Freescale Semiconductor
Application Note

Document Number:AN4700
Rev. 1.0, 4/2013

Control Loop, Data Structures and Compile Time Constants

by: Mark Pedley

1 Introduction

This Application Note is part of the documentation for 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 that license agreement.

1.1 Summary

This document gives guidance to users who are migrating the reference Freescale eCompass and calibration software to their end embedded system. It covers the overall control loop calling the eCompass and calibration algorithms, data structures and compile time constants that can be adjusted if needed.

Contents

2 Control Loop

2.1 Example final code

The code example below is a stripped down version of the control loop in the supplied reference software after the removal of extraneous lines of code only relevant for the algorithm validation in the sensor simulation environment. It uses the Android eCompass function and the 7 element calibration algorithm.

This code should be executed at a fixed sampling rate driven by timer interrupt or real-time operating system. Freescale uses 50 Hz on its demonstration eCompass software but smartphone vendors can use sampling rates as low as 10 Hz to minimize power consumption.

For simplicity, the reference software is single threaded with the expensive magnetic calibration algorithm called every `INITIALCALINTERVAL` or `FINALCALINTERVAL` iterations. Advanced users may wish to implement the calibration function as a second thread running continuously at low priority to avoid a timing glitch of a fraction of a second every time the calibration function is called.

```
// main sampling loop (typically 10Hz to 50Hz on real systems)
// for evaluation purposes, this is written as a loop over a fixed
// number of iterations. for a real time system, this should be
// implemented as an infinite loop either called at the sampling
// frequency by an RTOS or simply with an idle until timing interrupt
// at the top of each loop
{
    // call 6DOF sensor driver to get sensor data
    fSixDOFSensorDrivers(&thisAccel, &thisMag);

    // update the magnetometer measurement buffer integer magnetometer data
    fUpdateMagnetometerBuffer(&thisMagneticBuffer, &thisMag, &thisAccel, loopcounter);

    // remove hard and soft iron terms from Bp (uT) to get calibrated data Bc (uT)
    fInvertMagCal(&thisMag, &thisMagCal);

    // pass the accel and calibrated mag data to the Android eCompass
    feCompassAndroid(&thisSV6DOF, &thisMag, &thisAccel);

    // low pass filter the orientation matrix and get low pass quaternion and Euler angles
    fLPFOrientationMatrix(&thisSV6DOF, iCoordSystem, loopcounter);

    // shuffle the rotation matrix low pass filter delay lines
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
        {
            thisSV6DOF.fLPR6DOFnm2[i][j] = thisSV6DOF.fLPR6DOFnm1[i][j];
            thisSV6DOF.fLPR6DOFnm1[i][j] = thisSV6DOF.fLPR6DOFn[i][j];
            thisSV6DOF.fR6DOFnm2[i][j] = thisSV6DOF.fR6DOFnm1[i][j];
            thisSV6DOF.fR6DOFnm1[i][j] = thisSV6DOF.fR6DOFn[i][j];
        }

    // update the inclination angle low pass filter delay lines
    thisSV6DOF.fLPDelta6DOFnm2 = thisSV6DOF.fLPDelta6DOFnm1;
    thisSV6DOF.fLPDelta6DOFnm1 = thisSV6DOF.fLPDelta6DOFn;
    thisSV6DOF.fDelta6DOFnm2 = thisSV6DOF.fDelta6DOFnm1;
    thisSV6DOF.fDelta6DOFnm1 = thisSV6DOF.fDelta6DOFn;

    // the following section of code executes the calibration algorithms
    // and decides if the new calibration should be accepted. this code
    // is compute-intensive and is best implemented under an RTOS as a low
    // priority task which might execute every minute or so. here it is
    // implemented every INITALCALINTERVAL or FINALCALINTERVAL  iterations
    // which is a simpler way to demonstrate the procedure.
```

```
    // the drawback of the simpler approach here is a minor timing change
    // in the eCompass update interval when the calibration code executes.
    // check for enough data in magnetic buffer for a calibration
    if (thisMagneticBuffer.iMagBufferCount >= MINEQUATIONS)
        {
        // calibrate if this will be the first calibration
        // or initially ever INITIALCALINTERVAL iterations or
        // ultimately every FINALCALINTERVAL iterations
        if ((!thisMagCal.iValidMagCal) ||
            (thisMagCal.iValidMagCal && !(loopcounter % INITIALCALINTERVAL) &&
    (thisMagneticBuffer.iMagBufferCount <= MEDEQUATIONS)) ||
            (thisMagCal.iValidMagCal && !(loopcounter % FINALCALINTERVAL)))
        {
            // 7 point eigenpair calibration
            fUpdateCalibration7EIG(&thisMagCal, &thisMagneticBuffer, ftmpA7x7,
                ftmpB7x7, ftmpA7x1);

            // accept new calibration if:
            // a) number of measurements is MEDEQUATIONS or less or
            // b) fit error is reduced and geomagnetic field is in range
            // (actual range of geomagnetic field strength B on earth varies 22uT to 67uT)
            if ((thisMagneticBuffer.iMagBufferCount <= MEDEQUATIONS) ||
                (thisMagCal.ftrFitErrorpc <= thisMagCal.fFitErrorpc) &&
                (thisMagCal.ftrB >= MINBFIT) && (thisMagCal.ftrB <= MAXBFIT))
            {
                thisMagCal.fFitErrorpc = thisMagCal.ftrFitErrorpc;
                thisMagCal.fB = thisMagCal.ftrB;
                thisMagCal.fVx = thisMagCal.ftrVx;
                thisMagCal.fVy = thisMagCal.ftrVy;
                thisMagCal.fVz = thisMagCal.ftrVz;
                fmatrixAeqB(thisMagCal.finvW, thisMagCal.ftrinvW, 3, 3);
            }

        // age (increase) the calibration fit to avoid a good calibration preventing
        // future updates. FITERRORAGING is the reciprocal of the time (s) for the
        // fit error to increase by e=2.718. FINALCALINTERVAL * DELTAT is the interval
        // in seconds between each aging update of fFitErrorpc.
        // (1 + FITERRORAGING * FINALCALINTERVAL * DELTAT)^n=e defines n, the number
        // of updates for e fold increase.
        // approx n * FINALCALINTERVAL * DELTAT = 1. / FITERRORAGING
        // so FITERRORAGING is the reciprocal of the time in secs for e fold increase
        thisMagCal.fFitErrorpc += thisMagCal.fFitErrorpc * FITERRORAGING *
            (float) FINALCALINTERVAL * DELTAT;

        } // end of test whether to call calibration functions
    }    // end of test for MINEQUATIONS
} // end of infinite timing loop
```

