DT0075
Design tip

MotionFX orientation estimation quick guide

Andrea Vitali

Main components	
NUCLEO-F401RE	Nucleo-64 development board with STM32F401RE MCU, supports Arduino and ST morpho connectivity
X-NUCLEO-IKS01A1	Motion MEMS and environmental sensor expansion board for the STM32 Nucleo
STEVAL-MKI160V1	LSM6DS3 adapter board for the standard DIL24 socket
BLUEMICROSYSTEM1	Bluetooth low energy and sensors software expansion for the STM32Cube
osxMotionFX	Real-time motion-sensor data fusion software expansion for the STM32Cube

Purpose and benefits

This design note explains how to configure and use the osxMotionFX software library and includes:

- Hints for a fast startup when proper shutdown sequence is implemented by saving thresholds and calibration parameters for the gyroscope and the magnetometer
- Guide to accurate gyroscope calibration in static and dynamic conditions, including discussion on how a static condition can be automatically identified
- Guide to accurate magnetometer calibration, including discussion on how to track a varying magnetic environment and how to get convergence to true yaw in static conditions and control speed of convergence in dynamic conditions
- Guide to accelerometer use, including discussion on how to control speed of convergence in static and dynamic conditions
- Explanation of format and orientation conventions for inputs and outputs

Configuring MotionFX: control knobs

`Float ATime:` 0.5 to 10; if it is low, accelerometer is trusted more in the update phase (see below) in <u>static condition</u> (no motion); it is convenient to set it to a high value if accelerometer is considered unreliable because there are large offset and gain errors.

`Float MTime:` 0.5 to 10; if it is low, magnetometer is trusted more in the update phase (see below) in <u>dynamic condition</u>; it is convenient to set it to a high value if magnetometer is unreliable because of magnetic anomalies or imperfect hard/soft iron compensation. In the static condition, the magnetometer is not trusted.

`Float FrTime:` 0.5 to 10; if it is low, accelerometer is trusted more in the update phase (see below) in dynamic condition (high-g motion); it is convenient to set it to high value to reduce the effect of high-g motion, or if the accelerometer is considered unreliable.

`Unsigned char LMode:` specify gyroscope calibration.

- `LMode=0,` gyro bias calibration is not active
- `LMode=1,` gyro bias calibration is active in static condition (more accurate); static condition is identified as explained below.
- `LMode=2,` gyro bias calibration is active in dynamic condition (less accurate)

`Float gbias_mag_th_sc_6X,gbias_acc_th_sc_6X,bias_gyro_th_sc_6X:` if high-pass filtered output from mag/acc/gyro is below threshold, then a static condition is identified; thresholds for 6X data fusion (acc+gyro)

`Float gbias_mag_th_sc_9X,gbias_acc_th_sc_9X,gbias_gyro_th_sc_9X:` `if` high-pass filtered output from mag/acc/gyro is below threshold, then a static condition is identified; thresholds for 9X data fusion (acc+gyro+mag)

`Unsigned char modx:` down sampling factor to reduce computational complexity; `osx_MotionFX_propagate()` needs to be always executed in order to estimate orientation based on previous estimate and internal state; `osx_MotionFX_update()` can be executed always `(modx=1`) or less frequently (`modx>1`) to correct the estimate with information from sensors.

`Int start_automatic_gbias_calculation:` set to 1 to start automatic computation of aforementioned thresholds; automatically reset to 0 when computation is completed; sensors needs to be kept static when computation is ongoing.

Using MotionFX: pseudo-code sequence

Initialization (to be done once)

1. Init sensors (acc and gyro for 6X fusion, also mag for 9X fusion); on power-on wait for transients to be completed in order to get good data samples
2. Init MotionFX fusion: `osx_MotionFX_initialize()`
3. Init mag calibration: `osx_MotionFX_compass_Init()`
4. `osx_MotionFX_getKnobs()`; modify settings; `_setKnobs()`
5. Reset by disabling fusion: `osx_MotionFX_enable_6X(0)` / `_9X(0)`

Start fusion

1. Init gyro calibration if possible: `osx_MotionFX_setGbias()`

2. Init mag calib if possible: `osx_MotionFX_compass_setCalibrationData()`
3. Enable data fusion: `osx_MotionFX_enable_6X(1)` / `_9X(1)`

Data fusion (must be always executed at each iteration)

1. Read input data from accelerometer and gyroscope for 6X fusion
2. Read input data also from magnetometer for 9X fusion
3. Compensate mag hard/soft iron effects (e.g. compensate hard iron by subtracting offsets read when mag calibration is completed, see below)
4. Set input: acc in G, gyro in DPS (deg per second), mag in uT/50 (microtesla/50)
5. Set correct delta time: e.g. if 100Hz, then dt = 0.01msec
6. Call `osx_MotionFX_propagate()`
7. Call `osx_MotionFX_update()`; increase speed by setting `modx>1` in control knobs, this will cause the function to be effectively executed only once every modx calls
8. Get output: Euler angles (roll, pitch, yaw), quaternion, gravity, residual linear acceleration, heading to magnetic North

