Getting started with MotionAC accelerometer calibration library in X-CUBE-MEMS1 expansion for STM32Cube

Introduction

The MotionAC is a middleware library part of X-CUBE-MEMS1 software and runs on STM32. It provides real-time accelerometer calibration through offset and scale factor coefficients used to correct accelerometer data.

This library is intended to work with ST MEMS only.

The algorithm is provided in static library format and is designed to be used on STM32 microcontrollers based on the ARM® Cortex®-M3, ARM® Cortex®-M4 or ARM® Cortex®-M7 architecture.

It is built on top of STM32Cube software technology that ease portability across different STM32 microcontrollers.

The software comes with sample implementation running on an X-NUCLEO-IKS01A2 or X-NUCLEO-IKS01A3 expansion board on a NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L152RE development board.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
API	Application programming interface
BSP	Board support package
GUI	Graphical user interface
HAL	Hardware abstraction layer
IDE	Integrated development environment

2 MotionAC middleware library in X-CUBE-MEMS1 software expansion

2.1 MotionAC overview

The MotionAC library expands the functionality of the X-CUBE-MEMS1 software.

The library acquires data from the accelerometer and calculates the offset and scale factor coefficients together with the calibration quality value. Calibration can be done in two modes: dynamic calibration and 6-point calibration.The offset and scale factor coefficients are then used to compensate raw data coming from accelerometer.

The library is designed for ST MEMS only. Functionality and performance when using other MEMS sensors are not analyzed and can be significantly different from what described in the document.

Sample implementation is available on X-NUCLEO-IKS01A2 and X-NUCLEO-IKS01A3 expansion boards, mounted on a NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L152RE development board.

2.2 MotionAC library

Technical information fully describing the functions and parameters of the MotionAC APIs can be found in the MotionAC_Package.chm compiled HTML file located in the Documentation folder.

2.2.1 MotionAC library description

The MotionAC accelerometer calibration library manages data acquired from accelerometer; it features:

- offset compensation up to 0.2 g
- scale factor compensation, in range from 0.8 to 1.2 in every direction
- update frequency from 20 to 100 Hz
- resources requirements:
  - Cortex-M3: 12.6 kB of code and 1.1 kB of data memory
  - Cortex-M4: 10.1 kB of code and 1.1 kB of data memory
  - Cortex-M7: 9.9 kB of code and 1.1 kB of data memory
- available for ARM Cortex-M3, Cortex-M4 and Cortex-M7 architectures

2.2.2 MotionAC APIs

The MotionAC APIs are:

- `uint8_t MotionAC_GetLibVersion(char *version)`
  - retrieves the version of the library
  - `*version` is a pointer to an array of 35 characters
  - returns the number of characters in the version string

- `void MotionAC_Initialize(uint8_t enable)`
  - performs MotionAC library initialization and setup of the internal mechanism
  - the CRC module in STM32 microcontroller (in RCC peripheral clock enable register) has to be enabled before using the library

Note: This function must be called before using the accelerometer calibration library.

  - `enable` parameter enables (1) or disables (0) the library

- `void MotionAC_Update (MAC_input_t *data_in, uint8_t *is_calibrated)`
  - executes accelerometer calibration algorithm
  - `*data_in` parameter is a pointer to a structure with input data
  - the parameters for the structure type `MAC_input_t` are:
    - `Acc[3]` is an array of accelerometer sensor value in g
    - `TimeStamp` parameter is the timestamp for accelerometer output values in ms
    - `*is_calibrated` parameter is a pointer to a value which returns 1 if calibration is done with current sample, 0 otherwise

Note: This function has to be called periodically at the same period that is indicated in the initialization function.

- `uint8_t MotionAC_SetKnobs(MAC_knobs_t*knobs)`
  - sets current knob settings
  - returns 1 in case of success and the library is running, 0 otherwise
  - `*knobs` parameter is a pointer to a structure with settings
  - the parameters for the structure type `MAC_knobs_t` are:
    - `Run6PointCal` should be1 in case of 6-point calibration, 0 in case of dynamic calibration (slow motion of the device)
    - `Sample_ms` parameter is the accelerometer sensor sample period in ms
    - `MoveThresh_g` parameter is recommended to be kept between 0.15 - 0.30 g; a higher value loosens the condition on data selection for the calibration but reduces the accuracy (around moveThresh_g / 10)
- `void MotionAC_GetCalParams (MAC_output_t *data_out)`
  - retrieves the accelerometer calibration coefficients for offset and scale factor compensation and calibration quality factor
  - `*data_out` parameter is a pointer to a structure with input data
  - the parameters for the structure type `MAC_output_t` are:
    - `AccBias[3]` is an array of the offset for each axis in g
    - `SF_Matrix[3][3]` is the scale factor correction 3x3 matrix (diagonal matrix)
    - `CalQuality` is the calibration quality factor:
      - MAC_CALQSTATUSUNKNOWN = 0; accuracy of calibration parameters is unknown
      - MAC_CALQSTATUSPOOR = 1; accuracy of calibration parameters is poor, cannot be trusted
      - MAC_CALQSTATUSOK = 2; accuracy of calibration parameters is OK
      - MAC_CALQSTATUSGOOD = 3; accuracy of calibration parameters is good.

