

2.4 Windows 8

The Windows 8 rotation matrix for zero yaw angle ψ after rotation from the horizontal by pitch angle θ followed by roll angle ϕ is (see AN4676 equation 132):

$$\boldsymbol{R}_{Win8}(\psi = 0) = \begin{pmatrix} \cos\phi & \sin\phi\sin\theta & -\sin\phi\cos\theta \\ 0 & \cos\theta & \sin\theta \\ \sin\phi & -\cos\phi\sin\theta & \cos\phi\cos\theta \end{pmatrix} \tag{7}$$

The Windows 8 accelerometer reading at this orientation is, in units of g (see AN4685 equation 52):

$$\begin{pmatrix} G_{px} \\ G_{py} \\ G_{pz} \end{pmatrix} = \begin{pmatrix} \cos\theta \sin\phi \\ -\sin\theta \\ -\cos\theta \cos\phi \end{pmatrix} \tag{8}$$

The roll angle ϕ in the Windows 8 coordinate system varies between –90° and +90° giving a strictly non-negative value of cosϕ. The positive square root can, therefore, always be taken in equations (9) , (10) and (11) .

$$\tan\phi = \frac{-G_{px}}{G_{pz}} \Rightarrow \cos\phi = \frac{1}{\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2}} \tag{9}$$

$$\Rightarrow \sin\phi = \tan\phi\cos\phi = \frac{-G_{px}}{G_{pz}\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2}} \tag{10}$$

$$\cos\theta = \frac{G_{px}}{\sin\phi} = -G_{pz}\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2} \tag{11}$$

Substituting equations (9) through (11) into equation (7) gives the 3DOF orientation matrix directly in terms of the accelerometer reading as:

$$\boldsymbol{R}_{Win8}(\psi = 0) = \begin{pmatrix} \frac{1}{\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2}} & \frac{G_{px}G_{py}}{G_{pz}\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2}} & -G_{px} \\ 0 & -G_{pz}\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2} & -G_{py} \\ \frac{-G_{px}}{G_{pz}\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2}} & \frac{G_{py}}{\sqrt{1 + \left(\frac{G_{px}}{G_{pz}}\right)^2}} & -G_{pz} \end{pmatrix} \tag{12}$$

3 The 3DOF State Vector

By analogy with the 6DOF state vector defined in AN4700, the 3DOF state vector can be defined as follows. The yaw and compass heading angles are included to allow re-use of the 6DOF functions even though these angles are, by definition, always zero in the 3DOF orientation.

```
// 3DOF orientation structure definition
struct SV3DOF
{
    // Euler angles
    float fLPPhi3DOF;                    // 3DOF low pass roll (deg)
    float fLPThe3DOF;                    // 3DOF low pass pitch (deg)
    float fLPPsi3DOF;                    // 3DOF low pass yaw (deg)
    float fLPRho3DOF;                    // 3DOF low pass compass (deg)
    // orientation matrices and quaternions
    float xfR3DOFn[3][3];               // 3DOF rotation matrix for current sample n
    float *fR3DOFn[3];
    float xfR3DOFnm1[3][3];             // 3DOF rotation matrix for sample n-1
    float *fR3DOFnm1[3];
    float xfR3DOFnm2[3][3];             // 3DOF rotation matrix for sample n-2
    float *fR3DOFnm2[3];
    float xfLPR3DOFn[3][3];             // low pass 3DOF rotation matrix for sample n
    float *fLPR3DOFn[3];
    float xfLPR3DOFnm1[3][3];           // low pass 3DOF rotation matrix for sample n-1
    float *fLPR3DOFnm1[3];
    float xfLPR3DOFnm2[3][3];           // low pass 3DOF rotation matrix for sample n-1
    float *fLPR3DOFnm2[3];
    struct fquaternion fLPq3DOFn;       // low pass 3DOF orientation quaternion
};
```

For consistency with how matrices are handled elsewhere in the code as arrays of pointers to matrix rows (see AN4699), the 3DOF state vector matrices should be initialized as:

```
for (i = 0; i < 3; i++)
{
    thisSV3DOF.fR3DOFn[i] = thisSV3DOF.xfR3DOFn[i];
    thisSV3DOF.fR3DOFnm1[i] = thisSV3DOF.xfR3DOFnm1[i];
    thisSV3DOF.fR3DOFnm2[i] = thisSV3DOF.xfR3DOFnm2[i];
    thisSV3DOF.fLPR3DOFn[i] = thisSV3DOF.xfLPR3DOFn[i];
    thisSV3DOF.fLPR3DOFnm1[i] = thisSV3DOF.xfLPR3DOFnm1[i];
    thisSV3DOF.fLPR3DOFnm2[i] = thisSV3DOF.xfLPR3DOFnm2[i];
}
```

