DT0103
Design tip

Compensating for magnetometer installation error and hard-iron effects using accelerometer-assisted 2D calibration

By Andrea Vitali

Main components	
LSM303AGR	Ultra-compact high-performance eCompass module: ultra-low-power 3D accelerometer and 3D magnetometer
LSM303AH	Ultra-compact high-performance eCompass module: ultra-low-power 3D accelerometer and 3D magnetometer
LIS2MDL	Digital output magnetic sensor: ultra-low-power, high-performance 3-axis magnetometer

Purpose and benefits

This design tip explains how to perform 2D calibration (rotation in a plane) to compensate for magnetometer installation error (roll and pitch) and hard-iron effects (offset). After compensation, tilt-compensated eCompass output (yaw angle) is correct. Compensation is done before the fusion, by subtracting the offset and de-rotating the magnetometer data. The accelerometer can be used to verify the quality of 2D calibration and to enable the calibration in a tilted plane (not horizontal).

Benefits:

- Compensates for misalignment of the magnetometer sensor (installation error) and hard-iron effects (offset) by performing 2D calibration (rotation in a plane).
- Adds functionality with respect to data fusion provided by the MotionFX library (included in X-CUBE-MEMS1) which provides 9-axis Acc+Mag+Gyro and 6-axis Acc+Gyro fusion with accelerometer vibration rejection, gyroscope bias compensation, and magnetometer hard-iron (offset) compensation, enabled by 3D calibration.
- Adds functionality with respect to sensor calibration provided by the MotionMC library (included in X-CUBE-MEMS1) which compensates for hard-iron effects (offset) and soft-iron effects (sensitivity and cross-sensitivity) but not for installation error, and requires 3D calibration (rotation in space).
- Easy to use on every microcontroller – a reference MATLAB© implementation is presented here which can be easily translated to C code.

Introduction

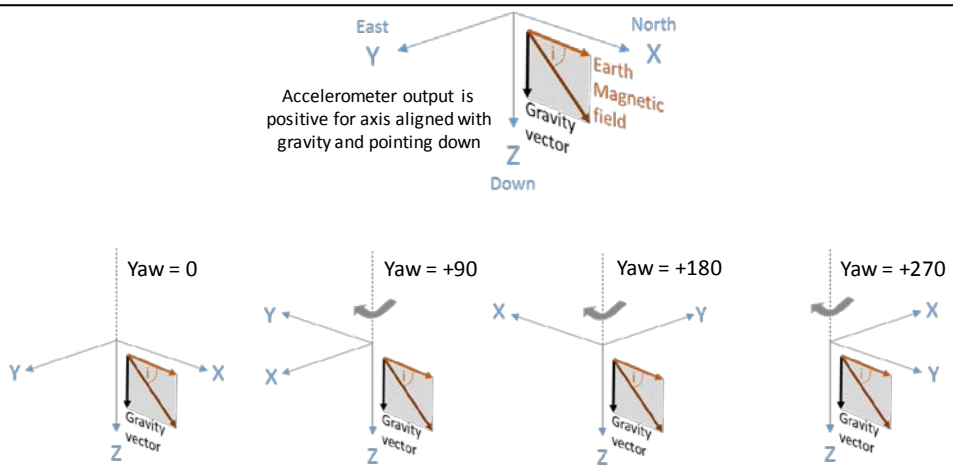
First, the ideal reference case is presented when no error occurs. Next, the consequences of installation error and hard-iron effects are discussed. Finally, the details of the algorithm (assumption and limitations, and performance verification) are explained.

What happens in the ideal reference case

The design tip DT0058 (Computing tilt measurement and tilt-compensated eCompass) explains how to compute tilt (roll and pitch angles) from accelerometer data and eCompass output (yaw angle) from magnetometer data. The eCompass output depends on the accelerometer output: magnetometer data must be tilt-compensated before it is used.

When the accelerometer and magnetometer are perfectly aligned, tilt-compensation is accurate and the eCompass output is correct. This can be verified by rotating the sensor platform, with the accelerometer and magnetometer, around the vertical axis: the output of the fusion is roll=0, pitch=0 and the correct yaw. However, in presence of any magnetometer installation error (non-zero roll and pitch of the magnetometer with respect to the accelerometer), the yaw will not be correct.

Figure 1. Reference orientation for input data from accelerometer and magnetometer, and reference orientation for output data: roll, pitch, yaw angles



Roll, Pitch, Yaw Degrees	Acc (Gravity vector) G = 1g = 1000mg	Mag (Earth mag field) B = 10..90uT = 100..900 mG
0, 0, **0**	0, 0, **+G**	**+B Cos(i),** 0, +B Sin(i)
0, 0, **+90**	0, 0, **+G**	0, **-B Cos(i),** +B Sin(i)
0, 0, **+180**	0, 0, **+G**	**-B Cos(i),** 0, +B Sin(i)
0, 0, **+270**	0, 0, **+G**	0, **+B Cos(i),** +B Sin(i)