2.2.3 Storing and loading calibration parameters

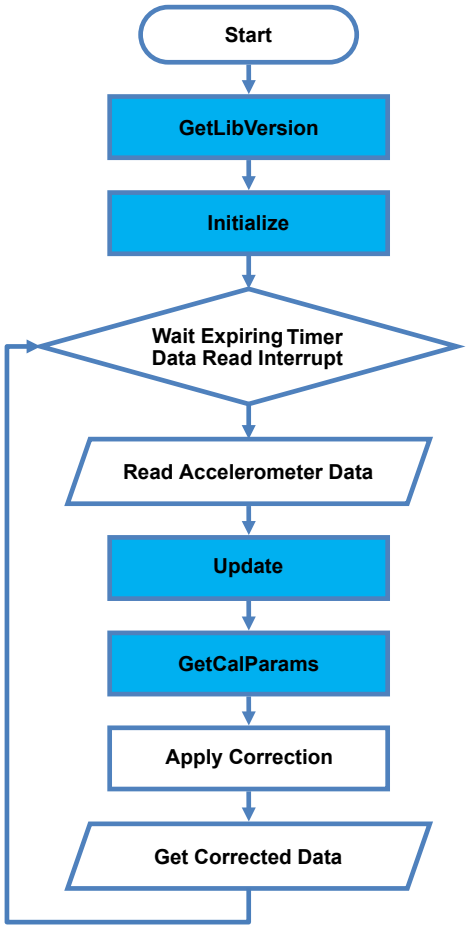
The following functions have to be implemented specifically for each target platform:

- `char MotionAC_LoadCalFromNVM (unsigned short intdataSize, unsigned int *data)`
  - the function is used to retrieve the accelerometer calibration parameters from storage
  - `dataSize` is the data size
  - `*data` is the data location pointer
  - returns 0 for correct loading, 1 otherwise
- `char MotionAC_SaveCalInNVM (unsigned short intdataSize, unsigned int *data)`
  - the function is used to save the accelerometer 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 accelerometer calibration library decides when to call these functions. They may be implemented as empty (always return 0) if saving and loading calibration coefficients is not needed.

2.2.4 API flow chart

Figure 1. **MotionAC API logic sequence**



2.2.5 Demo code

The following demonstration code reads data from accelerometer sensor and calculates compensated data.

```
[…]

#define VERSION_STR_LENG        35
#define REPORT_INTERVAL         20

[…]

/*** Initialization ***/
char lib_version[VERSION_STR_LENG];
MAC_knobs_t Knobs;

/* Accelerometer calibration API initialization function */
MotionAC_Initialize(1);
MotionAC_GetKnobs(&Knobs);
Knobs.Sample_ms = REPORT_INTERVAL;
(void)MotionAC_SetKnobs(&Knobs);

/* Optional: Get version */
MotionAC_GetLibVersion(lib_version);

[…]

/*** Using acceleromtere calibration algorithm ***/
Timer_OR_DataRate_Interrupt_Handler()
{
  MAC_input_t data_in;
  MAC_output_t data_out;;
  float acc_cal_x, acc_cal_y, acc_cal_z;

  /* Get acceleration X/Y/Z in g */
  MEMS_Read_AccValue(data_in.Acc[0], data_in.Acc[1], data_in.Acc[2]);

  /* Accelerometer calibration algorithm update */
  MotionAC_Update(&data_in);

  /* Get Calibration coeficients */
   MotionAC_GetCalParams(&data_out);

  /* Apply correction */
  acc_cal_x = (data_in.Acc[0] - data_out.AccBias[0])* data_out.SF_Matrix[0][0];
  acc_cal_x = (data_in.Acc[1] - data_out.AccBias[1])* data_out.SF_Matrix[1][1];
  acc_cal_x = (data_in.Acc[2] - data_out.AccBias[2])* data_out.SF_Matrix[2][2];
}
```

2.2.6 Calibration process

This calibration algorithm uses the normal motion of the three orthogonal axes of a stationary accelerometer sensor exposed to Earth's gravitation field.

