Getting started with MotionDI dynamic inclinometer library in X-CUBE-MEMS1 expansion for STM32Cube

Introduction

The MotionDI middleware library is part of the X-CUBE-MEMS1 software and runs on STM32. It provides real-time information about device orientation, including tilt information, and can also perform accelerometer and gyroscope calibration.

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®-M33, ARM® Cortex®-M4 or ARM® Cortex®-M7 architecture.

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

The software comes with a sample implementation running on X-NUCLEO-IKS01A3, X-NUCLEO-IKS4A1 or X-NUCLEO-IKS02A1expansion board on a NUCLEO-F401RE, NUCLEO-U575ZI-Q 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 MotionDI middleware library in X-CUBE-MEMS1 software expansion for STM32Cube

2.1 MotionDI overview

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

The library acquires data from the accelerometer and gyroscope and provides information about the device position (quaternions, Euler angles, linear acceleration, gravity vector).

The MotionDI filtering and predictive software uses advanced algorithms to intelligently integrate outputs from multiple MEMS sensors for optimum performance regardless of environmental conditions.

The library is designed for ST MEMS only. Functionality and performance when using other MEMS sensors have not been tested and can vary significantly from documented behavior.

A sample implementation is available for X-NUCLEO-IKS01A3, X-NUCLEO-IKS4A1 or X-NUCLEO-IKS02A1 expansion boards, mounted on a NUCLEO-F401RE, NUCLEO-U575ZI-Q or NUCLEO-L152RE development board.

2.2 MotionDI library

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

2.2.1 MotionDI library description

The MotionDI dynamic inclinometer library manages the data acquired from the accelerometer and gyroscope sensor; it features:

- real-time 6-axis motion sensor data fusion (accelerometer, gyroscope)
- computation of rotation vector, quaternions, gravity and linear acceleration data
- gyroscope bias calibration
- accelerometer bias and scale calibration
- recommended sensor data sampling frequency of 100 Hz
- resource requirements:
  - Cortex-M3: 55.9 kB of code and 7.3 kB of data memory
  - Cortex-M33: 47.1 kB of code and 7.3 kB of data memory
  - Cortex-M4: 48.2 kB of code and 7.3 kB of data memory
  - Cortex-M7: 47.4 kB of code and 7.3 kB of data memory

Note: Real size might differ for different IDEs (toolchain)

  - available for ARM® Cortex®-M3, ARM Cortex®-M33, ARM® Cortex®-M4 and ARM® Cortex®-M7 architectures

2.2.2 MotionDI library operation

The MotionDI library implements a sensor fusion algorithm for the estimation of 3D orientation in space. It uses a digital filter based on the Kalman filter theory to merge data from several sensors and compensate for the limitations of the single sensors. For instance, as gyroscope data drift can impact the orientation estimates, the accelerometer can be used to provide absolute tilt orientation information.

The MotionDI library integrates sensor fusion and calibration algorithms in one library; the calibration functionality can be enabled or disabled using the knob setting.

2.2.3 MotionDI library parameters

The library is based on parameters pertaining to an `MDI_knobs_t` structure.

The parameters `MDI_fusion_knobs_t` for the structure are:

- `ATime`, `FrTime`: represent the weighting stability of sensors for prediction (trust factor), from 0.5 to 5. Default values are recommended.
  - `ATime`: lowering the value increases the accelerometer weight and sensitivity towards external acceleration.
  - `FrTime`: lowering the value increases the trust on gyroscope data and lower the correction from the accelerometer.
- `modx`: represents the decimation of `MotionDI_update` call frequency
- `output_type`: represents the sensor fusion library output orientation: 0 = NED, 1 = ENU

The parameters `MDI_acc_cal_knobs_t` for the structure are:

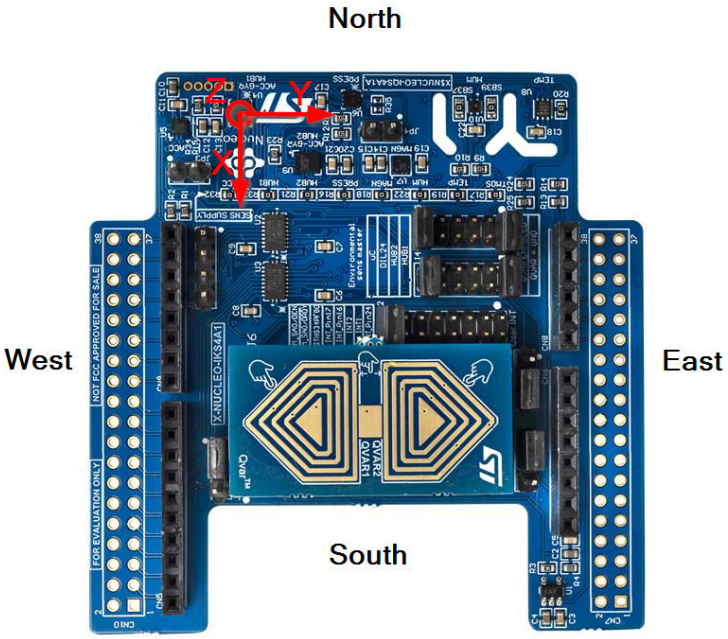
- `MoveThresh_g`: the maximum motion allowed during the calibration. Recommended in the range of 0.1-0.30 g. The expected accuracy of bias is the same as `MoveThresh_g` parameters
- `CalType`: the library can be configured to run the calibration and the corresponding frequency.
  - `MDI_CAL_NONE`: disables the calibration
  - `MDI_CAL_ONETIME`: runs the calibration one time and then disables it when it reaches a good calibration quality
  - `MDI_CAL_CONTINUOUS`: continuously runs the calibration.

The parameters `MDI_gyro_cal_knobs_t` for the structure are:

- `AccThr`: is the accelerometer threshold to detect steady state in g. The default value is 0.001 g. The input range is 0.0005 to 0.01 g. For higher accuracy, set the value low; for platforms inherent vibration or noisy sensors, set the value high.
- `GyroThr`: is the gyroscope threshold to detect steady state in degrees per second. The default value is 0.25 dps. The input range is 0.008 to 0.4 dps. For higher accuracy, set the value low.
- `MaxGyro`: is the maximum expected angular rate offset when still in [dps]. The default value is 15 dps.
- `GBiasDiffAVTh`: is the gyroscope threshold in dps unit. The default value of this variable is 0.33, the range of the input is 0.2 to 0.5. For higher accuracy, set the value low.
- `GBiasDiffVelTh`: is the accelerometer threshold in g unit. The default value of this variable is 0.033, the range of the input is 0.02 to 0.05. For higher accuracy, set the value low. Both variables (`GBiasDiffAVTh` and `GBiasDiffVelTh`) represent the minimum change in acceleration and angular velocity required to detect stationary condition.
- `CalType`: calibration type

As shown in the figure below, the X-NUCLEO-IKS4A1 accelerometer sensor has an SEU (x - South, y - East, z - Up), so the string is: “seu”.

Figure 1. Example of sensor orientations



2.2.4 MotionDI library output data rate

Set up the dynamic inclinometer library output data rate (ODR) properly. Although the ODR can be set to a higher value, 100 Hz is recommended.

2.2.5 Sensor calibration in the MotionDI library

Accelerometer calibration

Accelerometer calibration can improve the accuracy of tilt angle and is recommended for applications which demand a very accurate orientation. It aligns the system in six positions according to the gravity direction.

The accelerometer calibration can be configured to run one time since the calibration parameter drift is very low. Calibration can be done by placing the device in 6 different directions with respect to gravity. For example, device can be placed with ±X, ±Y, ±Z.

Gyroscope calibration

Gyroscope calibration is handled automatically by the MotionDI library by continuously compensating the zero-rate offset effect.

2.2.6 MotionDI APIs

The MotionDI APIs are:

- `uint8_t MotionDI_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 MotionDI_Initialize(float *freq)`
  - performs MotionDI library initialization and setup of the internal mechanism
  - `freq` is frequency at which the IMU data is available to library

Note: This function must be called before using the sensor fusion library and the CRC module in the STM32 microcontroller (in RCC peripheral clock enable register) has to be enabled.

