DT0106
Design tip

Residual linear acceleration by gravity subtraction to enable dead-reckoning

By Andrea Vitali

Purpose and benefits

This design tip explains how to compute the residual linear acceleration, both in terms of sensor body frame or in terms of world reference frame. The latter reference frame is needed to perform dead-reckoning based on double integration of the residual linear acceleration.

Benefits:

- Added functionality with respect to data fusion provided by MotionFX library.
- Short implementation which enables easy customization and enhancement by the end-user. Easy to use on every microcontroller.

Scope

This design tip applies to all accelerometers, eCompass modules, and iNemo inertial IMUs from STMicroelectronics.

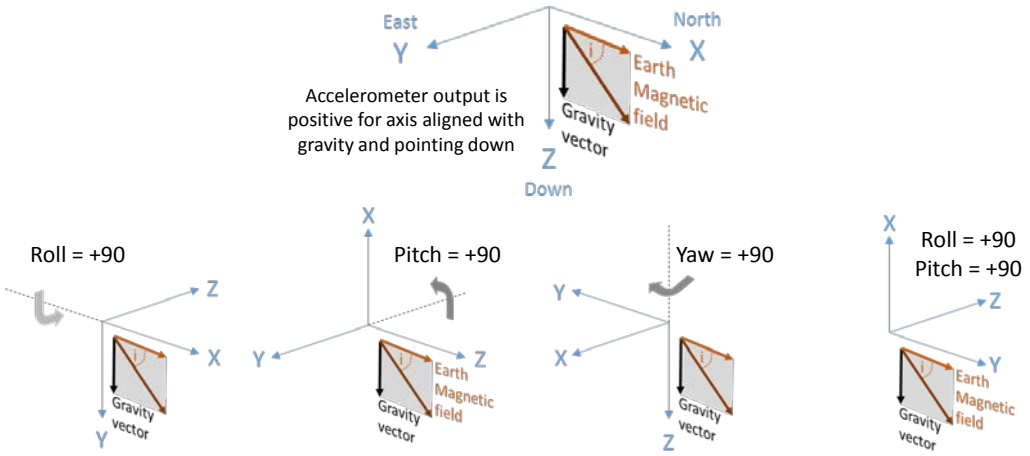
Residual linear acceleration in terms of the sensor body frame

The gravity vector is rotated based on the estimated orientation of the sensor, then it is subtracted from the acceleration vector. Looking at Figure 1, the gravity vector is [0;0;1] (column vector). The code is then the following:

```
linacc_body = -(acc – rotM*[0;0;1]) % gravity vector is rotated and subtracted
```

The negative sign is due to the fact that the output of the accelerometer axis is positive when pointing down towards the gravity vector. As an example, the Z-axis is pointing down and a positive acceleration can be obtained by an upward motion along –Z.

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

The rotation matrix can be computed from the current orientation, expressed as quaternion or Euler angles, using one of the following functions:

```
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

function M = euler2rotM(phi,theta,psi)
  % North-East-Down reference frame
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  % Rx_phi = [1,0,0;0,cos(phi),sin(phi);0,-sin(phi),cos(phi)];
  % Ry_theta = [cos(theta),0,-sin(theta);0,1,0;sin(theta),0,cos(theta)];
  % Rz_psi = [cos(psi),sin(psi),0;-sin(psi),cos(psi),0;0,0,1];
  % M = Rx_phi * Ry_theta * Rz_psi;
  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
```

Using the rotation matrix derived from the Euler angles and exploiting the fact that the gravity vector has two components out of three equal to zero, the following function can also be used:

```
function linacc = acceuler2linacc(acc,phi,theta)
  % North-East-Down reference frame
  % Roll(phi) Pitch(theta) Yaw(psi), angles in radians
  gx = -sin(theta);
  gy = cos(theta).*sin(phi);
  gz = cos(theta).*cos(phi);
  g = [gx,gy,gz]; % rotated gravity vector
  linacc = -(acc-g); % residual linear acceleration in sensor body coordinates
  % '-' because acc>0 when axis pointing down to gravity
end
```

Using the quaternion, the rotation can also be expressed as the product of three elements: the conjugate quaternion, the pure quaternion corresponding to the gravity vector [0,0,0,1], and the quaternion itself. The following function can then be used:

```
function linacc = accquat2linacc(acc,Q)
  % [0,gx,gy,gz] = conj(q) * [0,0,0,1] * q, in NED reference frame
  qw=Q(1); qx=Q(2); qy=Q(3); qz=Q(4); % quaternion
  gx = 2*(qx*qz-qw*qy);
  gy = 2*(qw*qx+qy*qz);
  gz = qw*qw-qx*qx-qy*qy+qz*qz;
  g = [gx,gy,gz]; % rotated gravity vector
  linacc = -(acc-g); % residual linear acceleration in sensor body coordinates
  % '-' because acc>0 when axis pointing down to gravity
end
```

Residual linear acceleration in terms of the world reference frame

The acceleration vector is de-rotated based on the estimated orientation of the sensor, then the gravity vector is subtracted. Looking at Figure 1, the gravity vector is [0;0;1] (column vector). The code is then the following:

```
linacc_world = -(rotM' * acc - [0;0;1]) % the inverse of rotM is the transpose
```

The negative sign is due to the fact that the output of the accelerometer axis is positive when pointing down towards the gravity vector. As an example, the Z-axis is pointing down and a positive acceleration can be obtained by an upward motion along –Z.