4 Source Code Listings

4.1 Function f3DOFTiltNED

Function `f3DOFTiltNED` implements equation (3) to compute the instantaneous Aerospace / NED tilt orientation matrix from the accelerometer reading:

```
// Xtrinsic Aerospace / NED accelerometer 3DOF tilt function
void f3DOFTiltNED(struct SV3DOF *pthisSV3DOF, struct AccelSensor *pthisAccel)
{
  // local variables
  float gx, gy, gz;        // normalized accelerometer reading
  float fmodG;             // modulus of the accelerometer reading
  float fR00;              // R[0][0] and normalization factor

  // compute the accelerometer magnitude and R00
  fmodG = (float) sqrt(pthisAccel->fGpx * pthisAccel->fGpx + pthisAccel->fGpy *
        pthisAccel->fGpy + pthisAccel->fGpz * pthisAccel->fGpz);
  fR00 = (float) sqrt(pthisAccel->fGpy * pthisAccel->fGpy + pthisAccel->fGpz *
        pthisAccel->fGpz);
  if ((fmodG != 0.0F) && (fR00 != 0.0F))
  {
        // normalize the accelerometer reading and R00
        gx = pthisAccel->fGpx / fmodG;
        gy = pthisAccel->fGpy / fmodG;
        gz = pthisAccel->fGpz / fmodG;
        fR00 /= fmodG;

        // construct the orientation matrix
        pthisSV3DOF->fR3DOFn[0][0] = fR00;
        pthisSV3DOF->fR3DOFn[0][1] = 0.0F;
        pthisSV3DOF->fR3DOFn[0][2]= gx;
        pthisSV3DOF->fR3DOFn[1][0] = -gx * gy / fR00;
        pthisSV3DOF->fR3DOFn[1][1] = gz / fR00;
        pthisSV3DOF->fR3DOFn[1][2]= gy;

        pthisSV3DOF->fR3DOFn[2][0] = -gx * gz / fR00;
        pthisSV3DOF->fR3DOFn[2][1] = -gy / fR00;
        pthisSV3DOF->fR3DOFn[2][2]= gz;
        }
  else
  {
        // either freefall or vertical orientation preventing solution
        fmatrixAeqI(pthisSV3DOF->fR3DOFn, 3);
  }

    return;
}
```

4.2 Function f3DOFTiltAndroid

Function `f3DOFTiltAndroid` implements equation (6) to compute the instantaneous Android tilt orientation matrix from the accelerometer reading:

```
// Xtrinsic Android accelerometer 3DOF tilt function
void f3DOFTiltAndroid(struct SV3DOF *pthisSV3DOF, struct AccelSensor *pthisAccel)
{
  // local variables
  float gx, gy, gz;           // normalized accelerometer reading
  float fmodG;                // modulus of the accelerometer reading
  float fR00;                 // R[0][0] and normalization factor

  // compute the accelerometer magnitude and R00
  fmodG = (float) sqrt(pthisAccel->fGpx * pthisAccel->fGpx + pthisAccel->fGpy *
        pthisAccel->fGpy + pthisAccel->fGpz * pthisAccel->fGpz);
  fR00 = (float) sqrt(pthisAccel->fGpy * pthisAccel->fGpy + pthisAccel->fGpz *
       pthisAccel->fGpz);
  if ((fmodG != 0.0F) && (fR00 != 0.0F))
  {
        // normalize the accelerometer reading and R00
        gx = pthisAccel->fGpx / fmodG;
        gy = pthisAccel->fGpy / fmodG;
        gz = pthisAccel->fGpz / fmodG;
        fR00 /= fmodG;

        // construct the orientation matrix
        pthisSV3DOF->fR3DOFn[0][0] = fR00;
        pthisSV3DOF->fR3DOFn[0][1] = 0.0F;
        pthisSV3DOF->fR3DOFn[0][2]= gx;

        pthisSV3DOF->fR3DOFn[1][0] = -gx * gy / fR00;
        pthisSV3DOF->fR3DOFn[1][1] = gz / fR00;
        pthisSV3DOF->fR3DOFn[1][2]= gy;

        pthisSV3DOF->fR3DOFn[2][0] = -gx * gz / fR00;
        pthisSV3DOF->fR3DOFn[2][1] = -gy / fR00;
        pthisSV3DOF->fR3DOFn[2][2]= gz;
        }
    else
    {
        // either freefall or vertical orientation preventing solution
        fmatrixAeqI(pthisSV3DOF->fR3DOFn, 3);
    }

    return;
}
```

