Getting started with MotionAC2 2-axis accelerometer calibration library in X-CUBE-MEMS1 expansion for STM32Cube

Introduction

The MotionAC2 is a middleware library part of X-CUBE-MEMS1 software and runs on STM32. It provides real-time 2-axis 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®-M0+, 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 a STEVAL-MKI209V1K MEMS inclinometer kit based on IIS2ICLX 2-axis accelerometer on a NUCLEO-F401RE, NUCLEO-L476RG, NUCLEO-L152RE or NUCLEO-L073RZ 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 MotionAC2 middleware library in X-CUBE-MEMS1 software expansion

2.1 MotionAC2 overview

The MotionAC2 library expands the functionality of the X-CUBE-MEMS1 software.

The library acquires data from the 2-axis accelerometer and calculates the offset and scale factor coefficients together with the calibration quality value. It involves aligning the system in four different orientations according to the gravity direction. The offset and scale factor coefficients are then used to compensate raw data coming from the 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 is described in the document.

Sample implementation is available on the STEVAL-MKI209V1K mounted on a NUCLEO-F401RE, NUCLEO-L476RG, NUCLEO-L152RE or NUCLEO-L073RZ development board.

2.2 MotionAC2 library

Technical information fully describing the functions and parameters of the MotionAC2 APIs can be found in the MotionAC2_Package.chm compiled HTML file located in the Documentation folder.

2.2.1 MotionAC2 library description

The MotionAC2 2-axis accelerometer calibration library manages data acquired from the 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

Note: The maximum sample rate supported by the library is set at 1000 Hz to guarantee a good performance in terms of hardware resources and application needs. The minimum sample rate of 20 Hz is recommended to speed up calibration. By lowering the ODR, sensor calibration time will be increased.

- available for ARM Cortex M0+, Cortex-M3, Cortex-M4 and Cortex-M7 architectures
- resources requirements:
  - Cortex-M0+: 6.0 kB of code and 0.3 kB of data memory
  - Cortex-M3: 5.8 kB of code and 0.3 kB of data memory
  - Cortex-M4: 4.8 kB of code and 0.3 kB of data memory
  - Cortex-M7: 5.2 kB of code and 0.3 kB of data memory

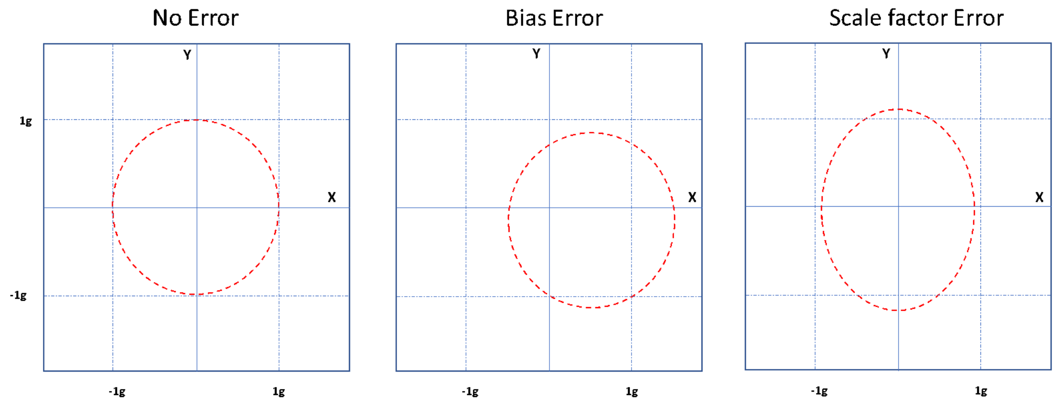
2.2.2 Accelerometer bias and scale factor

When the device is exposed to full gravity, the plot between X and Y axes can show the error in the accelerometer reading.

If the accelerometer reading is successful, the typical plot forms a circle with 1 g radius. Due to bias, the circle center can shift and the scale factor error distorts the circle, forming an ellipse.

The figure below shows the three different plots in the absence or presence of bias and scale factor error.

Figure 1. MotionAC2 accelerometer reading



MotionAC2 library automatically extracts data and exploits ellipse fitting to estimate bias and scale factor to correct accelerometer readings.