2.2 Commentary

The sensor data is read from the physical accelerometer and magnetometer sensors. This function call is to your I^2C or SPI driver code and not to the supplied sensor simulation driver. The arguments `iCoordSystem` and `&thisSimulation` (present in the reference code) are not therefore required. The sensor driver is extremely simple to write and specific to your hardware. Freescale does not therefore provide example drivers in the reference software since this would be more confusing than help. Since different circuit board designers will choose to orient the sensors at different angles, you will need to align the sensor axes to your product axes with a few lines of software. This is as simple as "flip the x and y accelerometer axes and negate the z axis" and similarly for the magnetometer sensor axes.

```
    // call 6DOF sensor driver to get sensor data
    fSixDOFSensorDrivers(&thisAccel, &thisMag);
```

The magnetometer measurement is placed in the magnetometer measurement buffer for future use by the calibration algorithm. The loopcounter is used to time stamp the measurement to allow the calibration algorithm to prioritize more recent measurements. This function call is cheap and users may want to consider keeping the magnetic buffer updated with measurements even when the eCompass is not executing. That way, the eCompass will be able to immediately calibrate itself without the need to perform a figure-8 hand movement to obtain new measurements.

```
// update the magnetometer measurement buffer integer magnetometer data
fUpdateMagnetometerBuffer(&thisMagneticBuffer, &thisMag, &thisAccel, loopcounter);
```