4.3 Function f3DOFTiltWin8

Function `f3DOFTiltWin8` implements equation (12) to compute the instantaneous Windows 8 tilt orientation matrix from the accelerometer reading:

```
// Xtrinsic Windows 8 accelerometer 3DOF tilt function
void f3DOFTiltWin8(struct SV3DOF *pthisSV3DOF, struct AccelSensor *pthisAccel)
{
  // local variables
  float gx, gy, gz;        // normalized accelerometer reading
  float fmodG;             // modulus of the accelerometer reading
  float fR00;              // R[0][0] and normalization factor

  // compute the accelerometer magnitude and R00
  fmodG = (float) sqrt(pthisAccel->fGpx * pthisAccel->fGpx + pthisAccel->fGpy *
        pthisAccel->fGpy + pthisAccel->fGpz * pthisAccel->fGpz);
  if ((fmodG != 0.0F) && (pthisAccel->fGpz != 0.0F))
  {
        // normalize the accelerometer reading and R00
        gx = pthisAccel->fGpx / fmodG;
        gy = pthisAccel->fGpy / fmodG;
        gz = pthisAccel->fGpz / fmodG;
        fR00 = 1.0F / (float) sqrt(1.0F + (gx * gx) / (gz * gz));

        // construct the orientation matrix
        pthisSV3DOF->fR3DOFn[0][0]  = fR00;
        pthisSV3DOF->fR3DOFn[0][1]  = gx * gy / gz * fR00;
        pthisSV3DOF->fR3DOFn[0][2]= -gx;

        pthisSV3DOF->fR3DOFn[1][0]  = 0.0F;
        pthisSV3DOF->fR3DOFn[1][1]  = -gz / fR00;
        pthisSV3DOF->fR3DOFn[1][2]= -gy;

        pthisSV3DOF->fR3DOFn[2][0]  = -gx / gz * fR00;
        pthisSV3DOF->fR3DOFn[2][1]  = gy * fR00;
        pthisSV3DOF->fR3DOFn[2][2]= -gz;
        }
  else
  {
        // either freefall or vertical orientation preventing solution
        fmatrixAeqI(pthisSV3DOF->fR3DOFn, 3);
  }

    return;
}
```

4.4 fLPFOrientationMatrix

The function `fLPFOrientationMatrix` listed below implements the same low pass Butterworth filter described in AN4697. The filter parameters `fb0`, `fa1` and `fa2` should be initialized by a call to `fInitLPFOrientationMatrix`. The function below is a more general version of the function by the same name in AN4697 which expects the orientation matrices to be provided within a structure of type struct SV6DOF. The function below can therefore be used for filtering any sequence of rotation matrices including those in an accelerometer tilt state vector of type `structSV3DOF`. Additional differences with the version listed in AN4697 are:

- the low pass filtering of the magnetic inclination angle is removed because this is not relevant to accelerometer-only tilt estimation
- the rotation matrix delay line shuffling is included in the function.

```
// function low pass filters an orientation matrix
void fLPFOrientationMatrix(float **fRn, float **fRnm1, float **fRnm2,
        float **fLPRn, float **fLPRnm1, float **fLPRnm2,
        int32 loopcounter, float fb0, float fa1, float fa2)
{
    // local variables
    int32 i, j;                         // loop counters

    // initialize delay lines on first pass
    if (loopcounter == 0)
    {
        // set R[LP,n-2]=R[LP,n-1]=R[n-2]=R[n-1]=R[n]
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                fLPRnm2[i][j] = fLPRnm1[i][j] = fRnm2[i][j] = fRnm1[i][j] = fRn[i][j];
            }
        }
    }

    // low pass filter the nine elements of the rotation matrix
    // the resulting matrix will no longer be perfectly orthonormal
    if (ANGLE_LPF_FPU == 1.0F)
    {
        // all pass case for orientation matrix
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                fLPRn[i][j] = fRn[i][j];
            }
        }
    }
    else
    {
        // apply the low pass Butterworth filter to the orientation matrix
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                fLPRn[i][j] = fb0 * (fRn[i][j] + 2.0F * fRnm1[i][j] + fRnm2[i][j]) +
                        fa1 * fLPRnm1[i][j] + fa2 * fLPRnm2[i][j];
            }
        }
    }

    // renormalize the low pass filtered orientation matrix
    fmatrixAeqRenormRotA(fLPRn);

    // shuffle the filter delay lines for the next iteration
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            fLPRnm2[i][j] = fLPRnm1[i][j];
            fLPRnm1[i][j] = fLPRn[i][j];
            fRnm2[i][j] = fRnm1[i][j];
            fRnm1[i][j] = fRn[i][j];
        }
    }

    return;
}
```