Due to bias and scale factor error, the relation between calibrated and raw accelerometer reading can be represented by:

$$X_{true} = a \cdot \left(X_{raw} - X_{off}\right) \tag{1}$$

$$Y_{true} = b \cdot \left(Y_{raw} - Y_{off}\right) \tag{2}$$

Since the total magnitude of acceleration is equal to gravity when the device is stationary:

$$X_{true}^2 + Y_{true}^2 = 1g \tag{3}$$

Rearranging the terms for X_{raw} and Y_{raw}:

$$\left(\frac{X_{raw} - X_{off}}{\left(\frac{1}{a}\right)}\right)^2 + \left(\frac{Y_{raw} - Y_{off}}{\left(\frac{1}{b}\right)}\right)^2 = 1 \tag{4}$$

Replacing 1/a and 1/b with aInv and bInv:

$$\left(\frac{X_{raw} - X_{off}}{a\mathrm{lnv}}\right)^2 + \left(\frac{Y_{raw} - Y_{off}}{b\mathrm{lnv}}\right)^2 = 1 \tag{5}$$

The last equation represents the ellipse equation when the library estimates all four parameters through the optimization method.

2.2.3 MotionAC2 APIs

The MotionAC2 APIs are:

- `void MotionAC2_Init(float *freq)`
  - performs MotionAC2 library initialization and setup of the internal mechanism
  - the CRC module in the STM32 microcontroller (in RCC peripheral clock enable register) has to be enabled before using the library
  - `*freq` parameter is the update frequency (sensor ODR)

Note: This function must be called before using the 2-axis accelerometer calibration library.

- `void MotionAC2_GetKnobs(MAC2_knobs_t *knobs)`
  - gets current knob settings
  - `*knobs` parameter is a pointer to a structure with settings
  - the parameters for the structure type `MAC2_knobs_t` are:
    - `FullScale` parameter is the accelerometer full range in the unit of g (default value is 2 g). It is recommended to set minimum 2 g to perform a successful calibration. A lower `FullScale` saturates the accelerometer signal and impacts the accuracy
    - `CalDuration_s` parameter is the expected duration of the calibration routine execution (default is 180 s). The library rejects the calibration motion if the overall motion is performed in more time than the set duration. The `CalDuration_s` is an overlapping window. The duration ensures the bias drift due to temperature does not affect the computation
    - `XlNoiseScale` parameter is the factor on noise of the accelerometer to detect the static condition to be used for calibration. It is recommended to set the value between 0.5 – 2 (default is 1); a higher value reduces the accuracy of calibration parameters. In some applications, the accelerometer noise might be higher due to vibration and requires setting a higher value. The value of `XlNoiseScale` derives from computing the variance of accelerometer data (single axis) at stationary and dividing by 5e-6 g2. Set the value with some positive margin (for example, if the computed value is around 0.8, set 0.85 or 0.9).