The hard and soft iron distortion is removed from the current magnetometer measurement:

```
// remove hard and soft iron terms from Bp (uT) to get calibrated data Bc (uT)
fInvertMagCal(&thisMag, &thisMagCal);
```

The instantaneous (and noisy) orientation is then computed (to the Android standard in this example).

```
// pass the accel and calibrated mag data to the Android eCompass
feCompassAndroid(&thisSV6DOF, &thisMag, &thisAccel);
```

The orientation is low pass filtered to reduce noise and the filter delay lines shuffled for the next iteration:

```
// low pass filter the orientation matrix and get low pass quaternion and Euler angles
fLPFOrientationMatrix(&thisSV6DOF, iCoordSystem, loopcounter);

// shuffle the rotation matrix low pass filter delay lines
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
    {
        thisSV6DOF.fLPR6DOFnm2[i][j] = thisSV6DOF.fLPR6DOFnm1[i][j];
        thisSV6DOF.fLPR6DOFnm1[i][j] = thisSV6DOF.fLPR6DOFn[i][j];
        thisSV6DOF.fR6DOFnm2[i][j] = thisSV6DOF.fR6DOFnm1[i][j];
        thisSV6DOF.fR6DOFnm1[i][j] = thisSV6DOF.fR6DOFn[i][j];
    }

// update the inclination angle low pass filter delay lines
thisSV6DOF.fLPDelta6DOFnm2 = thisSV6DOF.fLPDelta6DOFnm1;
thisSV6DOF.fLPDelta6DOFnm1 = thisSV6DOF.fLPDelta6DOFn;
thisSV6DOF.fDelta6DOFnm2 = thisSV6DOF.fDelta6DOFnm1;
thisSV6DOF.fDelta6DOFnm1 = thisSV6DOF.fDelta6DOFn;
```

The first test is to determine if sufficient magnetometer measurements are available to produce a reasonable calibration:

```
// the following section of code executes the calibration algorithms
if (thisMagneticBuffer.iMagBufferCount >= MINEQUATIONS)
    {
```

Do a calibration immediately if no valid calibration has yet been computed or, alternatively, every `INITIALCALINTERVAL` iterations while the number of magnetometer is limited but increasing to every `FINALCALINTERVAL` iterations when the magnetic measurement buffer is adequately filled. The reasoning here is that the calibration should be attempted as soon as is reasonably possible with the understanding that this initial estimate is based on a small number of measurements. In the short term, it makes sense to update the calibration more rapidly (every `INITIALCALINTERVAL` iterations) while the magnetic measurement buffer fills up in order to improve the calibration quality as quickly as possible. However, in the long term, the calibration interval should be increased to `FINALCALINTERVAL` iterations to reduce power consumption since the calibration will only change slowly as a result of temperature changes.

```
        // calibrate if this will be the first calibration
        // or initially ever INITIALCALINTERVAL iterations or
        // ultimately every FINALCALINTERVAL iterations
        if ((!thisMagCal.iValidMagCal) ||
            (thisMagCal.iValidMagCal && !(loopcounter % INITIALCALINTERVAL) &&
(thisMagneticBuffer.iMagBufferCount <= MEDEQUATIONS)) ||
            (thisMagCal.iValidMagCal && !(loopcounter % FINALCALINTERVAL)))
        {
```

In this example, the 7 element calibration algorithm is called:

```
            // 7 point eigenpair calibration
            fUpdateCalibration7EIG(&thisMagCal, &thisMagneticBuffer, ftmpA7x7, ftmpB7x7,
ftmpA7x1);
```

Accept the new trial calibration if it results from using more measurements (up to the limit `MEDEQUATIONS`) or if the fit error is reduced and the geomagnetic field value is sensible (no jamming is present):

```
            // accept new calibration if:
            // a) number of measurements is MEDEQUATIONS or less or
            // b) fit error is reduced and geomagnetic field is in range
            // (actual range of geomagnetic field strength B on earth varies 22uT to 67uT)
            if ((thisMagneticBuffer.iMagBufferCount <= MEDEQUATIONS) ||
                (thisMagCal.ftrFitErrorpc <= thisMagCal.fFitErrorpc) &&
                (thisMagCal.ftrB >= MINBFIT) && (thisMagCal.ftrB <= MAXBFIT))
            {
```