Step 1. Hold the device firmly as shown in position 1.

Step 2. Gently rotate the device by 180° around the YZ plane such that in position 4, the device is flipped to its back side.

Step 3. Rotate the device by 180° in a clockwise fashion around the XZ plane to reach position 1.

Important: *Try to rotate the device along a smooth path and at a constant speed.*

You can also perform standard six point calibration, holding the module stationary in six different directions (positive and negative X,Y and Z directions).

Figure 2. Calibration movement

2.2.7 Algorithm performance

Table 2. Cortex-M4 and Cortex-M3: elapsed time (µs) algorithm

Cortex-M4 STM32F401RE at 84 MHz									Cortex-M3 STM32L152RE at 32 MHz								
STM32CubeIDE 1.2.0			IAR EWARM 8.32.3			Keil µVision 5.27			STM32CubeIDE 1.2.0			IAR EWARM 8.32.3			Keil µVision 5.27		
Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
2	18	793	2	15	658	2	26	1202	19	52	1692	15	32	842	13	29	830

Table 3. Cortex-M7: elapsed time (µs) algorithm

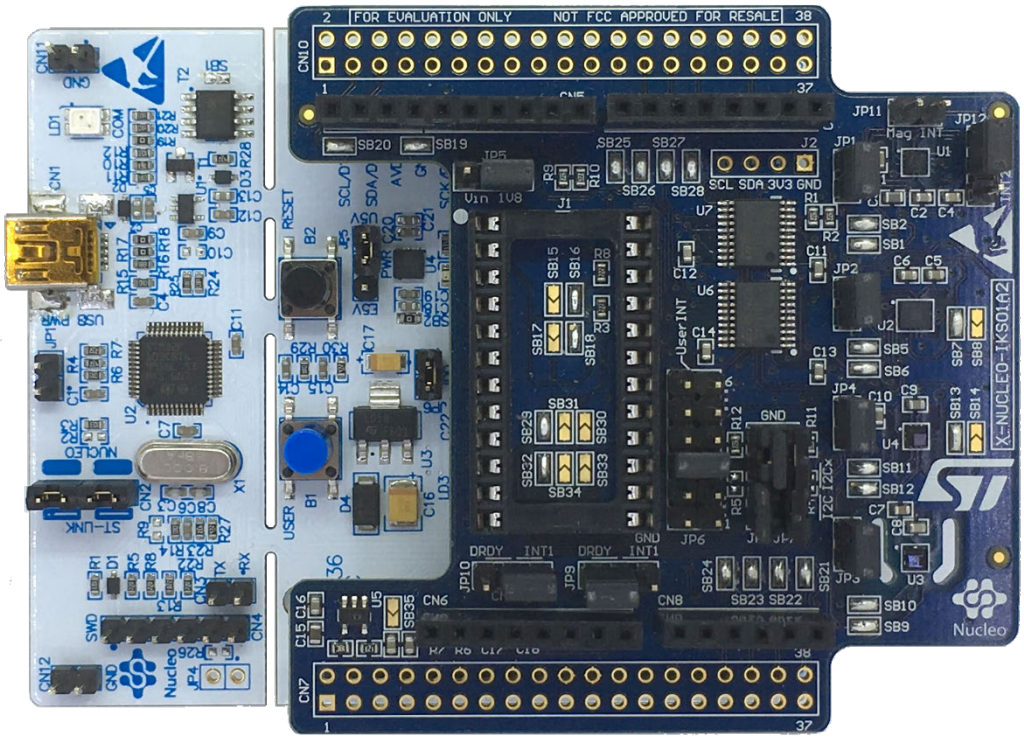
Cortex-M7 STM32F767ZI at 96 MHz								
STM32CubeIDE 1.2.0			IAR EWARM 8.32.3			Keil µVision 5.27		
Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
3	17	683	3	14	545	3	11	380

3 Sample application

The MotionAC middleware can be easily manipulated to build user applications; a sample application is provided in the Application folder.

It is designed to run on a NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L152RE development board connected to an X-NUCLEO-IKS01A2 or X-NUCLEO-IKS01A3 expansion board.

