UM2146
User manual

Getting started with osxMotionMC magnetometer calibration library for X-CUBE-MEMS1 expansion for STM32Cube

Introduction

The osxMotionMC add-on software package for X-CUBE-MEMS1 software runs on the STM32 and includes drivers that recognize the inertial sensors. It provides real-time magnetometer calibration using hard iron (HI) and scale factor (SF) coefficients to correct magnetometer data.

The osxMotionMC magnetometer calibration algorithm is provided in static library format and is designed to be used in STM32 microcontrollers based on the ARM Cortex-M3 or ARM Cortex-M4 architecture.

It is built on top of the STM32Cube software technology for portability across different STM32 microcontrollers.

The software comes with sample implementations running on the X-NUCLEO-IKS01A1 (with optional STEVAL-MKI160V1) or X-NUCLEO-IKS01A2 expansion board on a NUCLEO-F401RE or NUCLEO-L476RG development board.

Contents

1 osxMotionMC library add-on to X-CUBE-MEMS1 software expansion for STM32Cube 4
1.1 osxMotionMC overview 4
1.2 osxMotionMC architecture 4
1.3 osxMotionMC folder structure 5
1.4 osxMotionMC library 6
1.4.1 osxMotionMC library description 6
1.4.2 osxMotionMC APIs 6
1.4.3 Storing and loading calibration parameters 7
1.4.4 API flow chart 8
1.4.5 Magnetometer calibration demo code 8
1.4.6 Calibration process 9
2 Sample application 11
2.1 Unicleo-GUI utility 11
3 References 15
4 Revision history 16

List of figures

Figure 1: osxMotionMC plus X-CUBE-MEMS1 software architecture....5
Figure 2: osxMotionMC package folder structure....5
Figure 3: osxMotionMC API logic sequence....8
Figure 4: Rotation of STM32 Nucleo board during calibration....10
Figure 5: STM32 Nucleo development board plus X-NUCLEO-IKS01A1 and STEVAL-MKI160V1 boards....11
Figure 6: Unicleo main window....12
Figure 7: User Messages tab....12
Figure 8: Activity recognition for Wrist window....13
Figure 9: Datalog window....14

1 osxMotionMC library add-on to X-CUBE-MEMS1 software expansion for STM32Cube

1.1 osxMotionMC overview

This software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. It extends STM32Cube with a board support package (BSP) for the sensor expansion board and middleware components for serial communication with a PC.

The drivers abstract low-level details of the hardware and allow the middleware components and applications to access sensor data in a hardware independent fashion.

The osxMotionMC real-time software acquires data from the magnetometer and counts the hard iron (HI) and scale factor (SF) (diagonal elements of soft iron matrix) coefficients together with the calibration quality value. The HI and SF coefficients are then used to compensate raw data from the magnetometer.

The osxMotionMC package includes a sample application that developers can use to start experimenting with code that enables sensor data logging on a PC.

The key package features include:

