DT0076
Design tip

Compensating for accelerometer installation error: zeroing pitch and roll for a reference orientation

By Andrea Vitali

Main components	
LSM303AGR	Ultra compact high-performance e-compass: ultra-low-power 3D accelerometer and 3D magnetometer module
LSM6DSL	iNEMO inertial module: 3D accelerometer and 3D gyroscope
LIS2DH12	MEMS digital output motion sensor: ultra low-power high performance 3-axes femto accelerometer
LIS2DS12	MEMS digital output motion sensor ultra low-power high performance 3-axes "pico" accelerometer
H3LIS331DL	MEMS motion sensor: low power high g 3-axis digital accelerometer

Purpose and benefits

This design tip explains how to compensate for the accelerometer installation error. After compensation, pitch and roll angles, are zero for the reference orientation. Compensation can be done before fusion, by de-rotating input accelerometer data, or it can be done after fusion by de-rotating the output quaternion.

Benefits:

- Compensates for misalignment of silicon sensor with respect to plastic package, misalignment of plastic package with respect to PCB board, misalignment of PCB board with respect to device housing, and misalignment of device with respect to correct reference orientation (installation error).
- Adds functionality with respect to data fusion provided by osxMotionFX library which provides 9-axis Acc+Mag+Gyro and 6-axis Acc+Gyro fusion but not 6-axis Acc+Mag.
- Easy to use on every microcontroller (osxMotionFX can only be run on STM32 and only when the proper license has been issued by Open.MEMS license server).

Description of compensation for installation error and misalignments

The device is installed and moved until it is in the reference orientation. Fusion output should give 0 degrees for roll and pitch angles.

However, because of installation error and misalignments, fusion output will be different than 0 degree tilt.

There are three options to compensate for the installation error and misalignment:

1. Subtraction of reference Pitch and Roll, measured in reference orientation. This gives acceptable results only if installation error is small and only if angles to be corrected are small.

2. De-rotation of input data vector, so that the output of the fusion will give 0 in the reference orientation. This gives perfect compensation in all cases.

3. De-rotation of output quaternion, so that the output of fusion is corrected to give 0 in the reference orientation. This gives perfect compensation in all cases.

Compensation by subtraction of reference Pitch and Roll (not always effective)

Step 1, calibration: the device is installed and activated. The output of the fusion in this reference position, reference Roll (Phi) and reference Pitch (Theta), is stored in memory and used later to perform the compensation.

Step 2, compensation: the device is active and operating as usual. The output of the fusion is corrected by subtracting the reference Roll and reference Pitch recorded in the previous step. This will give acceptable accuracy only if the corrections to be applied is small and only if angles to be corrected are small.

If the output is a quaternion, it can be converted to Euler angles using the following function:

```
function [phi,theta,psi]=quat2euler(Q)
  % North-East-Down reference frame (see Figure)
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  qw=Q(1); qx=Q(2); qy=Q(3); qz=Q(4);
  qw2=qw*qw; qx2=qx*qx; qy2=qy*qy; qz2=qz*qz;
  m=qw2+qx2+qy2+qz2; th=0.5*m-0.01; t=qw*qy-qz*qx; % 0.01 tolerance for singularity
  if(m<=0.0),  psi=0.0;               theta=0.0;   phi=0.0; return; end; % error
  if(t>=+th),  psi=-2.0*atan2(qx,qw); theta=+pi/2; phi=0.0; return; end; % North
  if(t<=-th),  psi=+2.0*atan2(qx,qw); theta=-pi/2; phi=0.0; return; end; % South
  psi=atan2(2.0*(qw*qz+qx*qy),qw2+qx2-qy2-qz2);
  t=2.0*t/m; if(abs(t)>1.0), theta=0.0; else theta=asin(t); end;
  phi=atan2(2.0*(qw*qx+qy*qz),qw2-qx2-qy2+qz2);
end
```

North and South singularities: if Theta = +/-90 deg, Phi and Psi will describe a rotation around the same vertical axis (one degree of freedom is lost, this is also known as gimbal lock); when converting back and forth, the sum Psi+Phi will be correct when Theta=-90 deg, and the difference Psi-Phi will be correct when Theta=+90 deg.

Compensation by de-rotation of input data vector (always effective)

Step 1, calibration: the device is installed and activated. The output of the fusion in this reference position, Euler angles or quaternion, is converted to a de-rotation matrix, which is stored in memory and used later to perform the compensation.