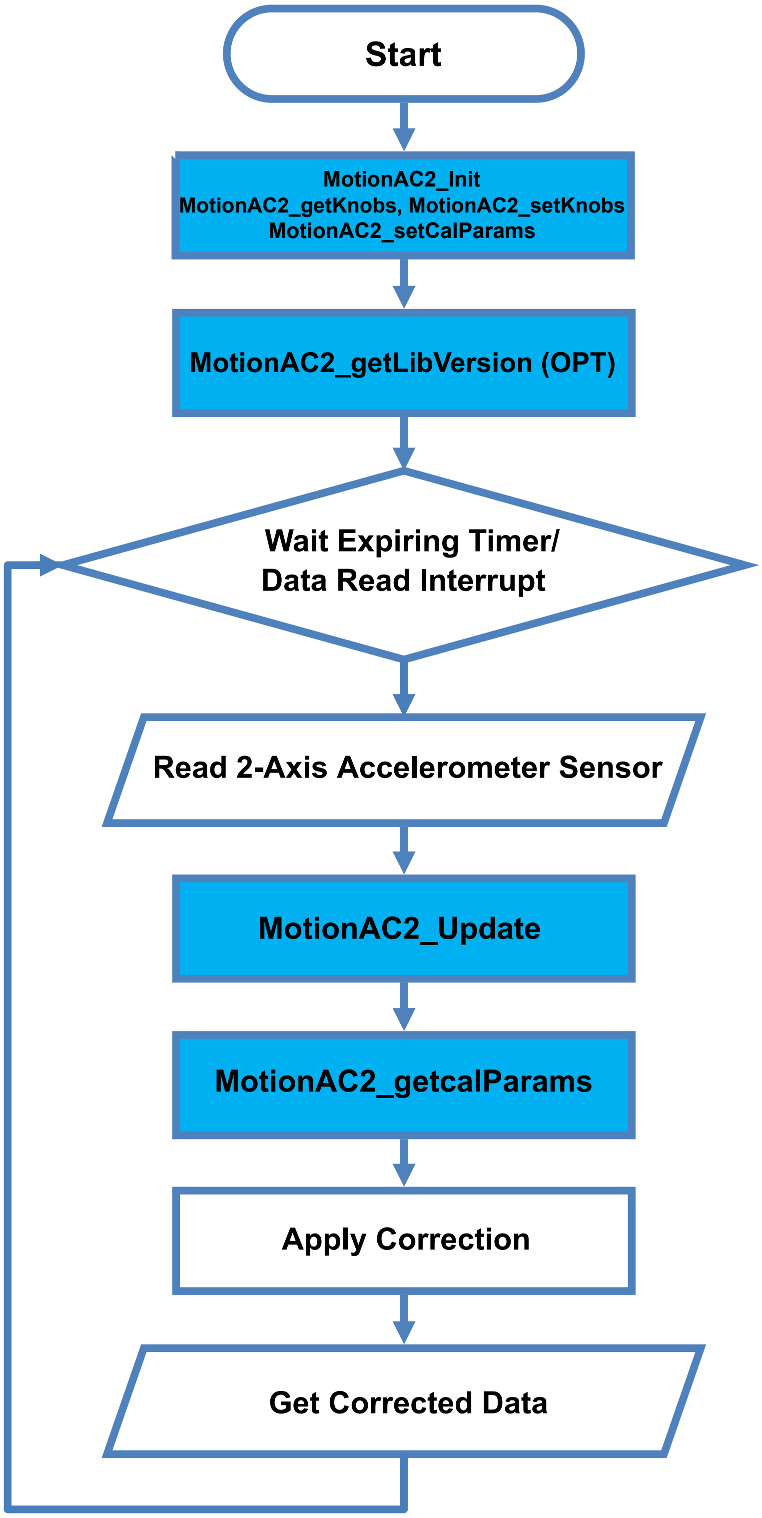
- `void MotionAC2_SetKnobs(MAC2_knobs_t*knobs)`
  - sets current knob settings
  - `*knobs` parameter is a pointer to a structure with settings (see `MotionAC2_GetKnobs`)
- `uint8_t MotionAC2_Update(MAC2_input_t *data_in, uint64_t timestamp_ms)`
  - executes 2-axis accelerometer calibration algorithm
  - `*data_in` parameter is a pointer to a structure with input data
  - the parameters for the structure type `MAC2_input_t` are:
    - - `Acc_X` is an accelerometer X axis value in g
      - `Acc_Y` is an accelerometer Y axis value in g
  - `timestamp_ms` parameter is a timestamp of the current sample in ms
  - returns 1 in case of calibration is done with the current sample, 0 otherwise

Note: This function must be called periodically at the same interval indicated in `MotionAC2_Init`.

- `void MotionAC2_GetCalParams(MAC2_cal_params_t *cal_params)`
  - retrieves the accelerometer calibration coefficients for offset and scale factor compensation and calibration quality factor
  - `*cal_params` parameter is a pointer to a structure with calibration parameters
  - the parameters for the structure type `MAC2_cal_params_t` are:
    - `Bias[2]` is an array of the offset for each axis in g
    - `SF[2]` is the scale factor for each axis
    - `CalStatus` is the calibration status:
      - `MAC2_CAL_UNKNOWN = 0` - accuracy of calibration parameters is unknown
      - `MAC2_CAL_POOR = 1` - accuracy of calibration parameters is poor, cannot be trusted, expected accuracy ~20 deg
      - `MAC2_CAL_OK = 2` - accuracy of calibration parameters is OK, expected accuracy ~2.5 deg
      - `MAC2_CAL_GOOD = 3` - accuracy of calibration parameters is good, expected accuracy ~0.5 deg
- `void MotionAC2_SetCalParams(MAC2_cal_params_t *cal_params)`
  - sets the initial accelerometer calibration coefficients from the last run if any
  - `*cal_params` parameter is a pointer to a structure with calibration parameters
- `uint8_t MotionAC2_GetLibVersion(char *version)`
  - retrieves the library version
  - `*version` is a pointer to an array of 35 characters
  - returns the number of characters in the version string

2.2.4 API flow chart

Figure 2. MotionAC2 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 ALGO_FREQ    100.0f

[…]

/*** Initialization ***/

char lib_version[VERSION_STR_LENG];