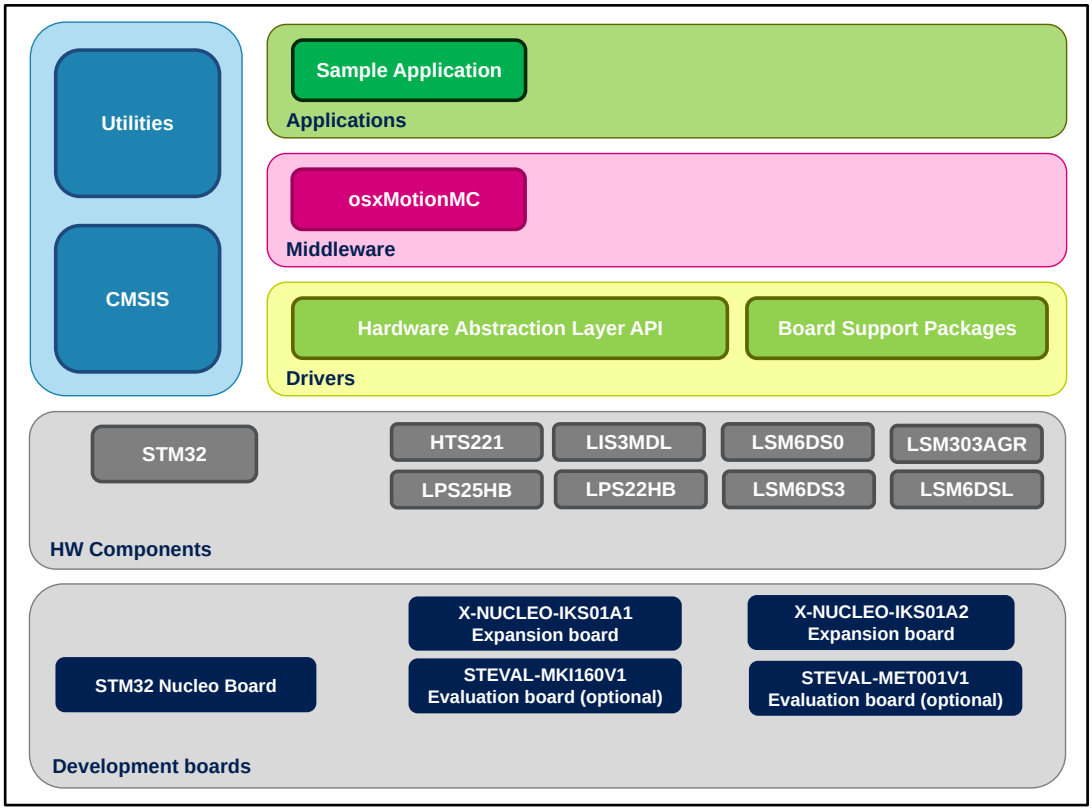
- Real-time magnetometer calibration algorithm (under Open.MEMS license) based exclusively on magnetometer data.
- Complete middleware to build applications on top of X-CUBE-MEMS1.
- Sample application to transmit real time sensor data to a PC.
- Easy portability across different MCU families, thanks to STM32Cube.
- PC-based Windows application to log sensor data.
- Free user-friendly license terms
- Sample implementations for X-NUCLEO-IKS01A2 and X-NUCLEO-IKS01A1 (with optional STEVAL-MKI160V1) expansion boards mounted on a NUCLEO-F401RE or NUCLEO-L476RG development board.

1.2 osxMotionMC architecture

The following software layers are used by the application to access and use the sensor expansion board:

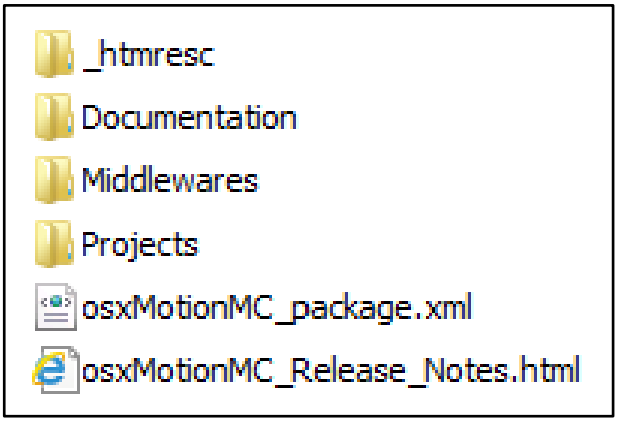
- **STM32Cube HAL layer**: consists of simple, generic and multi-instance APIs (application programming interfaces) which interact with the upper layer applications, libraries and stacks. These generic and extension APIs are based on a common framework so that overlying layers like middleware can function without requiring specific microcontroller unit (MCU) hardware information. This structure improves library code reusability and guarantees easy portability across other devices.
- **Board support package (BSP) layer**: provides software support for the STM32 Nucleo board peripherals, excluding the MCU. These specific APIs provide a programming interface for certain board specific peripherals like LEDs, user buttons, etc and can also be used to fetch individual board version information. It also provides support for initializing, configuring and reading data.

Figure 1: osxMotionMC plus X-CUBE-MEMS1 software architecture



1.3 osxMotionMC folder structure

Figure 2: osxMotionMC package folder structure



The following folders are included in the package:

- **Documentation**: contains a compiled HTML file detailing the software components and APIs.
- **Middlewares**: contains the osxMotionMC static library binary code, the library header file, documentation, license information plus header file for node-locked license validation.
- **Projects**: contains a sample application for the NUCLEO-F401RE or NUCLEO-L476RG development platforms to access sensor data and calibration coefficients in

the IAR Embedded Workbench for ARM, µVision (MDK-ARM) toolchain and System Workbench for STM32 integrated development environments.