If these conditions are met, accept the new calibration to replace the existing one:

```
                thisMagCal.fFitErrorpc = thisMagCal.ftrFitErrorpc;
                thisMagCal.fB = thisMagCal.ftrB;
                thisMagCal.fVx = thisMagCal.ftrVx;
                thisMagCal.fVy = thisMagCal.ftrVy;
                thisMagCal.fVz = thisMagCal.ftrVz;
                fmatrixAeqB(thisMagCal.finvW, thisMagCal.ftrinvW, 3, 3);
```

Artificially age the calibration fit error to prevent lockout by a single excellent prior calibration. This line will ultimately force a new calibration to replace any existing calibration.

```
    // age (increase) the calibration fit to avoid a good calibration preventing
    // future updates.
    thisMagCal.fFitErrorpc += thisMagCal.fFitErrorpc * FITERRORAGING *
            (float) FINALCALINTERVAL * DELTAT;
```

3 Data Structures

3.1 fquaternion

The quaternion structure is self-explanatory. Some mathematical texts label the vector components `q1`, `q2`, and `q3` as `qx`, `qy` and `qz`. The scalar component `q0` is sometimes listed in sequence after the vector component.

```
// quaternion structure definition
struct fquaternion
{
    float q0;        // scalar component
    float q1;        // x vector component
    float q2;        // y vector component
    float q3;        // z vector component
};
```

3.2 SimulationModel

The `SimulationModel` structure and the `thisSimulation` declaration should not be present in the final embedded software. This structure simulates phenomena which are implicit in real world measurements and will not be present in your final embedded software.

```
// 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 xinvW[3][3];          // simulated inverse soft iron matrix
    float *finvW[3];
    float xW[3][3];             // simulated forward 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)
};
```

3.3 AccelSensor

The `AccelSensor` structure and the `thisAccel` declaration simply store the current three-axis accelerometer reading in integer counts (as read from the sensor's registers) and in floating point units of g. The integer and floating point members are related via the sensor and range-specific compile time constant `FCOUNTSPERG`.

```
#define FCOUNTSPERG 4096.0F    // MMA8451 and FXOS8700 provide 4096 counts / g in 2g mode

// 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)
};
```

3.4 MagSensor

The `MagSensor` structure and the `thisMag` declaration store the current three-axis magnetometer reading and derived values in integer counts (as read from the sensor's registers) and in floating point units of uT.

The members `iBpx`, `iBpy` and `iBpz` are the raw magnetometer readings. The locus of these measurements is an ellipsoid offset from the origin by the hard iron offset and with a shape determined by the soft iron matrix.

The members `iBcx`, `iBcy` and `iBcz` are the calibrated magnetometer readings (after correction by the hard and soft iron distortion computed by the calibration algorithms). Their locus is a sphere centered at the origin with a radius equal to the geomagnetic field strength.

The members `iBfx`, `iBfy` and `iBfz` are the calibrated magnetometer readings de-rotated in roll and pitch using the accelerometer reading. Their locus is an annulus (ring) offset at approximately constant iBfz.

The integer and floating point members are related via the compile time constants `FCOUNTSPERUT` and `FUTPERCOUNT`.

```
#define FCOUNTSPERUT 10.0F  // MAG3110 and FXOS8700 provide 10 counts / uT
#define FUTPERCOUNT 0.1F    // MAG3110 and FXOS8700 provide 0.1uT per count resolution
```

```
// 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)
    int16 iBcx;                 // calibrated magnetometer sensor output x (counts)
    int16 iBcy;                 // calibrated magnetometer sensor output y (counts)
    int16 iBcz;                 // calibrated magnetometer sensor output z (counts)
    int16 iBfx;                 // de-rotated magnetometer sensor output x (counts)
    int16 iBfy;                 // de-rotated magnetometer sensor output y (counts)
    int16 iBfz;                 // de-rotated 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)
    float fBcx;                 // calibrated magnetometer sensor output x (uT)
    float fBcy;                 // calibrated magnetometer sensor output y (uT)
    float fBcz;                 // calibrated magnetometer sensor output z (uT)
    float fBfx;                 // de-rotated magnetometer sensor output x (counts)
    float fBfy;                 // de-rotated magnetometer sensor output y (counts)
    float fBfz;                 // de-rotated magnetometer sensor output z (counts)
};
```