Figure 3. Sensor expansion board and adapter connected to the STM32 Nucleo



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

3.1 Unicleo-GUI application

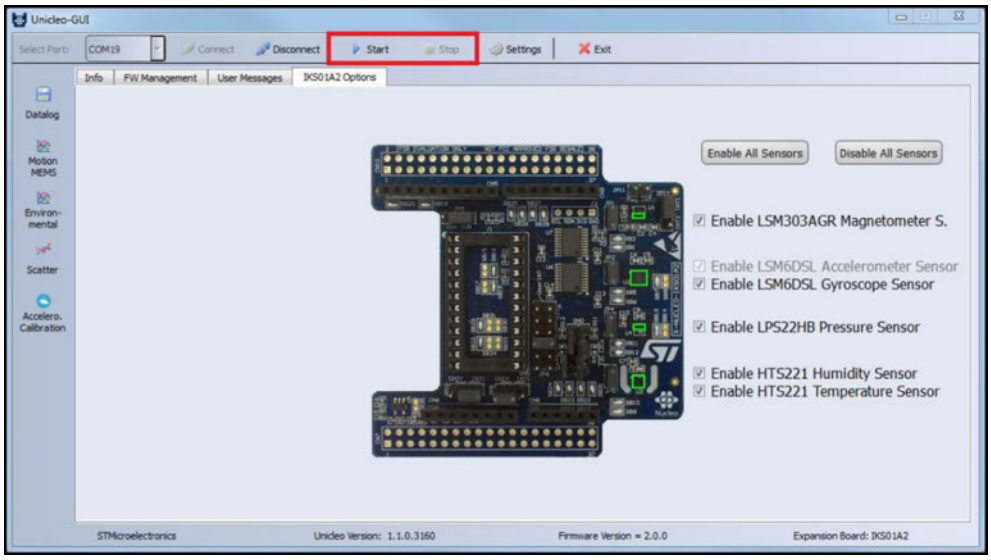
The sample application uses the Windows Unicleo-GUI utility, which can be downloaded from www.st.com.

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

Step 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 is automatically detected and the appropriate COM port is opened.

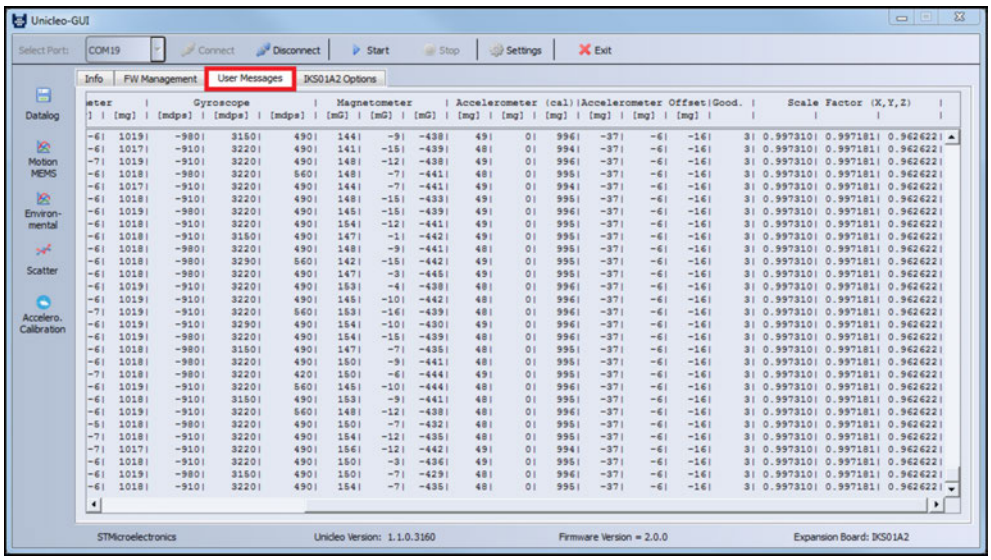
Figure 4. Unicleo main window



Step 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.