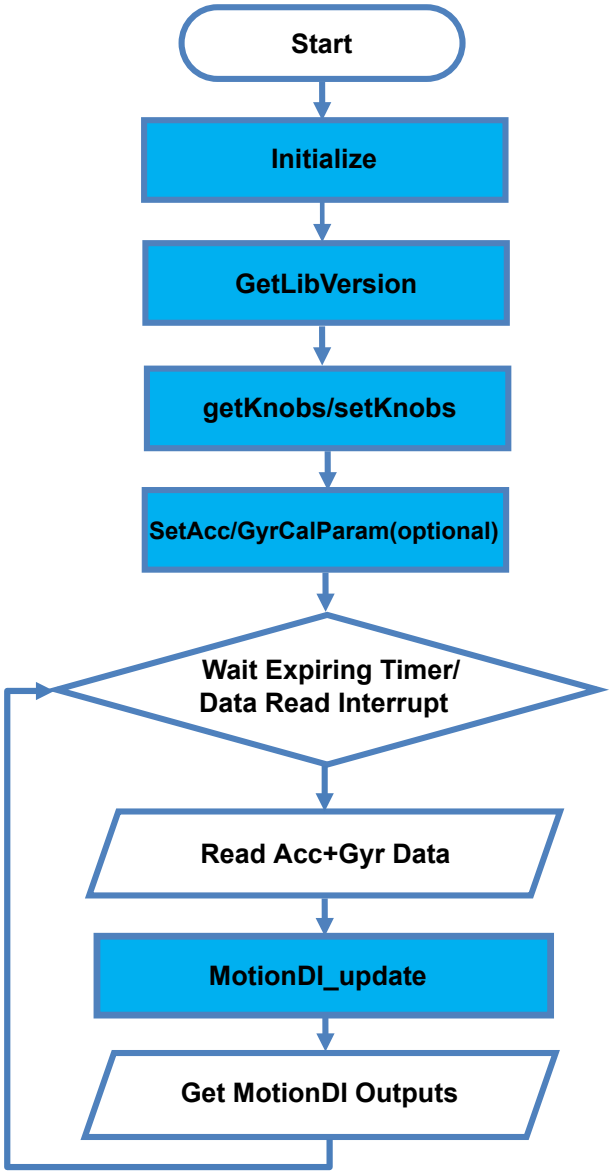
- `void MotionDI_setKnobs(MDI_knobs_t *knobs)`
  - sets the internal knobs
  - `knobs` is a pointer to a structure with knobs
- `void MotionDI_getKnobs(MDI_knobs_t *knobs)`
  - gets the current internal knobs
  - `knobs` is a pointer to a structure with knobs
- `void MotionDI_AccCal_getParams(MDI_cal_output_t *acc_cal)`
  - gets the accelerometer calibration parameters
  - `acc_cal` is a pointer to a structure with accelerometer calibration parameters. The accelerometer calibration parameters contain bias, scale factor and calibration quality.
- `void MotionDI_AccCal_setParams(MDI_cal_output_t *acc_cal)`
  - sets the accelerometer calibration parameters
  - `acc_cal` is a pointer to a structure with accelerometer calibration parameters. The accelerometer calibration parameters contain bias, scale factor and calibration quality.
    This function should be called after MotionDI_Initialize but before calling MotionDI_update function.
- `void MotionDI_AccCal_reset(void)`
  - restarts accelerometer calibration algorithm
- `void MotionDI_GyrCal_getParams(MDI_cal_output_t*gyro_cal)`
  - gets the gyroscope calibration parameters
  - `gyro_cal` is a pointer to a structure with gyroscope calibration parameters. The gyroscope calibration parameters contain bias, and calibration quality. The scale factor remains 1.
- `void MotionDI_GyrCal_setParams(MDI_cal_output_t *gyro_cal)`
  - sets the gyroscope calibration parameters
  - `gyro_cal` is a pointer to a structure with gyroscope calibration parameters. The gyroscope calibration parameters contain bias, and calibration quality. The scale factor remains 1.
    This function should be called after MotionDI_Initialize but before calling MotionDI_update function.
- `void MotionDI_GyrCal_reset(void)`
  - restarts gyroscope calibration algorithm
- `void MotionDI_update(MDI_output_t *data_out, MDI_input_t *data_in)`
  - runs the update
  - `data_out` is a pointer to output data structure
  - `data_in` is a pointer to input data structure

2.2.7 Portability to a generic microcontroller

The MotionDI library does not have any dependency on external libraries. It requires an FPU enabled on the MCU.