3.5 MagCalibration

The `MagCalibration` structure and the `thisMagCal` declaration hold details of the hard and soft iron calibration computed by the magnetic calibration algorithms. For additional information, refer to AN4684.

```
// magnetic calibration structure
struct MagCalibration
{
    float fVx;                  // x component of computed hard iron offset
    float fVy;                  // y component of computed hard iron offset
    float fVz;                  // z component of computed hard iron offset
    float fB;                   // computed geomagnetic field magnitude in uT
    float fFitErrorpc;          // computed fit error %
    float ftrVx;                // trial value of x component of hard iron offset
    float ftrVy;                // trial value of y component of hard iron offset
    float ftrVz;                // trial value of z component of hard iron offset
    float ftrB;                 // trial value of geomagnetic field magnitude in uT
    float ftrFitErrorpc;        // trial value of fit error %
    int32 iValidMagCal;         // valid magnetic calibration flag
    float xfinvW[3][3];         // estimated inverse soft iron matrix size
    float *finvW[3];
    float xfA[3][3];            // estimated ellipsoid matrix A
    float *fA[3];
    float xinvA[3][3];          // inverse of ellipsoid matrix A
    float *finvA[3];
    float xftrinvW[3][3];       // trial computed inverse soft iron matrix size
    float *ftrinvW[3];
};
```

3.6 SV6DOF

The `SV6DOF` structure and the `thisSV6DOF` declaration hold the instantaneous and low pass filtered smartphone orientation state vector in terms of Euler angles (roll, pitch, yaw and compass heading), orientation matrix and quaternion. For additional information, refer to AN4676.

```
// 6DOF orientation structure definition
struct SV6DOF
{
```

```
    // Euler angles
    float fPhi6DOF;                  // 6DOF roll (deg)
    float fThe6DOF;                  // 6DOF pitch (deg)
    float fPsi6DOF;                  // 6DOF yaw (deg)
    float fRho6DOF;                  // 6DOF compass (deg)
    float fLPPhi6DOF;                // 6DOF low pass roll (deg)
    float fLPThe6DOF;                // 6DOF low pass pitch (deg)
    float fLPPsi6DOF;                // 6DOF low pass yaw (deg)
    float fLPRho6DOF;                // 6DOF low pass compass (deg)
    float fDelta6DOFn;               // 6DOF inclination angle (deg) sample n
    float fDelta6DOFnm1;             // 6DOF inclination angle (deg) sample n-1
    float fDelta6DOFnm2;             // 6DOF inclination angle (deg) sample n-2
    float fLPDelta6DOFn;             // 6DOF low pass inclination angle (deg) sample n
    float fLPDelta6DOFnm1;           // 6DOF low pass inclination angle (deg) sample n-1
    float fLPDelta6DOFnm2;           // 6DOF low pass inclination angle (deg) sample n-2
    // orientation matrices and quaternions
    float xfR6DOFn[3][3];            // 6DOF rotation matrix for current sample n
    float *fR6DOFn[3];
    float xfR6DOFnm1[3][3];          // 6DOF rotation matrix for sample n-1
    float *fR6DOFnm1[3];
    float xfR6DOFnm2[3][3];          // 6DOF rotation matrix for sample n-2
    float *fR6DOFnm2[3];
    float xfLPR6DOFn[3][3];          // low pass 6DOF rotation matrix for sample n
    float *fLPR6DOFn[3];
    float xfLPR6DOFnm1[3][3];        // low pass 6DOF rotation matrix for sample n-1
    float *fLPR6DOFnm1[3];
    float xfLPR6DOFnm2[3][3];        // low pass 6DOF rotation matrix for sample n-2
    float *fLPR6DOFnm2[3];
    struct fquaternion fq6DOFn;      // current 6DOF orientation quaternion
    struct fquaternion fLPq6DOFn;    // low pass 6DOF orientation quaternion
};
```