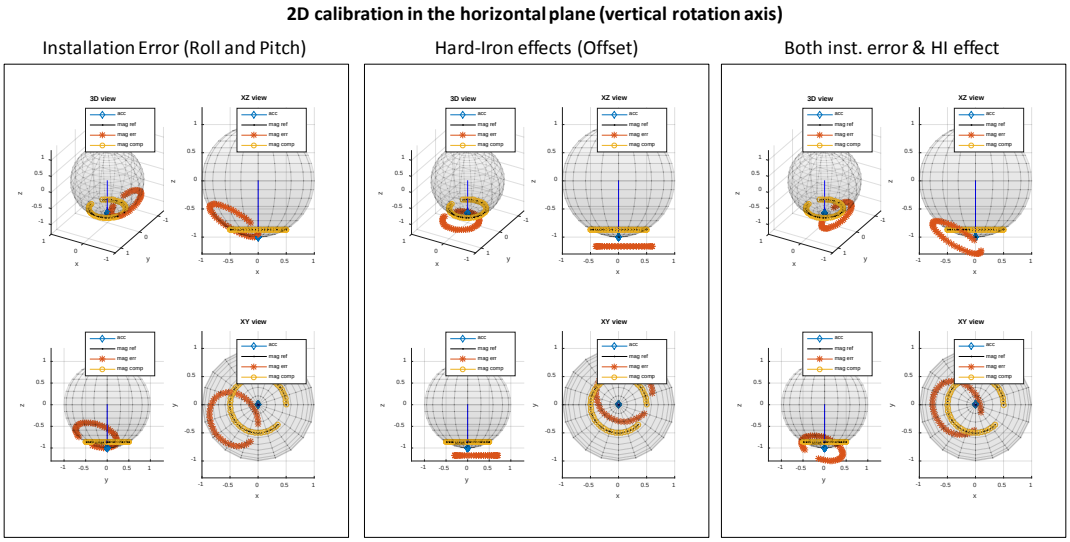
When one rotates the sensor platform around the vertical axis, accelerometer data will be constant and equal to [X=0, Y=0, Z=+1*g*], while magnetometer data will depend on the yaw angle and equal to [X = B cos(mi) cos(yaw), Y = -B cos(mi) sin(yaw), Z = B sin(mi)]. “B” is the magnetic field strength of the Earth, and “mi” is the inclination of the Earth’s magnetic field vector with respect to the horizontal plane. See figure 1.

What happens in presence of installation error and hard-iron effects

It is important to note that, when there is no installation error, magnetometer data will describe a circle lying in the horizontal plane (magnetometer X and Y output describe a circle, while Z output is constant). Also, if there are no hard-iron effects, the circle will be centered around the Z-axis.

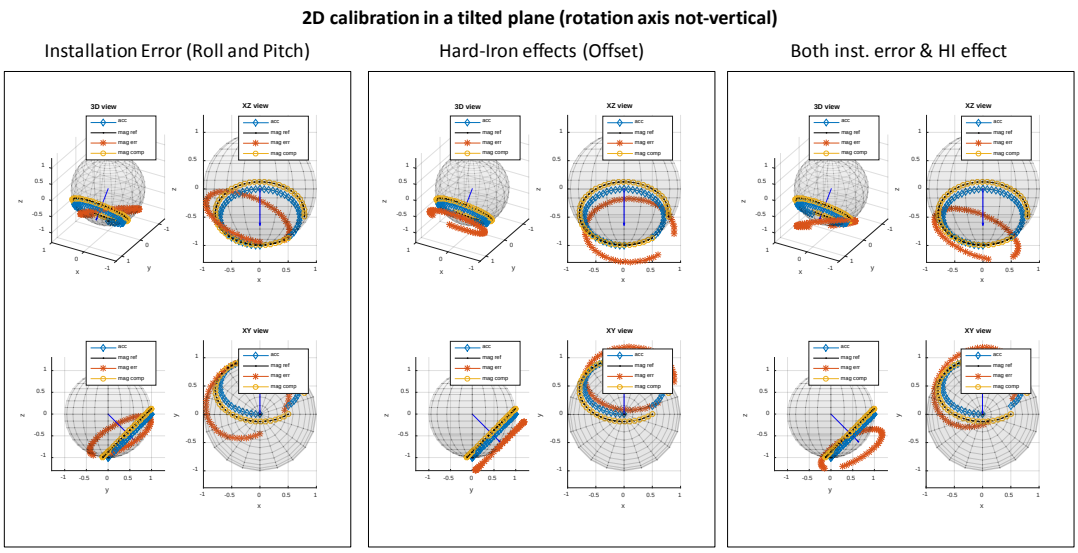
Conversely, if the magnetometer is not perfectly aligned with the accelerometer, magnetometer data will describe a circle lying in a tilted plane (magnetometer Z output is not constant). Also, if the magnetometer adds a spurious offset, magnetometer data will describe a circle which is not centered around the Z-axis. See figure 2.

Figure 2. 2D calibration in the horizontal plane (rotation axis is vertical), in presence of installation error (left), hard-iron effects (center) or both (right)