1.4 osxMotionMC library

Detailed technical information fully describing the functions and parameters of the osxMotionMC APIs can be found in the osxMotionMC_Package.chm compiled HTML file the package Documentation folder.

The osxMotionMC is provided as a node-locked library which allows derivative firmware images to run on a specific STM32 Nucleo device only. License activation codes must be requested from ST and included in the project (and become part of the build process) prior to usage. The resulting firmware binary image will therefore be node-locked.

For complete information about the open.MEMS license agreement, please refer to the license file located in the Middlewares/ST/STM32_OSX_MotionMC_Library folder.

1.4.1 osxMotionMC library description

The osxMotionMC magnetometer calibration library manages data acquired from a magnetometer; it features:

- update frequency up to 100 Hz
- hard iron compensation is theoretically unlimited (within sensor range)
- wide scale factor compensation range from 0.65 to 1.35 in every direction
- supports all ST magnetometer devices
- occupies 20 kB of code and 3 kB of data memory
- the library is available for ARM Cortex-M3 and Cortex-M4 architectures

1.4.2 osxMotionMC APIs

The exposed APIs of the osxMotionMC library are listed below:

- `uint8_t osx_MotionMC_GetLibVersion(char *version)`
  - retrieves the version of the library
  - `*version` is a pointer to an array of 35 characters
  - returns number of characters in the version string
- `int osx_MotionMC_Initialize(int sampleTime_ms, unsigned short int onOff)`
  - performs osxMotionMC library initialization and setup of the internal mechanism used for node-locking (see *Section 4: "References"*) and enables or disables library. This function must be called before using the magnetometer calibration library.
  - parameter `sampleTime_ms` is the 10 ms (100 Hz) to 60 ms (16.6 Hz) interval between update function calls, which should be greater than or equal to the magnetometer data rate.When the sample rate changes, this function should be called again for disable and enable. When sample rate is inside the recommended range it returns '1', otherwise it returns '0' and the calibration library will not run until a correct sample rate is set.
  - parameter `onOff` is enable ('1') or disable ('0') library.
  - returns 1 for correct initialization or 0 otherwise (e.g., 0 is returned for license errors)
- `void osx_MotionMC_Update(int x_mG, int y_mG, int z_mG, int timeStamp_ms)`
  - executes magnetometer calibration algorithm

  - this function must be called at the same frequency indicated in the initialization function
  - parameters `x_mG, y_mG, z_mG` represent X, Y and Z axis magnetic field strength in 10^{-3} Gauss (mG)
  - parameter `timeStamp_ms` is the timestamp for magnetometer output values
- `osx_MMC_CalQuality_t osx_MotionMC_GetCalParams(int *HI_bias, float SI_matrix[][3])`
  - retrieves the magnetometer calibration coefficients for hard/soft iron compensation and calibration quality factor
  - `*HI_bias` is the pointer to an array of 3 elements containing the hard iron offset for each axis, the unit is 10^{-3} G
  - `SI_matrix` is the soft iron correction matrix (symmetric 3x3 matrix).Only scale factor (diagonal elements of the soft iron matrix) applies, the skew (non-diagonal elements) are zero
  - returns calibration quality factor:
    - OSX_MMC_CALQSTATUSUNKNOWN = 0; accuracy of calibration parameters are unknown
    - OSX_MMC_CALQSTATUSPOOR = 1; accuracy of calibration parameters are poor, cannot be trusted
    - OSX_MMC_CALQSTATUSOK = 2; accuracy of calibration parameters are OK
    - OSX_MMC_CALQSTATUSGOOD = 3; accuracy of calibration parameters are good.