3.7 MagneticBuffer

The `MagneticBuffer` structure and the `thisMagneticBuffer` declaration hold magnetometer measurement for use by the magnetic calibration algorithms. For additional information, refer to AN4684.

```
// magnetometer measurement buffer
struct MagneticBuffer
{
     int16 iBx[MAGBUFFSIZE][MAGBUFFSIZE][MAGBUFFSIZE];   // array of x magnetic fields
     int16 iBy[MAGBUFFSIZE][MAGBUFFSIZE][MAGBUFFSIZE];   // array of y magnetic fields
     int16 iBz[MAGBUFFSIZE][MAGBUFFSIZE][MAGBUFFSIZE];   // array of z magnetic fields
     int32 index[MAGBUFFSIZE][MAGBUFFSIZE][MAGBUFFSIZE]; // array of time indices
     int32 iMagBufferCount;                              // number of magnetometer readings
};
```

4 Compile Time Constants

4.1 User adjustable

These values are specific to the sensors being used and the ranges selected.

```
// sensor scaling constants
#define FCOUNTSPERG 4096.0F   // MMA8451 and FXOS8700 provide 4096 counts / g in 2g mode
#define FCOUNTSPERUT 10.0F    // MAG3110 and FXOS8700 provide 10 counts / uT
#define FUTPERCOUNT 0.1F      // MAG3110 and FXOS8700 provide 0.1uT per count resolution
```

The sampling interval DELTAT must match the timing of the main control loop. ANGLE_LPF_FPU should be selected to give the required low pass filtering of the output Euler angles. The default setting is for a low pass cutoff at Nyquist / 16 seconds. Decreasing ANGLE_LPF_FPU will make the eCompass heading more jittery but with faster response and vice versa.

```
// orientation LPF cutoff = Nyquist / ANGLE_LPF_FPU
// so about ANGLE_LPF_FPU samples response
// ANGLE_LPF_FPU equal to 1.0F is valid and corresponds to all pass.
#define ANGLE_LPF_FPU 1.0F
// sampling interval (secs)
#define DELTAT 0.02F
```

The default magnetic buffer size is 6x6x6=216 entries. If RAM is critical in the application, then 5 x 5 x 5 = 125 entries is an alternative provided that MINEQUATIONS, MEDEQUATIONS and MAXEQUATIONS are reduced proportionally.

The magnetic calibration algorithms are initially called every INITIALCALINTERVAL iterations of the main control loop and then less often every FINALCALINTERVAL when the magnetic buffer has sufficient measurements to produce a reliable calibration. At a sampling rate of 50 Hz, the default value of FINALCALINTERVAL of 1500 iterations corresponds to 30 seconds between calibrations. This interval could be increased to one minute or longer to reduce power.

The first magnetic calibration is performed when MINEQUATIONS entries are present in the magnetic buffer. This has been found to give a reasonable first estimate of the magnetic calibration with minimum movement of the smartphone is a figure-eight motion to gather measurements. If the preference is for a more reliable estimate at the expense of more motion of the smartphone, then this could be increased to 32 measurements.

Any new calibration computed by the calibration algorithms is always accepted as up to MEDEQUATIONS entries become available in the magnetic buffer. The logic here is that i) more measurements will generally improve the calibration and ii) a low fit error can result from an initial calibration based on just a small number of noisy magnetometer measurements as a result of the calibration tending to fit itself around the noise. By always accepting the magnetic calibration solution for up to MEDEQUATIONS measurements, "lockout" by a low fit initial estimate is avoided. As an analogy, fitting a straight line to just two points (however noisy) on a piece of paper will always be a perfect fit to the two points (even if inaccurate due to noise) while the straight line fit to three or more noisy data points will generally not be a perfect fit even if more accurate.