The rotation matrix can be computed from the current orientation, expressed as quaternion or Euler angles, using one of the functions presented in the previous paragraph.

The residual linear acceleration, in terms of the world reference frame, can also be computed by de-rotating the residual linear acceleration expressed in terms of the sensor body frame:

```
linacc_world = (rotM' * linacc_body) % the inverse of rotM is the transpose
```

Dead-reckoning by double integration of world-referenced linear acceleration

The device velocity can be obtained by a first integration of the residual linear acceleration in terms of the world reference frame. The device position can then be obtained by a second integration of the velocity.

Care must be taken to eliminate any spurious offset in the residual linear acceleration. The error induced by the spurious offset will grow linearly with time after the first integration, and it will grow proportional to time squared after the second integration.

The spurious offset can be caused by an imperfect calibration of the accelerometer, or by errors in the orientation estimation which in turn causes an imperfect subtraction of the gravity vector.

The accelerometer can be calibrated using **6-tumble calibration** (see Design Tip DT0053) or by using **sphere/ellipsoid fitting** (see Design Tip DT0059). The former technique does require the capability to precisely set the orientation of the device when the data point is taken, while the latter does only require that the device is not subject to linear accelerations when the data point is taken.

When linear acceleration is present, the orientation should be estimated by **exploiting the gyroscope** (see Design Tip DT0060) and NOT by using the accelerometer data (see Design Tip DT0058). If a gyroscope is not present in the system, a virtual gyroscope can be emulated by **exploiting the magnetometer** (see Design Tip **DT0104**).

In the simplest implementation, the integration error can be controlled by using a leaky integrator which is the same as using a high-pass filter to remove the continuous component and the low-frequency portion of the velocity and position.

The reference code is listed below: **acc** is the Nx3 matrix holding the residual linear acceleration in world coordinates, **Ts** is the sampling interval in seconds, **alpha** is the leaky factor (usually set to 0.9-0.95).

```
% residual linear acceleration must be in world reference frame
function [poshp,velhp] = deadreckon_very_simple(acc,Ts,alpha)
  velhp=acc; poshp=acc; % pre alloc for speed
  acc=acc*9.81; % g->m/s
  %c=(1+alpha)/2; % unity gain at high frequencies
  %freqz([c -c],[1 -alpha],1024,Fs) % frequency response
  c=1; % simpler implementation but high frequencies are amplified
  for i=2:length(acc),
    % 1st integration, lin acceleration -> velocity
    velhp(i,:) = c*(acc(i,:)*Ts) + alpha*velhp(i-1,:);
    % 2nd integration, velocity -> position
    poshp(i,:) = c*(velhp(i,:)*Ts) + alpha*poshp(i-1,:);
  end;
end
```

Another technique which can be of help in controlling the integration error is known as **Zero-Velocity-Update** (ZVU): when the modulus of the acceleration is 1*g* and the output of the gyroscope is near 0, then it is assumed that the velocity is zero, and the corresponding integrator is reset to 0. If all the conditions are met in the above code, this instruction is executed: velhp(i,:)=[0,0,0].

It must be noted that a constant velocity would falsely trigger the ZVU reset. Then a constant velocity must be unlikely or impossible in the application of interest.

If real-time processing is not a requirement, it is possible to smooth out the abrupt transition that can happen when a ZVU reset happens: the residual linear acceleration can be integrated backward from the current ZVU point to the previous ZVU point and the output of the backward integrator can be mixed with the output of the forward integrator.

As an example, the weight **W** for the backward integrator output can linearly go from 1 down to 0 when going from the current ZVU point to the previous ZVU point; then the weight for the forward integrator can simply be set to **1-W**.

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
Design Tip, DT0053, 6-point tumble sensor calibration
Design Tip, DT0058, Computing tilt measurement and tilt-compensated e-compass
Design Tip, DT0059, Ellipsoid or sphere fitting for sensor calibration
Design Tip, DT0060, Exploiting the gyroscope to update tilt measure and e-compass
Design Tip, DT0104, Exploiting the magnetometer as a virtual gyroscope at low and ultra-high spin rates

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.