The de-rotation matrix is the inverse or transpose of the rotation matrix. This is the code and the functions to compute the de-rotation matrix from the Euler angles:

```
M = derotM(euler2rotM(phi,theta,psi); % de-rotation matrix

function MD = derotM(M)
  % inverse is the same as transpose, inv(M) == M'
  MD = [M(1,1),M(2,1),M(3,1); M(1,2),M(2,2),M(3,2); M(1,3),M(2,3),M(3,3)];
end

function M = euler2rotM(phi,theta,psi)
  % North-East-Down reference frame (see Figure)
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  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
```

The de-rotation matrix is computed from the conjugate quaternion. This is the code and the functions to compute the de-rotation matrix from the quaternion:

```
M = derotM(quat2rotM(Q)); % inverse/transpose rotation from quaternion
M = quat2rotM(quat2conj(Q)); % same as: rotation matrix from conjugate quaternion

function [QC] = quat2conj(Q)
  QC = [Q(1),-Q(2),-Q(3),-Q(4)];
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
```

Step 2, compensation: the device is active and operating as usual. The input data vector for the fusion is corrected by multiplication by the de-rotation matrix. De-rotation should be done for both the accelerometer data vector and the magnetometer data vector. After de-rotation, the data can be fed into the fusion algorithm.

```
function [xr,yr,zr] = rotMbyvect(M,x,y,z)
  xr = M(1,1)*x + M(1,2)*y + M(1,3)*z;
  yr = M(2,1)*x + M(2,2)*y + M(2,3)*z;
  zr = M(3,1)*x + M(3,2)*y + M(3,3)*z;
end
```

Compensation by de-rotation of output quaternion (always effective)

Step 1, calibration: the device is installed and activated. The output of the fusion in this reference position, Euler angles or quaternion, is converted to a de-rotation quaternion, which is stored in memory and used later to perform the compensation.

The de-rotation quaternion is the conjugate quaternion. If the output is Euler angles, it can be converted to de-rotation quaternion using the following function:

```
Q = quat2conj(euler2quat(phi,theta,psi));

function Q = euler2quat(phi,theta,psi)
  % North-East-Down reference frame (see Figure)
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  cosr2=cos(phi/2);   sinr2=sin(phi/2);
  cosp2=cos(theta/2); sinp2=sin(theta/2);
  cosy2=cos(psi/2);   siny2=sin(psi/2);
  qw = cosr2*cosp2*cosy2 + sinr2*sinp2*siny2;
  qx = sinr2*cosp2*cosy2 - cosr2*sinp2*siny2;
  qy = cosr2*sinp2*cosy2 + sinr2*cosp2*siny2;
  qz = cosr2*cosp2*siny2 - sinr2*sinp2*cosy2;
  Q=[qw qx qy qz];
end
```

Step 2, compensation: the device is active and operating as usual. The output of the fusion P is corrected by multiplication with the de-rotation quaternion Q. If the output of the fusion is Euler angles, it can be converted to a quaternion using the function listed in the first paragraph.

```
function R = quatbyquat(P,Q)
  pw=P(1); px=P(2); py=P(3); pz=P(4);
  qw=Q(1); qx=Q(2); qy=Q(3); qz=Q(4);
  rw = pw*qw - px*qx - py*qy - pz*qz;
  rx = pw*qx + px*qw + py*qz - pz*qy;
  ry = pw*qy - px*qz + py*qw + pz*qx;
  rz = pw*qz + px*qy - py*qx + pz*qw;
  R = [rw rx ry rz];
end
```

Simplified reference code

For convenience, a more compact function is presented here to compensate for the installation error and misalignments.