The constant MAXEQUATIONS determines the maximum number of measurements that will ever be used for magnetic calibration. If quality of calibration is important, then this could be increased but, in practice, little advantage is found above 120 measurements. If this value is increased, it should always be less than the 6 x 6 x 6 = 216 measurements in the magnetic buffer to allow room for old measurements to age out of use in the calibration. A maximum of 180 is probably sensible for a 6x6x6 magnetic measurement buffer.

The constant FITERRORAGING is used to prevent "lockout" by an earlier calibration with particularly low fit error. As an example, if the smartphone eCompass was calibrated with an extremely low fit error in outdoor conditions of -20° C and then moved, without powering down, to a warmer indoor temperature, the magnetic calibration would need to change but might produce another excellent calibration but with a very slightly larger fit error. Some means is needed to artificially age a calibration so that it will eventually always be replaced by a more recent calibration. The default value of 0.0033333 corresponds to a time period of order 300 seconds (not iterations) or 5 minutes over which an existing calibration fit error will exponentially increase. This increase continues until the point where the calibration fit error is guaranteed to be replaced with a more recent calibration.

```
// magnetic calibration constants
#define MAGBUFFSIZE 6                // magnetic buffer size: 6 implies 6^3 = 216 entries
#define INITIALCALINTERVAL 50        // 1s at 50Hz: initial calibration interval
#define FINALCALINTERVAL 1500        // 30s at 50Hz: normal calibration interval
#define MINEQUATIONS 24              // minimum number of measurements used for calibration
#define MEDEQUATIONS 80              // new solution always accepted up to this point
#define MAXEQUATIONS 120             // maximum number of measurements used for calibration
#define FITERRORAGING 0.0033333F     // reciprocal of time (s) for fit error to increase by
e=2.718
```

4.2 Not user adjustable

The two constants FMATRIXSCALING and FINVMATRIXSCALING can remain unchanged irrespective of the local geomagnetic field. Their role is to approximately normalize values in the matrices used in eigenanalysis to reduce numerical rounding errors. In practice, numerical round-off error has never been found to be a problem in the eigenanalysis function eigencompute since the matrices are only size 7 x 7 (for the 7 element calibration) and 10 x 10 (for the 10 element calibration).

```
// multiplicative conversion constants
#define PI 3.141592654F                 // Pi
#define FDEGTORAD 0.01745329251994F     // degrees to radians conversion = pi / 180
#define FRADTODEG 57.2957795130823F     // radians to degrees conversion = 180 / pi
#define FRECIP180 0.0055555555555F      // multiplicative factor 1/180
#define FMATRIXSCALING 0.02F            // approx normalizes geomagnetic field 50uT
#define FINVMATRIXSCALING 50.0F         // inverse of FMATRIXSCALING
```

The two values MINBFIT and MAXBFIT are used as a defensive test against deliberate jamming by placing a powerful magnet against the smartphone. This will cause the magnetometer to clip and the calibration algorithms will compute a fitted geomagnetic field hundreds of times larger than the true field. This will result in a low calibration fit error since the fit error is normalized to the estimated geomagnetic field. The calibration is therefore rejected if the fitted geomagnetic field is outside the range MINBFIT to MAXBFIT. These are set to a very wide range encompassing any valid geomagnetic field found on the earth's surface and are best left alone.

```
#define MINBFIT 5.0F          // minimum acceptable geomagnetic field B (uT) for valid
calibration
#define MAXBFIT 100.0F                  // maximum acceptable geomagnetic field B (uT) for
valid calibration
```

The largest matrix inverted in the software supplied is 4x4 in the 4 parameter calibration function. The larger size MAXMATINV=6 allowing up to 6x6 matrix inverses is provided for future enhancements to the software.

```
// matrix inverse integer arrays
#define MAXMATINV 6                      // maximum size supported in matrix inverse function
```

4.3 Not required in final code

Disc file input and output will not be present in a final embedded system.

```
// file access buffer size
#define BUFFSIZE 2048
```