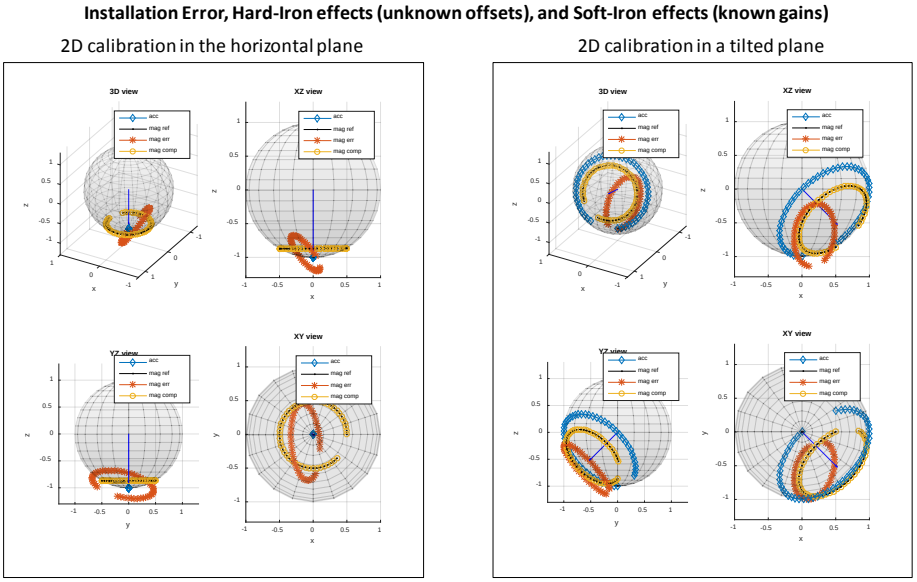
In figure 2, the rotation axis and accelerometer data are blue, reference error-free magnetometer data is black, magnetometer data affected by errors is red, and error-compensated magnetometer data is yellow. If compensation is successful, compensated data (yellow) does match the ideal reference data (black).

Figure 3. 2D calibration in a tilted plane (rotation axis is not vertical), in presence of installation error (left), hard-iron effects (center) or both (right)



In most cases, both errors (installation error and hard-iron effects) will be present. The algorithm will first compensate for the installation error, by de-rotating magnetometer data, so that the aforementioned circle is in the horizontal plane as expected; it will then compensate for hard-iron effects, by subtracting the offset, so that the center of the circle is on the Z-axis. If the rotation axis, as detected by the accelerometer, is not vertical, the algorithm will further de-rotate the magnetometer data to the corresponding tilted circle, where it would be in the ideal error-free case. See figure 3.

Figure 4. 2D calibration in the horizontal plane (left) and in a tilted plane (right) in presence of all errors: installation error, hard-iron effects, and soft-iron effects



Assumptions and algorithm limitations

The algorithm is able to complete the computation even if the data does not describe a full circle. A full 360 degrees rotation is not needed. However, a full rotation does maximize the quality of the computation.

Accelerometer data is essential when 2D calibration is performed on a tilted plane (rotation axis is not vertical). The accelerometer must be already calibrated (DT0053, 6-point tumble sensor calibration) and correctly installed (DT0076, Compensating for accelerometer installation error: zeroing pitch and roll for a reference orientation).

The magnetometer should already be calibrated (DT0059, Ellipsoid or sphere fitting for sensor calibration), and magnetometer data should already be compensated for soft-iron effects (axis gains, and cross-axis gains) and hard-iron effects (axis offsets). However, the algorithm presented here can work on imperfect data. See figure 4.

- There can be **unknown hard-iron effects** (axis offsets): if the Earth's magnetic field strength "B" and inclination "mi" are known, the algorithm presented here will be able to compute and compensate for the offsets. Even if hard-iron effects are already compensated, 2D calibration can still be used to refine the compensation

(as an example, this is useful when 3D calibration is no longer possible after device installation).

- There can be **known soft-iron effects** but the soft-iron matrix must be diagonal: coefficients out of the diagonal (cross-axis gains) must be zero; and coefficients along the diagonal (axis gains) must be known. This means that magnetometer data must not be affected by a spurious rotation induced by axis cross-talk in the sensor; and the algorithm presented here must be able to properly scale each axis to reduce the non-rotated ellipsoid to a sphere.

Generation of simulated data and performance verification

Accelerometer data is assumed to be normalized by dividing by 1*g*, so that the gravity vector as measured by the sensor is 1. Magnetometer data is also assumed to be normalized by dividing by B, so that the modulus of the Earth's magnetic field as measured by the sensor is 1. The only parameter for magnetometer data is "mi" the inclination of the Earth's magnetic field with respect to the horizontal plane (in the example script: mi = 60 degrees).

The parameters for errors are the following: roll and pitch for the installation error (Rerr = 20 deg, Perr = 30 deg in the example), hard-iron effects (HI = [0.1, -0.2, 0.3]), and soft-iron effects (SI = [0.5, 1.2, 0.9]).

The parameters for 2D calibration are the following: rotation axis (rotax = [0, 0.9, 0.9], not vertical), number of points in the simulation (N = 50), and rotation angles (0 to 315 deg, not a full circle).

Based on the specified parameters, three vectors are computed: accelerometer data (av), reference error-free magnetometer data (mv) and magnetometer data affected by errors (mve). Matrix rotations are applied first, then soft-iron effects, finally hard-iron effects.

Some Gaussian noise can be added (10 m*g* RMS and 5 mGauss RMS in the example script). If there is no noise, the estimated installation error and hard-iron effects will perfectly match the parameters of the simulation.

Description of 2D calibration and compensation

The description of the calibration and compensation algorithm goes from the simplest to the most complex case:

- 2D calibration in the horizontal plane (vertical rotation axis) and compensation of magnetometer **installation error** (roll and pitch); this is the simplest case; accelerometer data is not needed but nice to have.
- 2D calibration in the horizontal plane (vertical rotation axis) and compensation of magnetometer **installation error** (roll and pitch) and **hard-iron effects** (offset); accelerometer data is not needed but nice to have.
- 2D calibration in a **tilted plane** (any rotation axis, not only vertical) and compensation of magnetometer **installation error** (roll and pitch) and **hard-iron effects** (offset); this is the most complex case and accelerometer data is essential.

Estimation of rotation axis

This step is optional and it is needed only to verify that the rotation axis is vertical or to enable the calibration in a tilted plane.