2.2.8 API flow chart

Figure 2. MotionDI API logic sequence



2.2.9 Demo code

The following demonstration code reads data from the accelerometer and gyroscope sensors and gets the rotation, quaternions, gravity and linear acceleration.

```
[…]

#define VERSION_STR_LENG 35

[…]

/* Initialization */
char lib_version[VERSION_STR_LENG];
float freq = 100.0f;
MDI_knobs_t iKnobs;

/* Dynamic Inclinometer API initialization function */
MotionDI_Initialize(&freq);
```

```
/* OPTIONAL */
/* Get library version */
MotionDI_GetLibVersion(lib_version);

/* OPTIONAL */
MotionDI_getKnobs(&iKnobs);
/* Modify knobs settings */
MotionDI_setKnobs(&iKnobs);

[…]

/* Using Dynamic Inclinometer algorithm */
Timer_OR_DataRate_Interrupt_Handler()
{
  MDI_input_t data_in;
  MDI_output_t data_out;

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

  /* Get angular rate X/Y/Z in dps */
  MEMS_Read_GyroValue(data_in.Gyro[0], data_in.Gyro[1], data_in.Gyro[2]);

  data_in.Timestamp = CurrentTime;

  /* Run Dynamic Inclinometer algorithm */
  MotionDI_update(&data_out, &data_in);
}
```

2.2.10 Algorithm performance

The dynamic inclinometer algorithm uses data from the accelerometer and gyroscope sensors and runs at 100 Hz frequency.

Table 2. Algorithm elapse time (µs) Cortex-M4, Cortex-M3

Cortex-M4 STM32F401RE at 84 MHz			Cortex-M3 STM32L152RE at 32 MHz		
Min	Avg	Max	Min	Avg	Max
2	1100	2454	6	3500	27571

Table 3. Algorithm elapse time (µs) Cortex-M33 and Cortex-M7

Cortex- M33 STM32U575ZI-Q at 160 MHz			Cortex- M7 STM32F767ZI at 96 MHz		
Min	Avg	Max	Min	Avg	Max
1	530	997	860	1750	2899

2.3 Sample application

The MotionDI 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-U575ZI-Q or NUCLEO-L152RE development board connected to an X-NUCLEO-IKS01A3, X-NUCLEO-IKS4A1 or X-NUCLEO-IKS02A1 expansion board.

The application provides motion sensor data fusion (quaternions, Euler angles, linear acceleration, gravity vector) and tilt information in real-time or via received offline data. It also performs accelerometer and gyroscope calibration. Data can be displayed through a GUI.

USB cable connection is required to monitor real-time data or feed library with offline data. The board is powered by the PC via USB connection.

This working mode allows the user to display motion sensor fusion data and tilt information, accelerometer, and gyroscope data, timestamp and eventually other sensor data, in real time, using the MEMS-Studio.

2.4 MEMS-Studio application

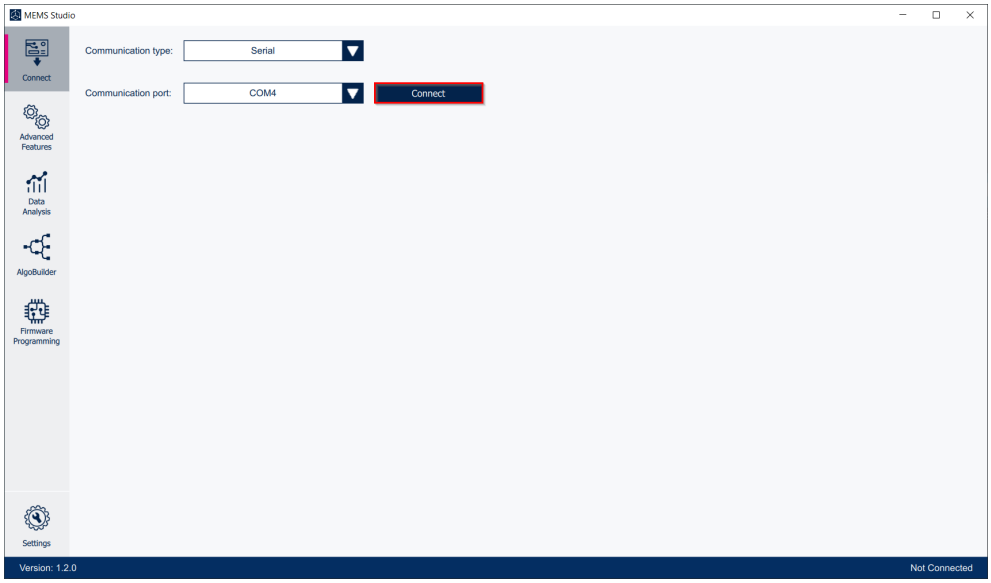
The sample application uses MEMS-Studio application, which can be downloaded from www.st.com.

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 MEMS-Studio application to open the main application window.