1.4.3 Storing and loading calibration parameters

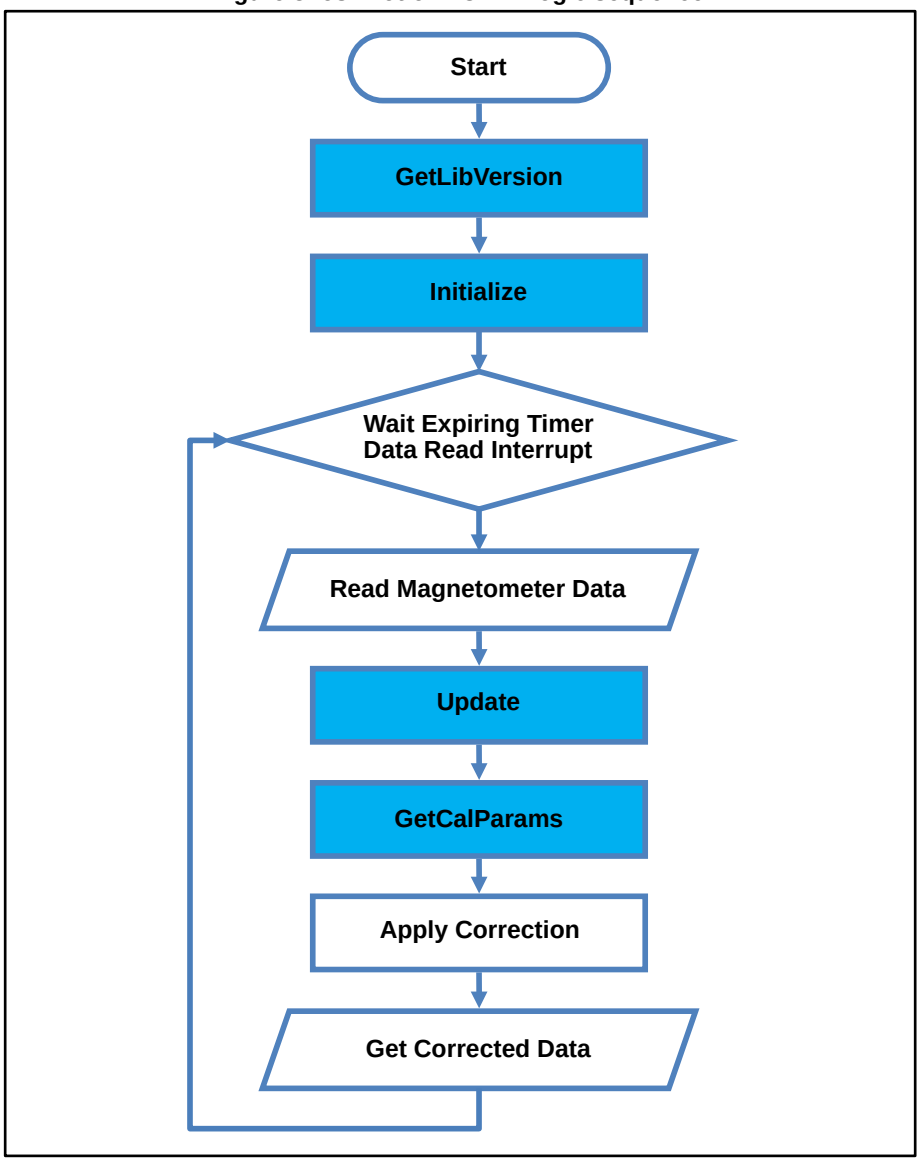
The following functions have to be implemented specifically for each target platform:

- `char MCalLoadFromNVM(unsigned short int datasize, unsigned int *data)`
  - the function is used to retrieve the magnetometer calibration parameters from storage
  - `dataSize` is the data size
  - `*data` is the data location pointer
  - returns 0 for correct loading, 1 otherwise
  - if calibration data are not available or deleted, default calibration parameters are used
- `char MCalSaveInNVM(unsigned short int datasize, unsigned int *data)`
  - the function is used to save the magnetometer calibration parameters in storage
  - `dataSize` is the data size
  - `*data` is the data location pointer
  - returns 0 for correct loading, 1 otherwise

These functions need to be implemented but should not be called; the magnetometer calibration library decides when to call these functions. They may be implemented as empty (always return 1) if saving and loading calibration coefficients is not needed.