The rotation axis is the normal to the plane of the accelerometer data (na). The normal is computed by the plane fitting function. Three different plane fitting functions are available, each with different robustness and complexity trade-off: **planefitls** (based on Least Square), **planefiteig** (based on Eigen values) and **planefiteig_simple** (simplified).

When the rotation axis is vertical, accelerometer data will be clustered around a single point, [0, 0, 1]. The result of the plane fitting function should be ignored in this case. In general, if the average distance between the barycenter and the data points is less than the RMS noise, it means that the rotation axis is vertical.

When the rotation axis is not vertical, the corresponding de-rotation matrix is computed (aderotM) and applied to the accelerometer data (av). One can verify the quality of the plane fitting by checking that the de-rotated accelerometer data (avc) is indeed in the horizontal plane: the range of Z values should be much less than the range of X and Y values.

2D calibration in the horizontal plane, compensation of installation error

The first step is to identify the plane corresponding to the magnetometer data by applying one of the aforementioned plane fitting functions. As mentioned for the accelerometer data, if the average distance of the barycenter and the data points is less than the RMS noise, data points are clustered, and the output of the plane fitting cannot be used.

When the output of the plane fitting functions is deemed reliable, the corresponding de-rotation matrix is computed (derotM) and applied to the magnetometer data (mv). As explained above for the accelerometer, one can verify the quality of the plane fitting by checking that the de-rotated magnetometer data (mvc) are indeed in the horizontal plane.

At this point, if the rotation axis is vertical, the installation error is fully compensated.

2D calibration in the horizontal plane, compensation of installation error and hard-iron effects

The second step is to identify the center of the circle/ellipsoid described by the de-rotated magnetometer data (mvc). Two different 2D fitting functions are provided with different capabilities, robustness and complexity trade-off: **ellipsoid_2D_fit** (for circle, ellipse or rotated ellipse) and **ellipsoid_2D_robust** (rotated ellipse only).

If there is no installation error, the X and Y coordinates of the center of the circle/ellipsoid (o2) are the offset to be subtracted. If there is an installation error, or if one wants to compute also the Z offset to be subtracted, the reference magnetic vector is needed (mref which depends on the parameter “mi”) to compute all the coordinates of the center (o3). The three offsets to be subtracted are computed by de-rotating the center (o3): this is the estimate of the hard-iron effects (HIE).

At this point, if the rotation axis is vertical, the installation error as well as the hard-iron effects are fully compensated.

2D calibration in a tilted plane, compensation of installation error and hard-iron effects

The third and final step is needed only if the rotation axis is not vertical. In this case the magnetometer data should be further de-rotated to be in the correct plane (the same plane as the accelerometer data).

The additional de-rotation will introduce a spurious yaw rotation which must be identified and compensated. The Newton method is utilized to search for the correction factor (Ycorr). The method has been modified for this specific application in order to complete the computation even in case of stationary points and to reject invalid solutions.

At this point, even if the rotation axis is not vertical, the installation error as well as hard-iron effects are fully compensated.

Reference MATLAB code

Euler, quaternion and rotation matrix conversion functions

```
function [euler] = acc2euler(acc)
  x=acc(:,1); y=acc(:,2); z=acc(:,3);
  r=atan2(y,z);
  p=atan(-x/(y.*sin(r) + z.*cos(r)));
  y=0;
  euler=[r p y];
end

function M = euler2rotM(euler)
  % North-East-Down reference frame
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  phi=euler(1); theta=euler(2); psi=euler(3);
  m11= cos(theta)*cos(psi);
  m12= cos(theta)*sin(psi);
  m13=-sin(theta);
  m21=sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi);
  m22=sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi);
  m23=sin(phi)*cos(theta);
  m31=cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi);
  m32=cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi);
  m33=cos(phi)*cos(theta);
  M = [m11,m12,m13; m21,m22,m23; m31,m32,m33];
end

function euler = rotM2euler(M)
  m11=M(1,1); m12=M(1,2); m13=M(1,3);
  m21=M(2,1); m22=M(2,2); m23=M(2,3);
  m31=M(3,1); m32=M(3,2); m33=M(3,3);
  m13=min(+1,m13); m13=max(-1,m13);
  theta=-asin(m13); % alternative value: pi-theta
  costh=cos(theta);
  if abs(costh)<0.001, % singularity
    % psi-phi = atan2(+m21,+m31) or atan2(-m32,+m22) at North theta=+pi/2
    % psi+phi = atan2(-m21,-m31) or atan2(-m32,+m22) at South theta=-pi/2
    sgnth=sign(theta);  phi=0; % phi can be anything in -pi..+pi
    %psi1=sgnth*(phi-atan2(m21*sgnth,m31*sgnth));
    %psi2=sgnth*(phi-atan2(-m32,+m22));
    psi=-sgnth*atan2(-m32,+m22); % if phi~=0 use one of the formulas above
  else
    psi=atan2(m12/costh,m11/costh);
    phi=atan2(m23/costh,m33/costh);
  end;
  euler=[phi theta psi]; % roll pitch yaw
end

function M = quat2rotM(Q)
  qw=Q(1); qx=Q(2); qy=Q(3); qz=Q(4);
  qw2=qw*qw; qx2=qx*qx; qy2=qy*qy; qz2=qz*qz;
  n=1/(qw2+qx2+qy2+qz2);
  m11=( qx2 -qy2 -qz2 +qw2)*n;
  m22=(-qx2 +qy2 -qz2 +qw2)*n;
  m33=(-qx2 -qy2 +qz2 +qw2)*n;
  t1=qx*qy; t2=qz*qw; m12=2*(t1+t2)*n; m21=2*(t1-t2)*n;
  t1=qx*qz; t2=qy*qw; m13=2*(t1-t2)*n; m31=2*(t1+t2)*n;
  t1=qy*qz; t2=qx*qw; m23=2*(t1+t2)*n; m32=2*(t1-t2)*n;
  M=[m11 m12 m13; m21 m22 m23;m31 m32 m33];
end
```