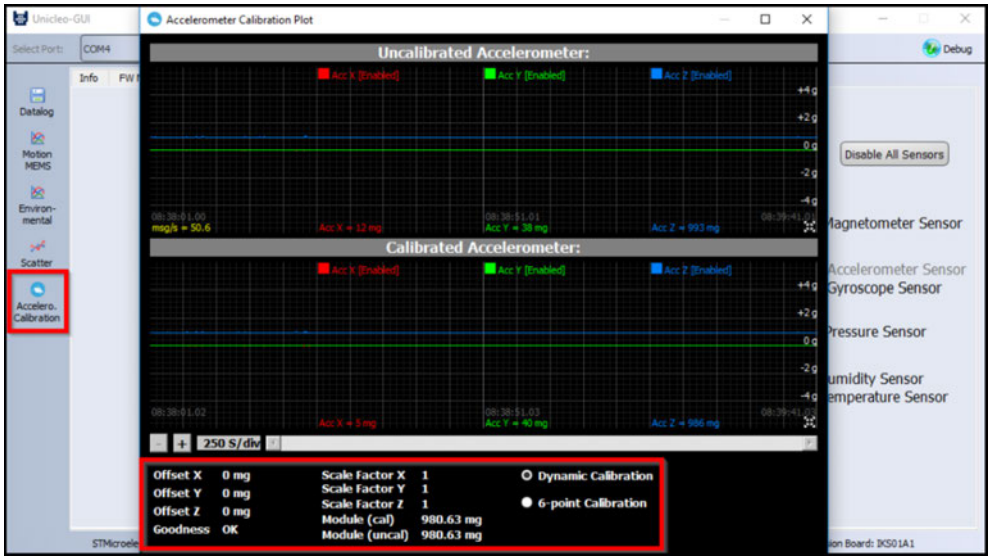
Figure 5. User Messages tab



Step 4. Click on the Accelerometer Calibration icon in the vertical tool bar to open the dedicated application window.

The window is split into one section with uncalibrated data, another with the calibrated data, another section with offset, scale factor and quality of calibration information and the last section with radio button to switch the desired calibration mode.

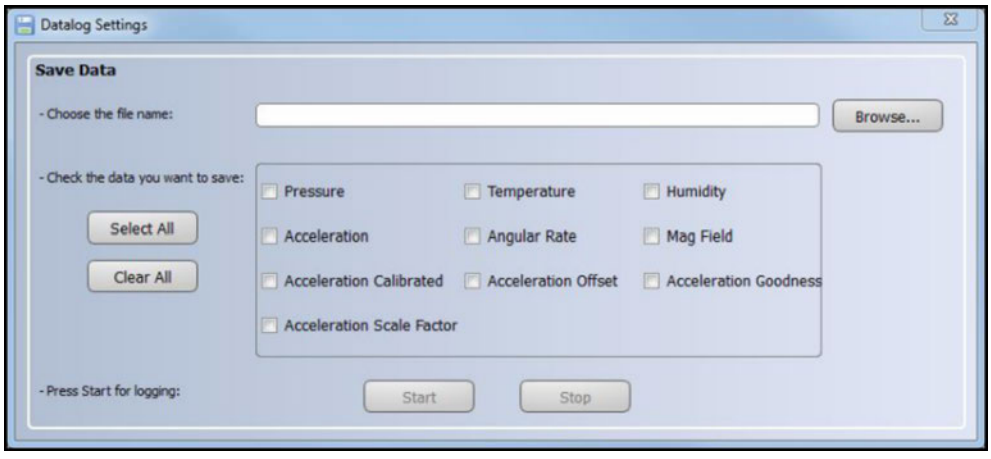
Figure 6. Accelerometer calibration window



Step 5. Click on the Datalog icon in the vertical tool bar to open the datalog configuration window:

you can select which sensor and activity data to save in files. You can start or stop saving by clicking on the corresponding button.

Figure 7. Datalog window



4 References

All of the following resources are freely available on www.st.com.

1. UM1859: Getting started with the X-CUBE-MEMS1 motion MEMS and environmental sensor software expansion for STM32Cube
2. UM1724: STM32 Nucleo-64 board
3. UM2128: Getting started with Unicleo-GUI for motion MEMS and environmental sensor software expansion for STM32Cube

Revision history

Table 4. **Document revision history**

Date	Version	Changes
10-Apr-2017	1	Initial release.
26-Jan-2018	2	Updated Section 2.2.2 MotionAC APIs, Section 2.2.5 Demo code, Section 3.1 Unicleo-GUI application. Added references to NUCLEO-L152RE development board and Section 2.2.7 Algorithm performance.
20-Mar-2018	3	Updated Introduction and Section 2.1 MotionAC overview.
14-Feb-2019	4	Updated Figure 3. Sensor expansion board and adapter connected to the STM32 Nucleo and Table 2. Elapsed time (µs) algorithm. Added X-NUCLEO-IKS01A3 expansion board compatibility information.
24-Mar-2020	5	Updated Introduction, Section 2.2.1 MotionAC library description and Section 2.2.7 Algorithm performance. Added ARM Cortex-M7 architecture compatibility information.

Contents

List of tables

List of figures