MAC2_knobs_t Knobs;

/* 2-Axis Accelerometer calibration API initialization function */

MotionAC2_Init(ALGO_FREQ);

/* Optional: Get version */

MotionAC2_GetLibVersion(lib_version);

/* Optional: Adjust Knobs settings Fulls Scale = 2g, Higher noise in the system */

MotionAC2_GetKnobs(&Knobs);

Knobs.FullScale = 2.0f;
Knobs.XlNoiseScale  = 1.1f;

MotionAC2_SetKnobs(&Knobs);


[…]

/*** Using 2-axis accelerometer calibration algorithm ***/


Timer_OR_DataRate_Interrupt_Handler()

{

MAC2_input_t data_in;

MAC2_output_t data_out;

float acc_cal[2];

/* Get acceleration X/Y in g */

MEMS_Read_AccValue(&data_in.Acc_X, &data_in.Acc_Y);

/* Accelerometer calibration algorithm update */

MotionAC2_Update(&data_in);

/* Get Calibration coeficients */

MotionAC_GetCalParams(&data_out);


/* Apply correction */

acc_cal[0] = (data_in.Acc_X - data_out.Bias[0]) * data_out.SF[0];
acc_cal[1] = (data_in.Acc_Y - data_out.Bias[1]) * data_out.SF[1];
}
```

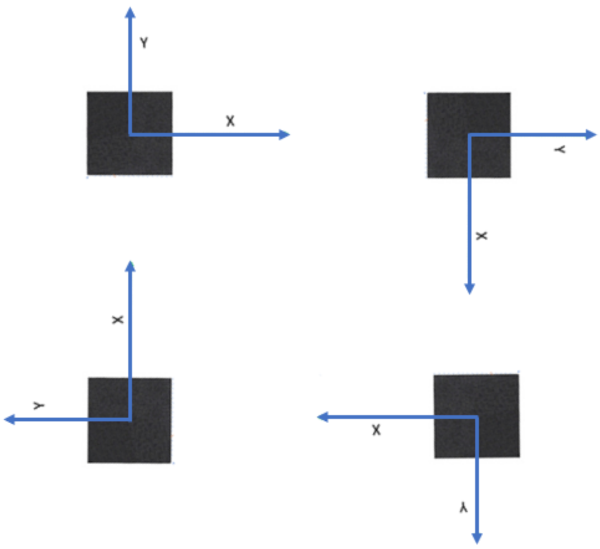
2.2.6 Calibration process

The calibration process is based on the 4-point tumble calibration of the 2-axis accelerometer sensor when exposed to Earth gravitation field. The data is collected at the stationary condition.

During calibration, the device must be placed in four different orientations at least.

Calibration can be performed by placing the device in four different directions with respect to gravity. For example, the device can be placed with ±X, ±Y.

Figure 3. **Calibration motion**