3D plane fitting functions

```
function [n,p] = planefitls(xyz) % xyz: Nx3 matrix, each line is a data point
  % least square solution, minimize distance along main axis
  p=mean(xyz,1); % p point in the plane
  x=xyz(:,1)-p(1); y=xyz(:,2)-p(2); z=xyz(:,3)-p(3);
  xx=sum(x.*x); yy=sum(y.*y); zz=sum(z.*z);
  xy=sum(x.*y); xz=sum(x.*z); yz=sum(y.*z);
  Dx=yy*zz-yz*yz; Dy=xx*zz-xz*xz; Dz=xx*yy-xy*xy; % use largest
                  D=abs(Dx); flag=0;
  if abs(Dy)>D, D=abs(Dy); flag=1; end;
  if abs(Dz)>D, D=abs(Dz); flag=2; end;
  if D==0, n=[0,0,0]; return; end;
  switch flag
    case 0, a=1;                b=(xz*yz-xy*zz)/Dx; c=(xy*yz-xz*yy)/Dx;
    case 1, a=(yz*xz-xy*zz)/Dy ; b=1;               c=(xy*xz-yz*xx)/Dy;
    case 2, a=(yz*xy-xz*yy)/Dz;  b=(xz*xy-yz*xx)/Dz; c=1;
  end
  n=[a,b,c]./sqrt(a*a+b*b+c*c); % normal to plane
end

function [n,p,B] = planefiteig(xyz) % xyz: Nx3 matrix, each line is a data point
  % least square solution, minimize orthonormal distance
  p=mean(xyz,1); % point in the plane
  xyz(:,1)=xyz(:,1)-p(1); xyz(:,2)=xyz(:,2)-p(2); xyz(:,3)=xyz(:,3)-p(3);
  [eigvect,eigval]=eig(xyz' * xyz); % eigen vectors of a 3x3 matrix
  n=eigvect(:,1)'; % normal to plane
  B=eigvect(:,2:end)'; % rows are an orthonormal basis for the plane
end

function [n,p,B] = planefiteig_simple(xyz) % xyz: Nx3 matrix, each line is a data point
  % least square solution, minimize orthonormal distance
  p=mean(xyz,1); % point in the plane
  x=xyz(:,1)-p(1); y=xyz(:,2)-p(2); z=xyz(:,3)-p(3);
  % XYZ' * XYZ, 3x3 matrix
  %   x1 x2 ...  * x1 y1 z1 = sum(x2) sum(xy) sum(xz)
  %   y1 y2 ...    x2 y2 z2   sum(xy) sum(yy) sum(yz)
  %   z1 z2 ...    :  :  :    sum(xz) sum(yz) sum(z2)
  xx=sum(x.*x); yy=sum(y.*y); zz=sum(z.*z);
  xy=sum(x.*y); xz=sum(x.*z); yz=sum(y.*z);
  [eigvect,eigval]=eig([xx xy xz; xy yy yz; xz yz zz]); % eigen vectors of a 3x3 matrix
  n=eigvect(:,1)'; % normal to plane
  B=eigvect(:,2:end)'; % rows are an orthonormal basis for the plane
end
```