1.4.4 API flow chart

Figure 3: osxMotionMC API logic sequence



1.4.5 Magnetometer calibration demo code

The following demonstration code reads data from magnetometer sensor in 10^{-3} Gauss (mG) (`raw_x, raw_y, raw_z`) and calculates compensated data in 10^{-3} Gauss (`cal_x, cal_y, cal_z`).

```
[…]
#define VERSION_STR_LENG 35
#define SAMPLETIME       40   /* 40ms -> 25 Hz */

[…]

/*** Initialization ***/
volatile uint32_t timeStamp = 0;
volatile uint8 t isLibRunning = 0;
volatile int sampleTime = SAMPLETIME;
char strVersion[VERSION_STR_LENG];
```

```
/* Magnetometer sensor setup (Output Data Rate, …) */
int magSensorRate; /* current Output Data Rate */

[…]

/* Optional: Get version */
osx MotionMC GetLibVersion(strVersion);

/* Mandatory: Set update period and enable library */
isLibRunning = osx MotionMC Initialize(sampleTime, 1);

/*** Using Magnetometer Calibration algorithm ***/
Timer OR DataRate Interrupt Handler()
{
  int raw_x, raw_y,  raw_z;
  int cal x, cal y,  cal z;

  int HI bias[3];        /* Hard iron - offset */
  float SI_matrix[3][3]; /* Soft iron - correction matrix */
  osx_MMC_CalQuality_t goodness;
  /* Magnetometer sensor current Output Data Rate has meanwhile changed */
  if (sampleTime != magSensorRate)
  {
    sampleTime = magSensorRate

    /* Disable library*/
    isLibRunning = osx MotionMC Initialize(sampleTime, 0);

    /* Re-enable library*/
    isLibRunning = osx_MotionMC_Initialize(sampleTime, 1);
  }
  /* Get X/Y/Z in mG */
  MEMS Read MagValue(&raw x, &raw y, &raw z);

  /* Magnetometer calibration algorithm update*/
  osx_MotionMC_Update(raw_x, raw_y, raw_z, timeStamp * sampleTime);

  /* Get correction */
  goodness = osx MotionMC GetCalParams(HI bias, SI matrix);

  /* Apply correction */
  cal_x = (int)((raw_x - HI_bias[0])* SI_matrix[0][0]
              + (raw_y - HI_bias[1])* SI_matrix[0][1]
              + (raw z - HI bias[2])* SI matrix[0][2]);

  cal_y = (int)((raw_x - HI_bias[0])* SI_matrix[1][0]
              + (raw_y - HI_bias[1])* SI_matrix[1][1]
              + (raw_z - HI_bias[2])* SI_matrix[1][2]);

  cal z = (int)((raw x - HI bias[0])* SI matrix[2][0]
              + (raw y - HI bias[1])* SI matrix[2][1]
              + (raw_z - HI_bias[2])* SI_matrix[2][2]);

  timeStamp++;}
```

1.4.6 Calibration process

This calibration algorithm uses the specific motion of the magnetometer sensor exposed to Earth's gravitation field.

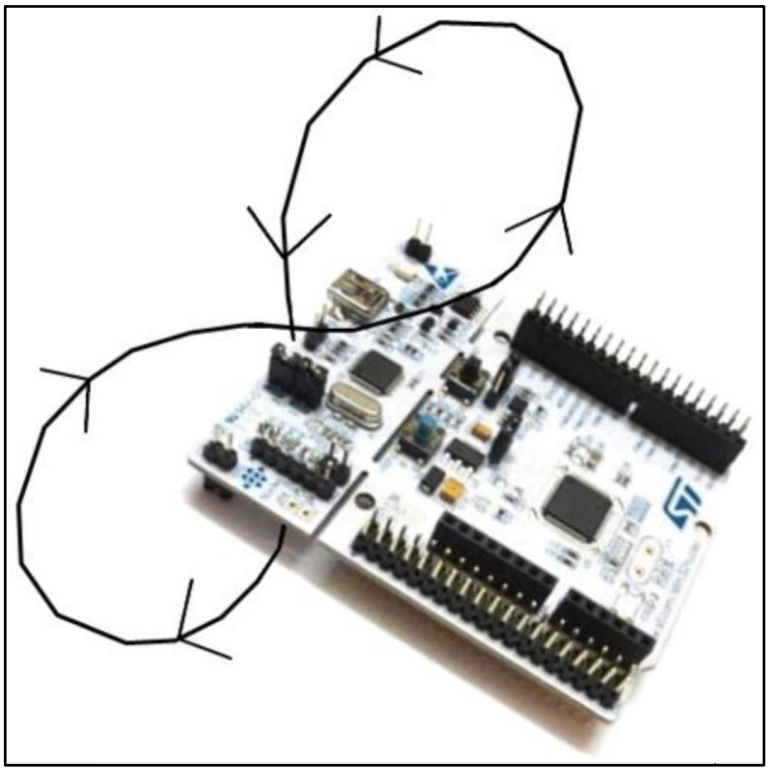
1 Rotate the STM32 Nucleo board slowly in a figure eight pattern through 3D space.