You can place the device in any orientation without the need of putting it in a perfect flat position but ensuring the device remains in a stationary position for at least 3 seconds before changing its orientation.

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 1.3.0			IAR EWARM 8.32.3			Keil µVision 5.27			STM32CubeIDE 1.3.0			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	2	133	2	2	88	2	3	123	20	41	3253	17	27	1958	15	29	3276

Table 3. **Cortex-M0+ and Cortex-M7: elapsed time (µs) algorithm**

Cortex-M0+ STM32L073RZ at 32 MHz									Cortex-M7 STM32F767ZI at 96 MHz								
STM32CubeIDE 1.3.0			IAR EWARM 8.32.3			Keil µVision 5.27			STM32CubeIDE 1.3.0			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
<1000	<1000	10000	<1000	<1000	4000	<1000	<1000	5000	4	4	129	3	3	99	3	3	226

3 Sample application

The MotionAC2 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, NUCLEO-L152RE or NUCLEO-L073RZ development board connected to the STEVAL-MKI209V1K.

Accelerometer algorithm output data can be displayed in real-time through Unicleo-GUI.

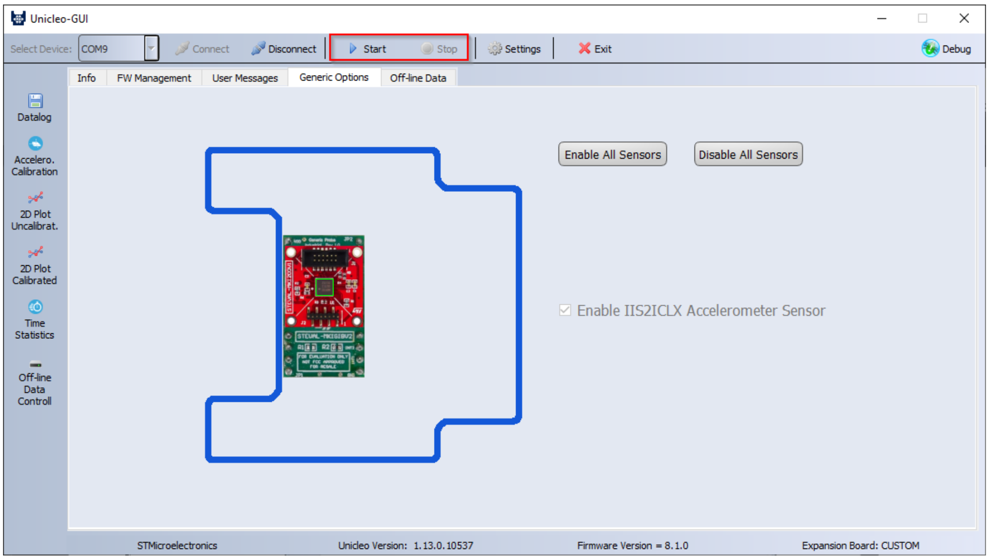
3.1 Unicleo-GUI application

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.

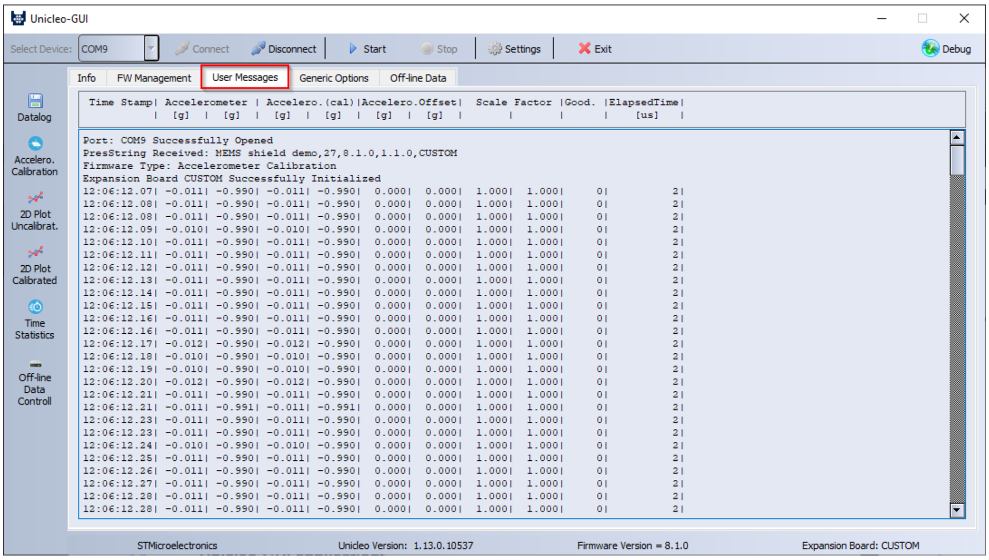
Step 2. Launch the Unicleo-GUI application to open the main application window.
If an STM32 Nucleo board with supported firmware is connected to the PC, it is automatically detected and the appropriate COM port is opened.

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 [**Accelero. Calibration**] icon in the vertical tool bar to open the dedicated application window.