2D circle/ellipse/rotated ellipse fitting functions

```
function [ofs,gain,rotM] = ellipsoid_2D_fit(XY,varargin)
% Fit an (non)rotated ellipsoid or sphere to a set of xy data points
% XY: N(rows) x 3(cols), matrix of N data points (x,y)
% optional flag f, default to 0 (fitting of rotated ellipsoid)
x=XY(:,1); y=XY(:,2); if nargin>1, f=varargin{1}; else f=0; end;
if     f==0, D=[x.*x,  y.*y, 2*x.*y, 2*x,2*y]; % any axes (rotated ellipsoid)
elseif f==1, D=[x.*x,  y.*y,         2*x,2*y]; % XYZ axes (non-rotated ellipsoid)
elseif f==2, D=[x.*x+y.*y,           2*x,2*y]; % and radius x=y (circle)
end;
v = (D'*D)\(D'*ones(length(x),1)); % least square fitting
if f==0, % rotated ellipsoid
  A = [ v(1) v(3) v(4); v(3) v(2) v(5); v(4) v(5) -1 ];
  ofs=-A(1:2,1:2)\[v(4);v(5)]; % offset is center of ellipsoid
  Tmtx=eye(3); Tmtx(3,1:2)=ofs'; AT=Tmtx*A*Tmtx'; % ellipsoid translated to (0,0,0)
  [rotM ev]=eig(AT(1:2,1:2)/-AT(3,3)); % eigenvectors (rotation) and eigenvalues (gain)
  gain=sqrt(1./diag(ev)); % gain is radius of the ellipsoid
else % non-rotated ellipsoid
  if     f==1, v = [ v(1) v(2) 0  v(3) v(4) ];
  elseif f==2, v = [ v(1) v(1) 0  v(2) v(3) ]; % sphere
  end;
  ofs=-(v(1:2).\v(4:5))'; % offset is center of ellipsoid
  rotM=eye(2); % eigenvectors (rotation), identity = no rotation
  g=1+(v(4)^2/v(1)+v(5)^2/v(2));
  gain=(sqrt(g./v(1:2)))'; % find radii of the ellipsoid (scale)
end

function [ofs,gain,rotM] = ellipsoid_2D_fit_robust(XY)
  % Fit a rotated 2D ellipsoid to a set of xy data points
  % XY: N(rows) x 2(cols), matrix of N data points (x,y)
  x=XY(:,1); y=XY(:,2);

  % translate to (0,0), subtract centroid, may increase accuracy
  mx=mean(x); x=x-mx;
  my=mean(y); y=y-my;

  D1 = [x.^2, x.*y, y.^2]; D2 = [x, y, ones(size(x))];
  S1 = D1'*D1; S2 = D1'*D2; S3 = D2'*D2;
  T = -inv(S3)*S2'; M = S1+S2*T;
  M = [M(3,:)./2; -M(2,:); M(1,:)./2];
  [e,eval] = eig(M);
  c = 4*e(1,:).*e(3,:)-e(2,:).^2;
  v1 = e(:,find(c>0));
  v = [v1; T*v1];
```

```
  % translate back to centroid, skip if centroid is not subtracted
  v4 = v(4)-2*v(1)*mx-v(2)*my;
  v5 = v(5)-2*v(3)*my-v(2)*mx;
  v6 = v(6)+v(1)*mx^2+v(3)*my^2+v(2)*mx*my-v(4)*mx-v(5)*my;
  v(4)=v4; v(5)=v5; v(6)=v6;

  ofs =[-2*v(1),-v(2);-v(2 ),-2*v(3)]\[v(4);v(5)];

  % here we use eig()
  A=[2*v(1),  v(2),  v(4); ...
       v(2),2*v(3),  v(5); ...
       v(4),  v(5),2*v(6)]/v(6);
  Tmtx=eye(3); Tmtx(3,1:2)=ofs'; AT=Tmtx*A*Tmtx';
  [rotM ev]=eig(AT(1:2,1:2)/-AT(3,3));
  gain=sqrt(1./diag(ev));

  % alternative: here we do not use eig()
  %T=A(1:2,1:2)/-AT(3,3); trT=T(1,1)+T(2,2); detT=T(1,1)*T(2,2)-T(1,2)*T(2,1);
  %ev1=(trT+sqrt(trT^2-4*detT))/2; ev2=(trT-sqrt(trT^2-4*detT))/2;
  %gain=sqrt(1./[ev2;ev1]);
  %TT=T-eye(2)*ev1; evect1=TT(1,:)/sqrt(TT(1,1)^2+TT(1,2)^2);
  %                 evect1=TT(2,:)/sqrt(TT(2,1)^2+TT(2,2)^2);
  %TT=T-eye(2)*ev2; evect2=TT(1,:)/sqrt(TT(1,1)^2+TT(1,2)^2);
  %                 evect2=TT(2,:)/sqrt(TT(2,1)^2+TT(2,2)^2);
  %rotM=[evect1',evect2'];
end
```

Auxiliary scripts

test_Ycorr.m

```
%---- compute only the coefficients of interest in M
% yawrotM = euler2rotM([0 0 -Ycorr]); % this has a simplified structure
% M = derotM*yawrotM*aderotM'; % actually only 1st row is needed
% RPY = rotM2euler(M); % goal is to make the yaw, RPY(3), equal to 0

% see euler2rotM([0 0 -y]) and definition of M above:
%   cy = cos(-y); sy = sin(-y); % used to compute M coefficients
%   M11 = cy*(a11*m11+a12*m12) + sy*(a12*m11-a11*m12) + a13*m13;
%   M12 = cy*(a21*m11+a22*m12) + sy*(a22*m11-a21*m12) + a23*m13;
%   M13 = cy*(a31*m11+a32*m12) + sy*(a32*m11-a31*m12) + a33*m13;

% see rotM2euler():
%   pitch theta = -asin(M13); % pitch is needed to check yaw
%   yaw   psi   = atan2(M12/cos(theta),M11/cos(theta)); % yaw must be 0
%   roll  phi   = atan2(M23/cos(theta),M33/cos(theta)); % roll is unnecessary

% for yaw (psi) to be 0:
% 1) M12 must be 0: used with Newtown method to find roots
% 2) M11*cos(asin(M13)) must be >0: used to check root validity

m11=derotM(1,1);  m12=derotM(1,2);  m13=derotM(1,3);
a11=aderotM(1,1); a12=aderotM(1,2); a13=aderotM(1,3);
a21=aderotM(2,1); a22=aderotM(2,2); a23=aderotM(2,3);
a31=aderotM(3,1); a32=aderotM(3,2); a33=aderotM(3,3);

%---- Newton method
done=0; maxiter=20; iter=1;
c1=(a21*m11+a22*m12); c2=(a22*m11-a21*m12); c3=a23*m13; y=0; % init
while ~done,
  y = mod(y,2*pi); if (y>pi), y=y-2*pi; end;
  cy = cos(y); sy = sin(-y);
  f  = cy*c1 + sy*c2 + c3; % M12 function
  fd = sy*c1 - cy*c2;      % M12' function derivative
  if abs(fd)<1e-2, y=y+pi/2; continue; end; % get out of stationary point
  yt = y - f/fd; % new root estimate
  if abs(y-yt)<1e-6, % root candidate: check validity
    M11 = cy*(a11*m11+a12*m12) + sy*(a12*m11-a11*m12) + a13*m13;
    M13 = cy*(a31*m11+a32*m12) + sy*(a32*m11-a31*m12) + a33*m13;
    if M11*cos(asin(M13))<0, y=y+pi; continue; end; % jump to other solution
    done=1;
  end;
  y=yt; iter=iter+1; if iter>maxiter, break; end;
end;
Ycorr=y;
```