This is not a planar movement; you need to tilt and rotate the device to cover as many different 3D spatial positions as possible.

Use the 3D Plot (see *Section 2.1: "Unicleo-GUI utility"*) during calibration; the measured data points should cover as much of the 3D sphere as possible.

While performing calibration pattern, keep the Nucleo board clear of other magnetic objects such as cell phones, computers and other steel objects.

Figure 4: Rotation of STM32 Nucleo board during calibration



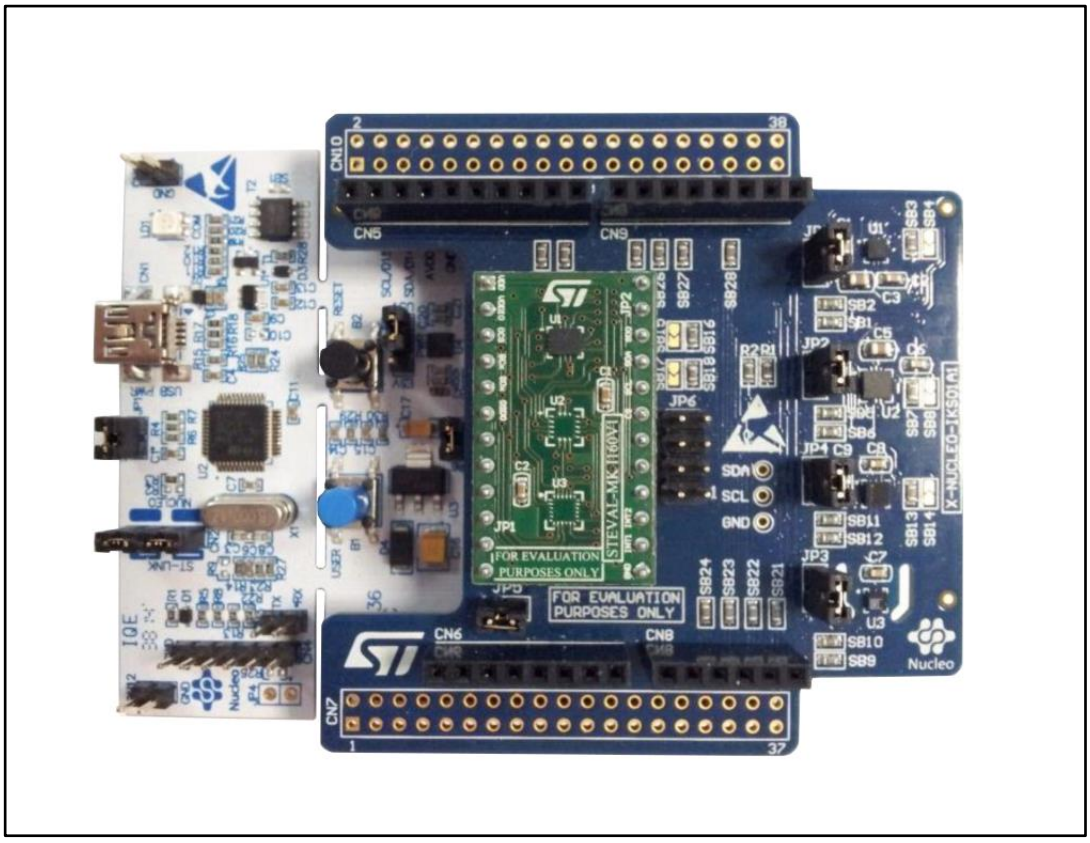
2 Sample application

The osxMotionMC middleware can be easily manipulated to build user applications.

A sample application example in the Projects folder is designed to run on either:

- a NUCLEO-F401RE or NUCLEO-L476RG development board connected to an X-NUCLEO-IKS01A1 expansion board with optional STEVAL-MKI160V1 board
- a NUCLEO-F401RE or NUCLEO-L476RG development board connected to an X-NUCLEO-IKS01A2 expansion board.