Mag calibration (can be executed less frequently, at some iteration)

1. If not started: `osx_MotionFX_compass_forceReCalibration()`
2. Set input: acc in mG (milli-g), mag in mGauss (milli-Gauss)
3. Call `osx_MotionFX_compass_saveAcc()`
4. Call `osx_MotionFX_compass_saveMag()`
5. Call `osx_MotionFX_compass_run()`, this effectively runs the calibration
6. If `osx_MotionFX_compass_isCalibrated()` becomes true, <u>and periodically</u>:
   a. `osx_MotionFX_getCalibrationData()` and save for compensation above
   b. for immediate convergence to true yaw: reset fusion by disable/enable; note that convergence to true yaw is allowed only in dynamic condition, because in static condition the magnetometer is not trusted

Stop fusion

a. Save gyro calibration for later use: `osx_MotionFX_getGbias()`
b. Save mag calibration for later use: `osx_MotionFX_getCalibrationData()`
c. Disable `osx_MotionFX_enable_6X(0)` / `_9X(0)`

MotionFX magnetometer calibration

Magnetometer calibration is running only if `osx_MotionFX_compass_SaveAcc()` and `_SaveMag()` functions are called in the main loop.

Calibration is started by `osx_MotionFX_forceReCalibration()`. Completion status is checked by `osx_MotionFX_compass_isCalibrated()`.

After completion, calibration parameters can be read and used to compensate hard/soft iron effects in magnetometer data before they are fed to orientation estimation functions `osx_MotionFX_propagate()` and `_update()`. In the current version of the library, only offsets are estimated, hence only hard iron effects can be compensated.

Even after completion, if `_SaveAcc()` and `_SaveMag()` are called, calibration continues to run in the background and may update the calibration parameters; this is why calibration parameters should be read periodically even after completion, so that any update can be captured and exploited.

In the current version of the library, magnetic anomalies are not rejected, but they are filtered out. If there is a limited amount of anomalies with respect to good data, accuracy of calibration is minimally affected. In some case, it can also happen that the anomalies' effects do cancel each other out.

MotionFX input/output conventions

Input is coming from 3-axis sensors: accelerometer, gyroscope and magnetometer. The orientation of sensors is specified by a three-letter code, one for each axis.

Letters (NSWEUD) indicate where axes (XYZ) are pointing. First letter for X, second letter for Y, third letter for Z. In figure 1, there are four examples: **NED, ENU, SEU and NWU** system/sensor orientation (left). The specific configuration for sensors on **IKS01A1** Nucleo eXpansion board is also shown (right).

Output is the orientation in different formats: Quaternion and Euler angles. Output is referenced to a coordinate system specified by a three-letter code.

Quaternions are arrays of 4 values, **Q = [X, Y, Z, W]**. When processing quaternions, it is important to remember that they are a redundant representation of orientation. Q is equivalent to –Q. As an example: averaging Q and –Q should not give zero as output, but Q or –Q.

Euler angles in the **NED system**, where XYZ axes are North-East-Down, are as follows:

- Roll, also known as bank, is a clockwise rotation around X looking to North
- Pitch, also known as attitude, is a clockwise rotation around Y looking to East
- Yaw, also known as heading, is a rotation around Z looking Down

Figure 1. Sensor orientation: NED, ENU, SEU, NWU. Sensor orientation in IKS01A1 Nucleo eXpansion board.