test_plot.m

```
hold on; grid on; zoom on; colormap(gray); % NED -> plot(x,-y,-z)
plot3( av(:,1), -av(:,2), -av(:,3),'d-');
plot3( mv(:,1), -mv(:,2), -mv(:,3),'k.-');
plot3(mve(:,1),-mve(:,2),-mve(:,3),'*-');
plot3(mvc(:,1),-mvc(:,2),-mvc(:,3),'o-');
surf(XS,YS,ZS,'FaceAlpha',0.1,'EdgeColor',[0.1 0.1 0.1],'EdgeAlpha',0.1);
quiver3(0,0,0,na(1),-na(2),-na(3),'b'); % rotax by acc data
%quiver3(0,0,0,nme(1),-nme(2),-nme(3),'r');
legend('acc','mag ref','mag err','mag comp'); xlabel('x'); ylabel('y'); zlabel('z');
axis([-lim +lim -lim +lim -lim +lim]); axis equal;
```

Main script: algorithm for 2D magnetometer calibration

test.m

```
mi   = 60*pi/180;           % mag field vector inclination w.r.t horizontal plane
mref = [cos(mi);0;sin(mi)]; % mag field reference vector
aref = [0;0;1];             % acc gravity reference vector

Rerr = 0; Perr = 0; % default: no installation error
Rerr = 20; % mag pitch installation error (degrees)
Perr = 30; % mag roll installation error (degrees)
RPYerr=[Rerr,Perr,0]*pi/180; % mag installation error, Roll Pitch (Yaw=0)
rotMerr = euler2rotM(RPYerr); % corresponding rotation matrix, derotM should match this
%RPYerr2 = rotM2euler(rotMerr)); % reverse computation

HI = [0,0,0]; SI = [1,1,1]; % default: no measurement errors
HI = [0.1,-0.2,0.3]; % Hard Iron effects (offset), OK if non-zero and unknown
SI = [0.5, 1.2, 0.9]; % Soft Iron effects (sensitivity), must be unity or known

rotax = [0,0,1]; % default: vertical rotation axis
rotax = [0,0.9,0.9]; % vector corresponding to the rotation axis
rotax = rotax./sqrt(sum(rotax.*rotax)); % normalization

N = 50; % points in the simulation
rv = linspace(0,(360-45)/180*pi,N); % vector of rotation angles
av = zeros(N,3); % vector of accelerometer data XYZ
mv = zeros(N,3); % vector of magnetometer data XYZ
mve = mv; % mag data with installation and measurement error

% simulation of calibration with rotation around rotax
for i = 1 : N,
  r2 = rv(i)/2; % current rotation angle
  rotM  = quat2rotM([cos(r2), sin(r2).*rotax]); % rotation matrix
  av(i,:) = rotM*aref; % acc
  mv(i,:) = rotM*mref; % mag without installation error
  mve(i,:) = (rotMerr*rotM )*mref; % mag with installation error
  mve(i,:) = (mve(i,:).*SI)+HI; % and measurement error (HI/SI)
end;

% add some Gaussian noise
aRMS = 0; mRMS = 0; % default: 0 RMS noise
aRMS = 0.010/1.0; % 10mg RMS noise / 1g gravity vector
mRMS = 0.005/0.5; % 5mGa RMS noise / 500mGa earth mag field vector
av  = av  + randn(size(av ))*aRMS;
mv  = mv  + randn(size(mv ))*mRMS;
mve = mve + randn(size(mve))*mRMS;

% estimation of rotation axis, should be vertical
avm = ones(N,1)*mean(av,1); % convenient matrix from the center of data points
avmd = sum(sqrt(sum((av-avm).^2,2)))/N; % planefit quality ~ avg distance w.r.t. center
[na,pa] = planefitls(av); % planefit[ls/eig/eig_simple]
if dot(na,[0,0,1])<0, na=-na; end; % choose min rotation w.r.t. vertical
if avmd<=(6*aRMS), na=[0,0,1]; end; % cluster of points indicates rotax is vertical
RPYrotax = acc2euler(na); % when rotax is not vertical Yaw does matter but it is not known
%RPYrotax(3) = Ycorr; % correction yaw is found in the final step
aderotM = euler2rotM(RPYrotax); % identity if rotax is vertical
avc = av*aderotM; % derotate acc data to see if it is in the horiz plane
avcX=max(avc(:,1))-min(avc(:,1)); % check quality of derotation to horiz plane
avcY=max(avc(:,2))-min(avc(:,2)); % avcZ must be much less than avcX,avcY
avcZ=max(avc(:,3))-min(avc(:,3)); % avcZ ~ 0 if there is no noise

% estimation and compensation of inst.err, SI must be unity or known
mvem = ones(N,1)*mean(mve,1); % convenient matrix from the center of data points
mvemd = sum(sqrt(sum((mve-mvem).^2,2)))/N; % sum of distances w.r.t. center
[nme,pme] = planefitls(mve); % planefit[ls/eig/eig_simple]
if dot(nme,[0,0,1])<0, nme=-nme; end; % choose min rotation w.r.t. vertical
RPYerrE = acc2euler(nme.*SI); % this is the estimated installation error
derotM = euler2rotM(RPYerrE); % corresponding derotation matrix
SIc = ones(N,1)*(1./SI); % convenient matrix for SI compensation