The final product will use software built for just one coordinate system and there will be no requirement to test for the coordinate system in use. If the product uses the Android coordinate system, for example, then all functions for the Aerospace/NED and Windows 8 coordinate systems can be deleted to reduce program memory size.

```
// coordinate system and eCompass option selected
#define NED 0             // identifier for NED angle output
#define ANDROID 1         // identifier for Android angle output
#define WIN8 2            // identifier for Windows 8 angle output
```

5 Separating the Magnetic Calibration as a Background Task

Advanced users of the software using a Real Time Operating System may wish to place the calibration process as a low priority background task running in parallel with the 50 Hz (typical) high priority eCompass task. This is very straightforward and the approach below has been verified by Freescale on the MQX operating system.

The problem is greatly simplified by noting that the magnetic data buffer structure thisMagneticBuffer is only written by the high priority 50 Hz eCompass task and is only read by the low priority magnetic calibration task. Similarly, the magnetic calibration structure `thisMagCal` is only read by the eCompass task and is only written by the magnetic calibration task. These structures can be reliably shared between the two tasks without the overhead of double buffering.

Since the eCompass task is the higher priority, it will never be swapped out by the magnetic calibration task so there is no risk of the magnetic calibration changing halfway through the eCompass task. In fact, even it did, this would be no problem.

The first step of the magnetic calibration algorithms is to extract measurements from the magnetic measurement buffer and construct matrices from which the calibration parameters are computed. Once the matrices are constructed, there is no further interaction with the magnetic calibration buffer and it does not matter that this structure may be updated multiple times during processing of the matrices.

The key line of code in the magnetic calibration algorithm is the test:

```
if (pthisMagneticBuffer->index[j][k][l] != -1)
```

If this test fails (the entry is invalid with flag -1) and the calibration process is swapped out, and the buffer entry is filled with valid data, then this is no problem since the data is simply not read during this magnetic calibration.

If the test passes because there is valid data which is subsequently invalidated (due to aging for example) while the magnetic process is swapped out, then this is also no problem since the magnetic measurements are not over-written when the data is flagged as invalid and remain valid measurements.

If the test passes and this buffer entry is over-written with a new measurement, then the worst that happens is that, as an example, the magnetic calibration uses the x component of the magnetic field from one iteration and the y and z components from a different iteration. Since the magnetic buffer elements are indexed by orientation pseudo-angles, these measurements will be very similar and have a trivial effect on the resulting calibration.

Note that the calibration algorithms compute a local sum of how many measurements they have extracted from the magnetic buffer in the local variable ilocalMagBufferCount . This provides robustness against the actual real time number of measurements stored in `thisMagneticBuffer.iMagBufferCount` changing due to updates by the high priority 50Hz process.

There are no shared working arrays between the high priority eCompass task and the background 7 and 4 element calibration algorithms, as can be verified by their function prototypes:

```
void fUpdateCalibration7EIG(struct MagCalibration *pthisMagCal,
    struct MagneticBuffer *pthisMagneticBuffer,
    float **ftmpA7x7, float **ftmpB7x7, float **ftmpA7x1)

void fUpdateCalibration4INV(struct MagCalibration *pthisMagCal,
    struct MagneticBuffer *pthisMagneticBuffer,
    float **ftmpA4x4, float **ftmpB4x4, float **ftmpA4x1,
    float **ftmpB4x1, int32 **icolind, int32 **irowind, int32 **ipivot)
```

Users of the 10 element calibration code should note that this shares the two working arrays `ftmpA3x3` and `ftmpA3x1` with the eCompass task. If the 10 element calibration code is placed as background task then it should be provided with two new working arrays, say `ftmpB3x3` and `ftmpB3x1`, which are not used by the eCompass task.

```
void fUpdateCalibration10EIG(struct MagCalibration *pthisMagCal,
struct MagneticBuffer *pthisMagneticBuffer, float **ftmpA10x10,
float **ftmpB10x10, float **ftmpA10x1, float **ftmpA3x3, float **ftmpA3x1)
```

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: AN4700
Rev. 1.0, 4/2013