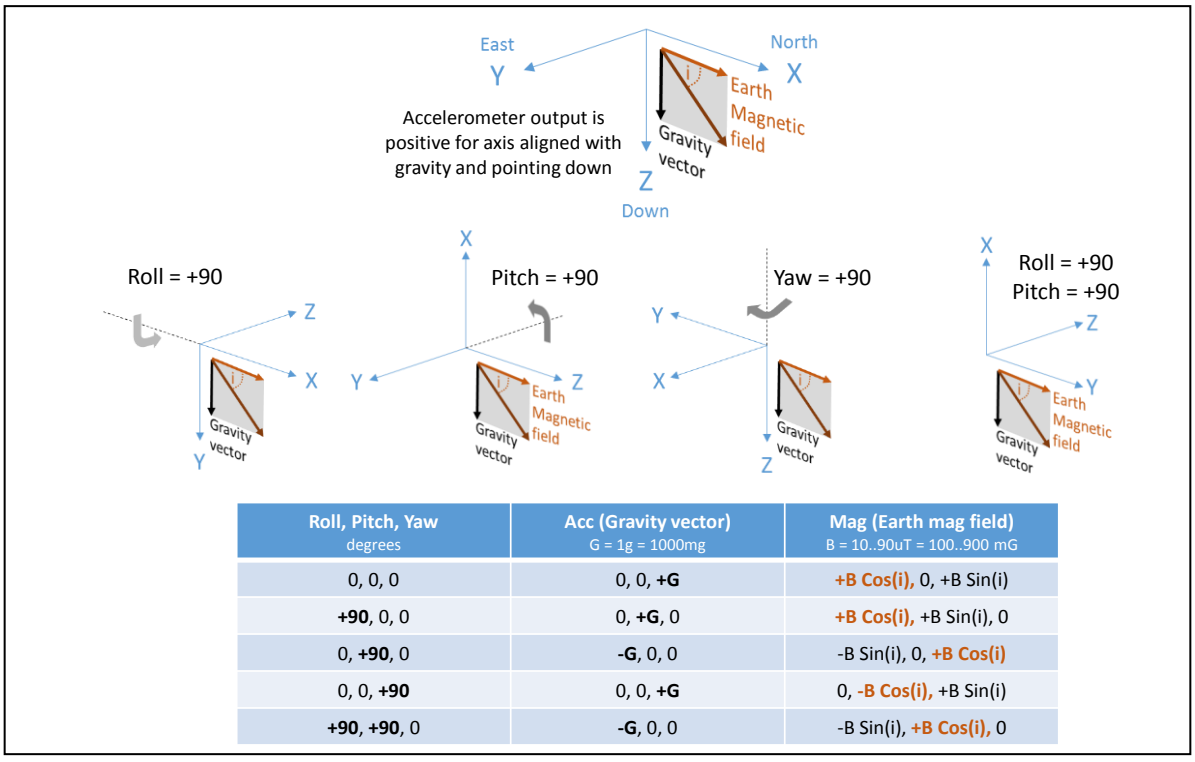
This is the function to de-rotate the input data vector, when Euler angles (Phi for Roll, Theta for Pitch, Psi for Yaw) are known for the reference orientation:

```
function [xr,yr,zr] = derotate_vector(phi,theta,psi,x,y,z)
  % North-East-Down reference frame (see Figure)
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  % derotation matrix coefficients (rotation matrix is transposed)
  m11= cos(theta)*cos(psi);
  m21= cos(theta)*sin(psi);
  m31=-sin(theta);
  m12=sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi);
  m22=sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi);
  m32=sin(phi)*cos(theta);
  m13=cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi);
  m23=cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi);
  m33=cos(phi)*cos(theta);
  % derotate vector (product matrix by vector)
  xr = m11*x + m12*y + m13*z;
  yr = m21*x + m22*y + m23*z;
  zr = m31*x + m32*y + m33*z;
end
```

This is the function to de-rotate the output quaternion, when Euler angles (Phi for Roll, Theta for Pitch, Psi for Yaw) are known for the reference orientation:

```
function [rw,rx,ry,rz] = derotate_quat(phi,theta,psi,pw,px,py,pz)
  % North-East-Down reference frame (see Figure)
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  % derotation quaternion (rotation quaternion is conjugate)
  cosr2=cos(phi/2);   sinr2=sin(phi/2);
  cosp2=cos(theta/2); sinp2=sin(theta/2);
  cosy2=cos(psi/2);   siny2=sin(psi/2);
  qw =  cosr2.*cosp2.*cosy2 + sinr2.*sinp2.*siny2;
  qx = -sinr2.*cosp2.*cosy2 + cosr2.*sinp2.*siny2;
  qy = -cosr2.*sinp2.*cosy2 - sinr2.*cosp2.*siny2;
  qz = -cosr2.*cosp2.*siny2 + sinr2.*sinp2.*cosy2;
  % derotate quaternion (product quaternion by conjugate quaternion)
  rw = pw*qw - px*qx - py*qy - pz*qz;
  rx = pw*qx + px*qw + py*qz - pz*qy;
  ry = pw*qy - px*qz + py*qw + pz*qx;
  rz = pw*qz + px*qy - py*qx + pz*qw;
end
```

Figure 1. Reference orientation for input data from accelerometer and magnetometer, and reference orientation for output data: roll, pitch and 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)
+90, 0, 0	0, **+G**, 0	**+B Cos(i),** +B Sin(i), 0
0, **+90**, 0	**-G**, 0, 0	-B Sin(i), 0, **+B Cos(i)**
0, 0, **+90**	0, 0, **+G**	0, **-B Cos(i),** +B Sin(i)
+90, **+90**, 0	**-G**, 0, 0	-B Sin(i), **+B Cos(i),** 0

Support material

Related design support material
BlueMicrosystem1, Bluetooth low energy and sensors software expansion for STM32Cube
Open.MEMS, MotionFX, Real-time motion-sensor data fusion software expansion for STM32Cube
Documentation

Related design support material
Design Tip, DT0058, Computing tilt measurement and tilt-compensated e-compass

Revision history

Date	Version	Changes
12-Jan-2017	1	Initial release