%mvc = mve*derotM; % compensation of inst.err only, when SI is unity
mvc = ((mve.*SIc)*derotM); % compensation of SI and inst.err, vert rotax, step (a)
mvcX=max(mvc(:,1))-min(mvc(:,1)); % check quality of derotation to horiz plane
mvcY=max(mvc(:,2))-min(mvc(:,2)); % mvcZ must be much less than mvcX,mvcY
mvcZ=max(mvc(:,3))-min(mvc(:,3)); % mvcZ ~ 0 if there is no noise

% estimation and compensation of HI, to be done after inst.err compensation
[o2,g2,rotM2] = ellipsoid_2D_fit(mvc,0); % 2=circle, 1=ellipse, 0=rotated ellipse
%[o2,g2,rotM2] = ellipsoid_2Dfit_robust(mvc); % alternative for rotated ellipse
mref2 = aderotM' * mref; % compensate for non-vertical rotation axis, inv(rotM)=rotM'
o3  = [o2;mean(mvc(:,3))-mref2(3)]; % this is the point corresponding to (0,0,0)
HIE = (derotM*o3)'.*SI; % this is the estimated HI, SI must unity or known
HIc = (ones(N,1)*HIE); % convenient matrix for HI compensation
mvc = ((mve-HIc).*SIc)*derotM; % compensation of HI/SI and inst.err, vert rotax, opt step (b)

% finalize compensation, to be done after (a) or (a)+(b) when rotax is not vertical
test_Ycorr; % search for Ycorr, to be done when rotax is not vertical
yawrotM = euler2rotM([0 0 -Ycorr]); % this will make RPYerr_final(3)=0
derotM = derotM * yawrotM * aderotM'; % compensate for non-vertical rotation axis
RPYerrE = rotM2euler(derotM); % recompute installation error
mvc = ((mve-HIc).*SIc)*derotM; % compensation of HI/SI and inst.err, any rotax

% plot section: 3D and 2D views
[XS,YS,ZS]=sphere(20); % reference sphere for XYZ data plot
```

```
lim = 1.3; % range of 3D plot, used in data plot
figure;
subplot(2,2,1); test_plot; view(210,30); title('3D view');
subplot(2,2,2); test_plot; view(0,0); title('XZ view');
subplot(2,2,3); test_plot; view(90,0); title('YZ view');
subplot(2,2,4); test_plot; view(0,90); title('XY view');

% print section
fprintf('reliability of acc data for planefit: %.2g >= 6*RMS: %.2g, ratio
%.2g\n',avmd,6*aRMS,avmd/(6*aRMS));
fprintf('quality of rotax estim. from acc derotation: Zdiff %.2g << Xdiff,Ydiff %.2g %.2g, ratio
%.2g %.2g\n',avcZ,avcX,avcY,avcX/avcZ,avcY/avcZ);
fprintf('rotation axis              : %.3f %.3f %.3f\n',rotax(1),rotax(2),rotax(3));
fprintf('rotation axis estim. from acc: %.3f %.3f %.3f\n',na(1),na(2),na(3));
fprintf('\n');

fprintf('reliability of mag data for planefit: %.2g >= 6*RMS: %.2g, ratio
%.2g\n',mvemd,6*mRMS,mvemd/(6*mRMS));
fprintf('quality of inst.err estim. from mag derotation: Zdiff %.2g << Xdiff,Ydiff %.2g %.2g,
ratio %.2g %.2g\n',mvcZ,mvcX,mvcY,mvcX/mvcZ,mvcY/mvcZ);
fprintf('yaw derotation correction (function of mag and acc planes): %.4f\n',Ycorr);
fprintf('inst.err true (RPY in rad) %.4f %.4f %.4f\n',RPYerr(1),RPYerr(2),RPYerr(3));
fprintf('inst.err estimated (in rad) %.4f %.4f %.4f\n',RPYerrE(1),RPYerrE(2),RPYerrE(3));
fprintf('\n');

fprintf('Hard Iron offset error  %.4f %.4f %.4f\n',HI(1),HI(2),HI(3));
fprintf('Hard Iron estimated     %.4f %.4f %.4f\n',HIE(1),HIE(2),HIE(3));
fprintf('Soft Iron gain error  %.4f %.4f %.4f\n',SI(1),SI(2),SI(3));
```

Support material

Related design support material
Software expansion, X-CUBE-MEMS1, Sensor and motion algorithm software expansion for STM32Cube
Software function pack, FP-SNS-MOTENV1, STM32 ODE function pack for IoT node with BLE connectivity and environmental and motion sensors
Software function pack, FP-SNS-ALLMEMS1, STM32 ODE function pack for IoT node with BLE connectivity, digital microphone, environmental and motion sensors
Evaluation kit, STEVAL-STLKT01V1, SensorTile development kit
Documentation
Design Tip, DT0053, 6-point tumble sensor calibration
Design Tip, DT0058, Computing tilt measurement and tilt-compensated eCompass
Design Tip, DT0059, Ellipsoid or sphere fitting for sensor calibration
Design Tip, DT0060, Exploiting the gyroscope to update tile measure and eCompass
Design Tip, DT0076, Compensating for accelerometer installation error: zeroing pitch and roll for a reference orientation

Revision history

Date	Version	Changes
28-Aug-2018	1	Initial release

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.