Figure 5: STM32 Nucleo development board plus X-NUCLEO-IKS01A1 and STEVAL-MKI160V1 boards



Magnetometer algorithm output data may be displayed in real time through a specific GUI.

2.1 Unicleo-GUI utility

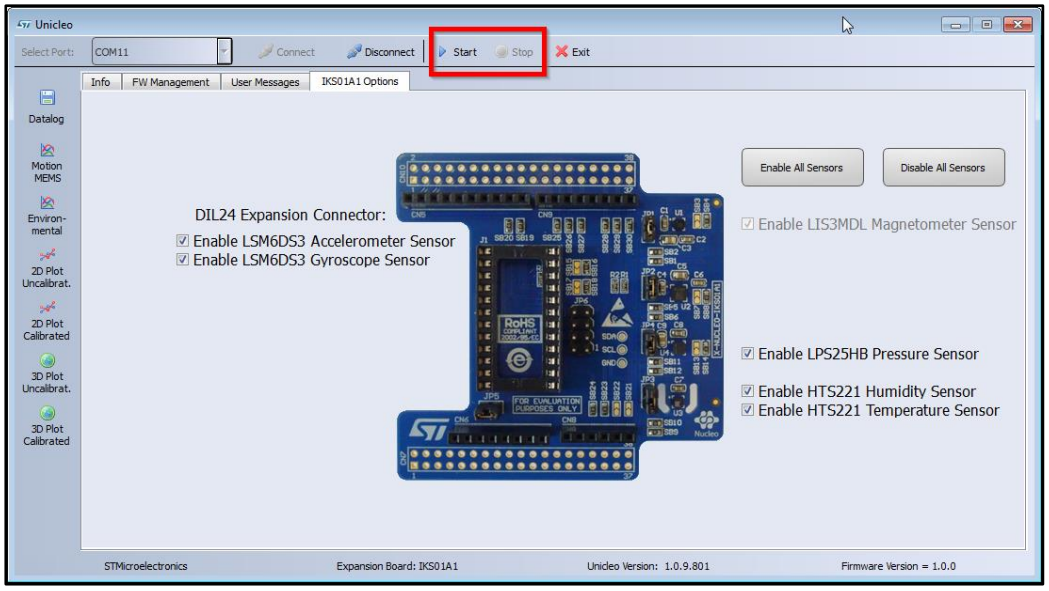
The osxMotionMC software package for STM32Cube uses the Windows Unicleo-GUI utility, which can be downloaded from *www.st.com* (see *5*).

1 Ensure that the necessary drivers are installed and the STM32 Nucleo board with appropriate expansion board is connected to the PC.

2 Launch the Unicleo-GUI application to open the main application window.

If an STM32 Nucleo board with supported firmware is connected to the PC, it will automatically be detected and the appropriate COM port will be opened.