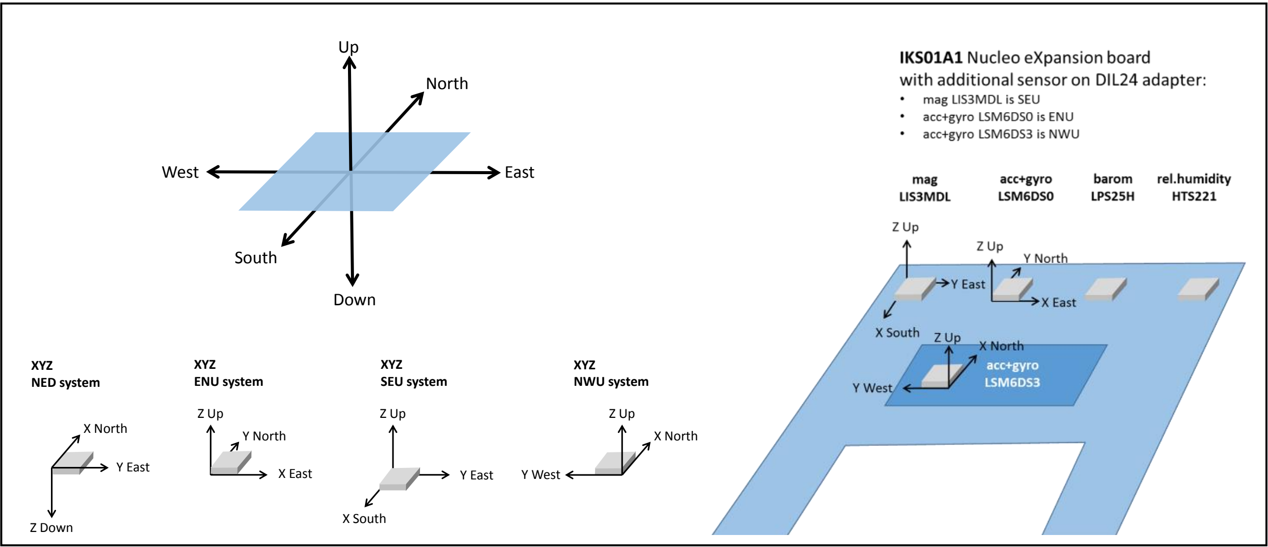
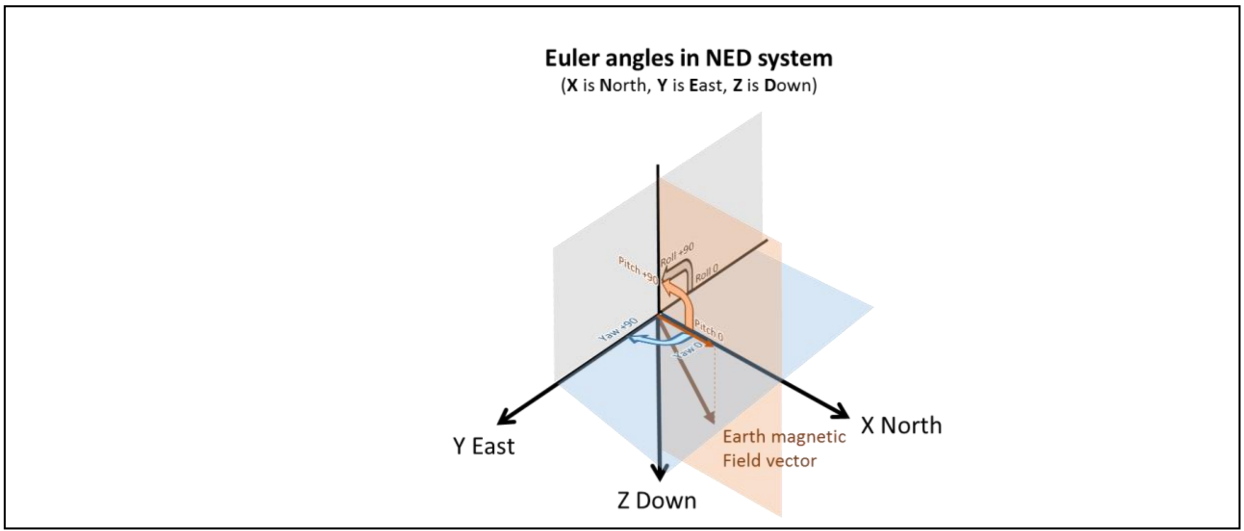


Figure 2. Euler angles in North-East-Down coordinate system.



Support material

Related design support material
Product Evaluation board – X-NUCLEO-IKS01A1, Motion MEMS and environmental sensor expansion board for STM32 Nucleo
Product Evaluation board –STEVAL-MKI160V1, LSM6DS3 adapter board for standard DIL24 socket
Product Evaluation board – NUCLEO-F401RE, Nucleo-64 development board with STM32F401RE MCU, supports Arduino and ST morpho connectivity
Product Evaluation board – BLUEMICROSYSTEM1, Bluetooth low energy and sensors software expansion for STM32Cube
Development kit – osxMotionFX Real-time motion-sensor data fusion software expansion for

Related design support material
STM32Cube X-CUBE-MEMS1
Documentation
Databrief, DB2531, Real-time motion-sensor data fusion software expansion for STM32Cube
User manual, UM1866, Getting started with the osxMotionFX fusion and compass library for X-CUBE-MEMS1 expansion for STM32Cube
Application note, AN4615, Fusion and compass calibration APIs for STM32 Nucleo with the X-NUCLEO-ISK01A1 sensors expansion board
Design Tip, DT0050, How to install and run osxMotionFX Sensor Data Fusion library

Revision history

Date	Version	Changes
11-Jan-2017	1	Initial release