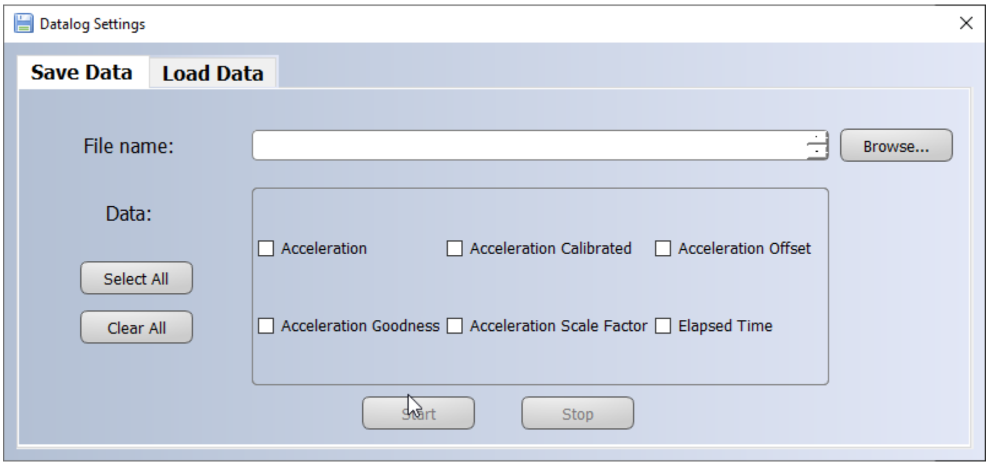
The window is split into different sections including uncalibrated data, calibrated data, offset, scale factor and quality of calibration. The calibration process can be invoked by pressing the [**Start Calibration**] button.

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.

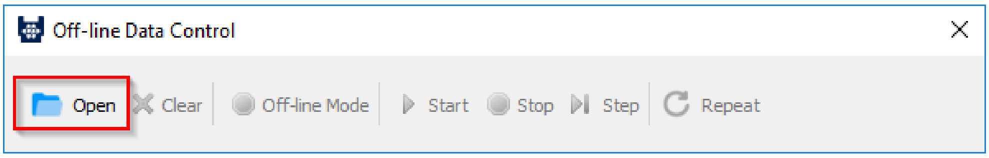
Figure 7. Datalog window



Step 6. To process the previously captured data, click on the [**Offline Data Control**] icon in the vertical tool bar and open the dedicated window.

The data are processed by the MCU firmware.

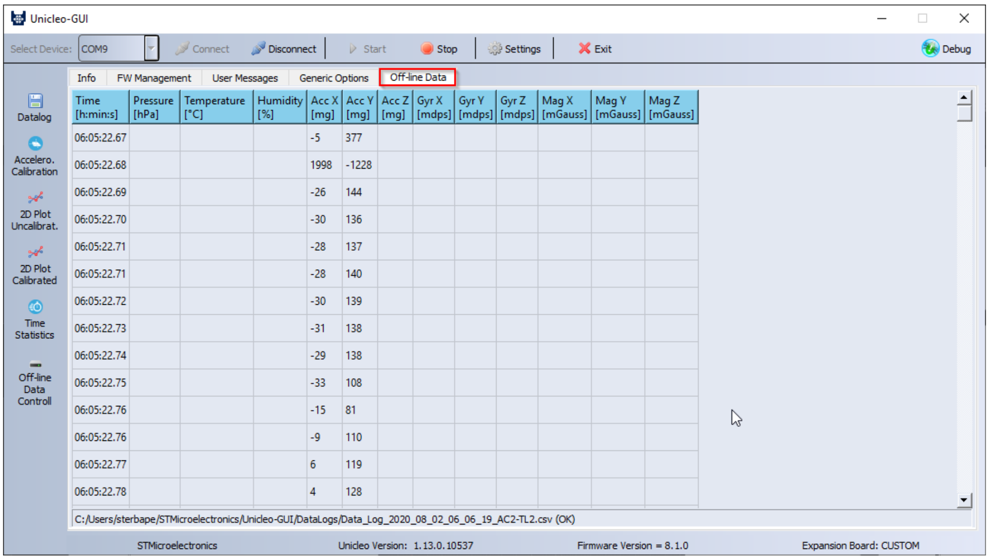
Figure 8. Offline Data Control menu bar



Step 7. Click on the [**Open**] button to select the file with offline data in CSV format.

The data are loaded into the Offline Data tab.

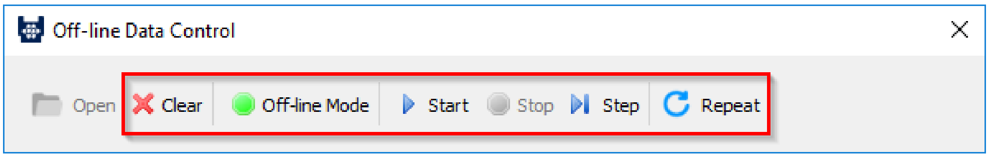
Figure 9. Offline Data Control window



Other buttons in the Offline Data Control window are activated. You can click on:

– [**Offline Mode**] button to switch the firmware offline mode on/off.
– [**Start**]/[**Stop**]/[**Step**]/[**Repeat**] buttons to control the data sent by Unicleo-GUI to the firmware.
– [**Clear**] button to remove data from Unicleo-GUI.

Figure 10. Offline Data Control window - other buttons



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 boards (MB1136)
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
08-Sep-2020	1	Initial release.

Contents

List of tables

List of figures