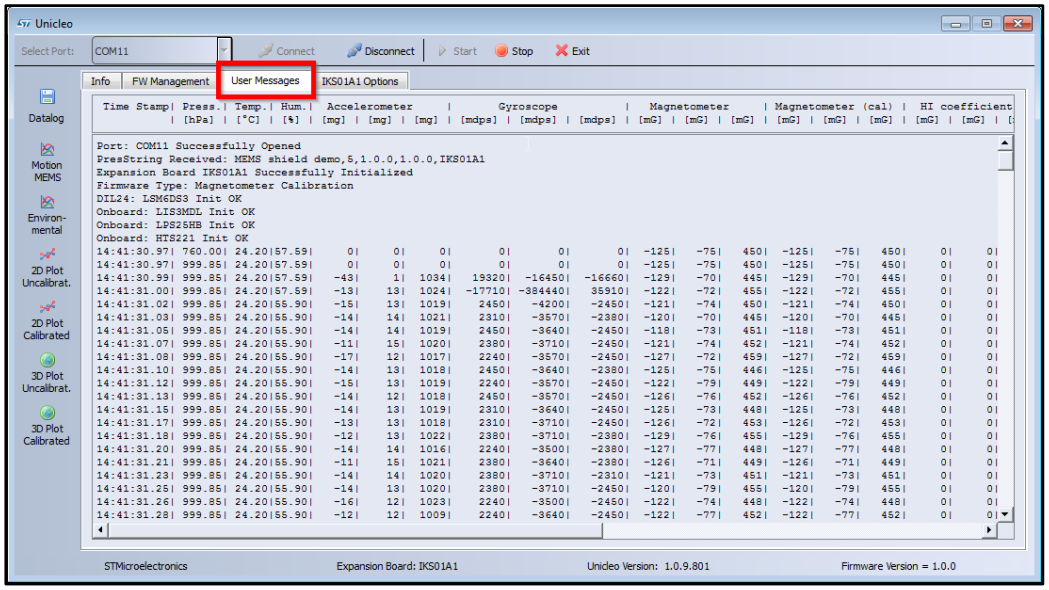
Figure 6: Unicleo main window



3 Start and stop data streaming by using the appropriate buttons on the vertical tool bar.

The data coming from the connected sensor can be viewed in the User Messages tab.

Figure 7: User Messages tab



4 Click on 2D Plot Uncalibrated and 2D Plot Calibrated icons in the vertical tool bar to open the corresponding windows.

The uncalibrated data window shows the hard-iron distortion. The calibrated data window shows all the data centered on the origin (0,0,0) as the hard-iron distortion has been removed through calibration.

- red = X-Y plane
- green = X-Z plane
- blue = Y-Z plane

Figure 8: Activity recognition for Wrist window

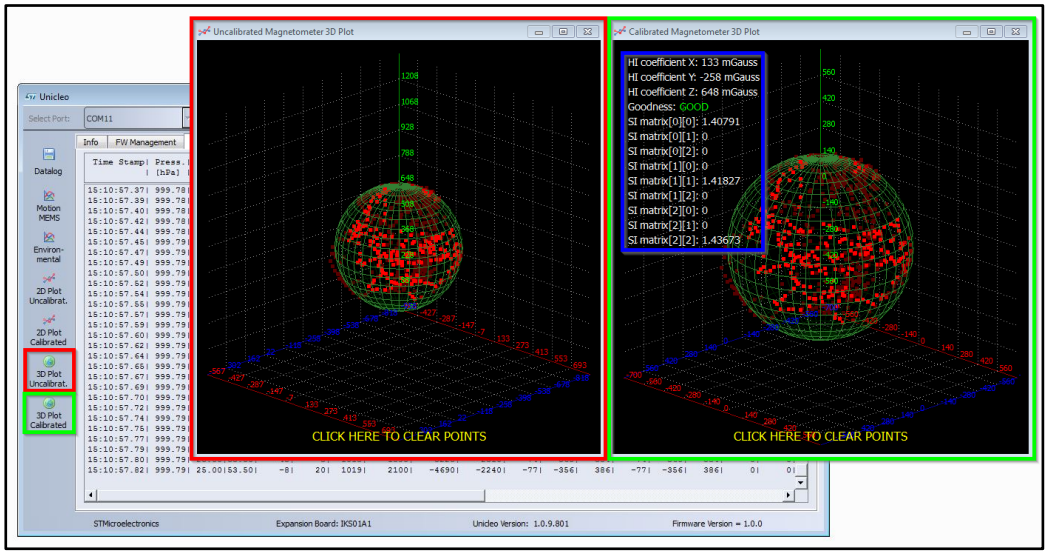
5 Click on 3D Plot Uncalibrated and 3D Plot Calibrated icons in the vertical tool bar to open the corresponding windows.

The uncalibrated data window shows the hard-iron distortion, with the 3D sphere subject to offsets in the x,y and z directions. The calibrated data window shows the 3D sphere centered on the origin (0,0,0) as the hard-iron distortion has been removed through calibration.

The Calibrated Magnetometer 3D Plot also shows the hard-iron and soft-iron coefficients and calibration quality (Goodness).

in soft-iron matrix, only scale-factor (diagonal elements) applies, skew (non-diagonal) elements are always zero

Figure 9: Datalog window



3 References

1. UM1859: Getting started with the X-CUBE-MEMS1 motion MEMS and environmental sensor software expansion for STM32Cube
2. DB3132: Real-time megnetometer calibration software expansion for STM32Cube
3. UM2012: osxMotionXX system setup
4. UM1724: STM32 Nucleo-64 boards
5. UM2128: Unicleo-GUI

4 Revision history

Table 1: Document revision history

Date	Version	Changes
18-Nov-2016	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.