5 Control Loop Listing

This code should be placed in the main sampling loop immediately after the accelerometer data structure `thisAccel` is populated with the accelerometer reading for the current iteration. These calls compute the instantaneous orientation matrix, then low pass filter that matrix and compute the low pass filtered quaternion and Euler angles from the low pass filtered matrix.

```
// 3DOF: apply the tilt estimation algorithm to get the instantaneous orientation matrix,
// then compute the low pass filtered orientation matrix, low pass filtered quaternion and
// low pass filtered Euler angles
switch (iCoordSystem)
{
case NED:
  f3DOFTiltNED(&thisSV3DOF, &thisAccel);
  fLPFOrientationMatrix(thisSV3DOF.fR3DOFn, thisSV3DOF.fR3DOFnm1, thisSV3DOF.fR3DOFnm2,
      thisSV3DOF.fLPR3DOFn, thisSV3DOF.fLPR3DOFnm1, thisSV3DOF.fLPR3DOFnm2,
      loopcounter, fb0, fa1, fa2);
  fQuaternionFromRotationMatrix(thisSV3DOF.fLPR3DOFn, &(thisSV3DOF.fLPq3DOFn));
  fNEDAnglesDegFromRotationMatrix(thisSV3DOF.fLPR3DOFn, &(thisSV3DOF.fLPPhi3DOF),
      &(thisSV3DOF.fLPThe3DOF), &(thisSV3DOF.fLPPsi3DOF), &(thisSV3DOF.fLPRho3DOF));
  break;
case ANDROID:
  f3DOFTiltAndroid(&thisSV3DOF, &thisAccel);
  fLPFOrientationMatrix(thisSV3DOF.fR3DOFn, thisSV3DOF.fR3DOFnm1, thisSV3DOF.fR3DOFnm2,
      thisSV3DOF.fLPR3DOFn, thisSV3DOF.fLPR3DOFnm1, thisSV3DOF.fLPR3DOFnm2,
      loopcounter, fb0, fa1, fa2);
  fQuaternionFromRotationMatrix(thisSV3DOF.fLPR3DOFn, &(thisSV3DOF.fLPq3DOFn));
  fAndroidAnglesDegFromRotationMatrix(thisSV3DOF.fLPR3DOFn, &(thisSV3DOF.fLPPhi3DOF),
      &(thisSV3DOF.fLPThe3DOF), &(thisSV3DOF.fLPPsi3DOF), &(thisSV3DOF.fLPRho3DOF));
  break;
case WIN8:
default:
  f3DOFTiltWin8(&thisSV3DOF, &thisAccel);
  fLPFOrientationMatrix(thisSV3DOF.fR3DOFn, thisSV3DOF.fR3DOFnm1, thisSV3DOF.fR3DOFnm2,
      thisSV3DOF.fLPR3DOFn, thisSV3DOF.fLPR3DOFnm1, thisSV3DOF.fLPR3DOFnm2,
      loopcounter, fb0, fa1, fa2);
  fQuaternionFromRotationMatrix(thisSV3DOF.fLPR3DOFn, &(thisSV3DOF.fLPq3DOFn));
  fWin8AnglesDegFromRotationMatrix(thisSV3DOF.fLPR3DOFn, &(thisSV3DOF.fLPPhi3DOF),
        &(thisSV3DOF.fLPThe3DOF), &(thisSV3DOF.fLPPsi3DOF), &(thisSV3DOF.fLPRho3DOF));
  break;
}
```

Document Number AN4709
Rev 1.0, 5/2013
8 May 2013