If an STM32 Nucleo board with supported firmware is connected to the PC, it is automatically detected. Press the [**Connect**] button to establish connection to the evaluation board.

Figure 3. MEMS-Studio - Connect

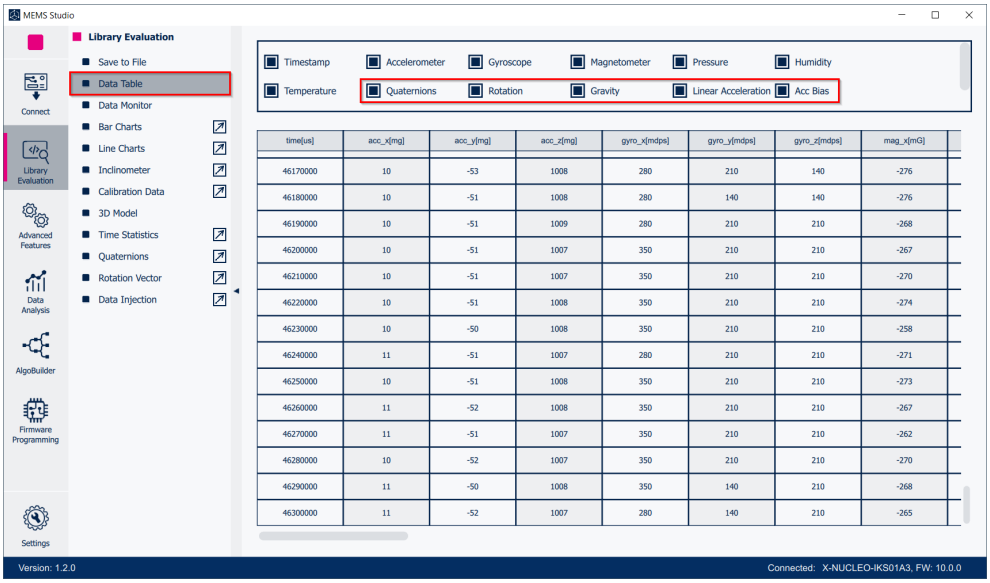


Step 3. When connected to a STM32 Nucleo board with supported firmware [**Library Evaluation**] tab is opened.

To start and stop data streaming, toggle the appropriate [**Start**] or [**Stop**] button on the outer vertical tool bar.

The data coming from the connected sensor can be viewed selecting the [**Data Table**] tab on the inner vertical tool bar.

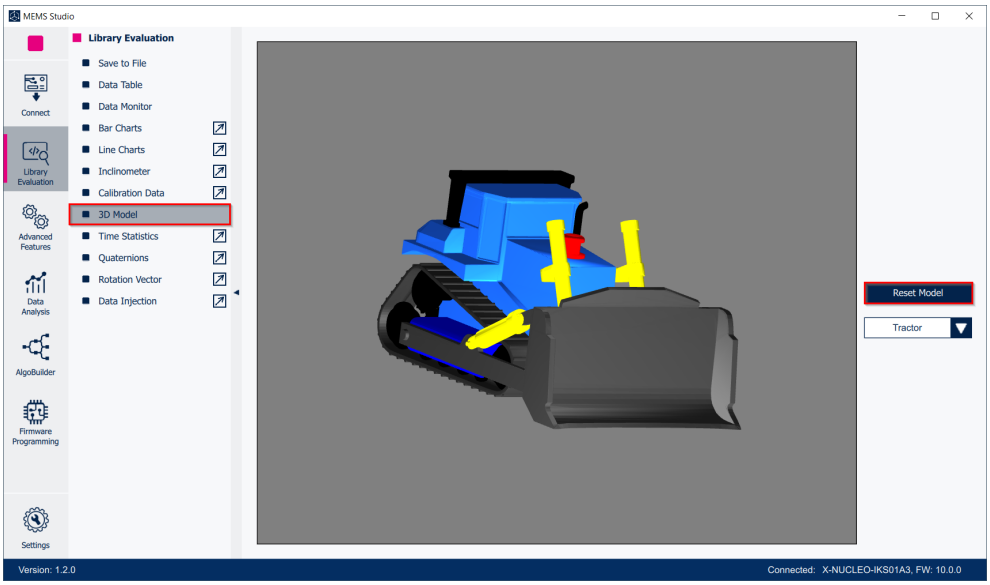
Figure 4. MEMS-Studio - Library Evaluation - Data Table



Step 4. Click on the [**3D Model**] to open the application window displaying the vehicle model.

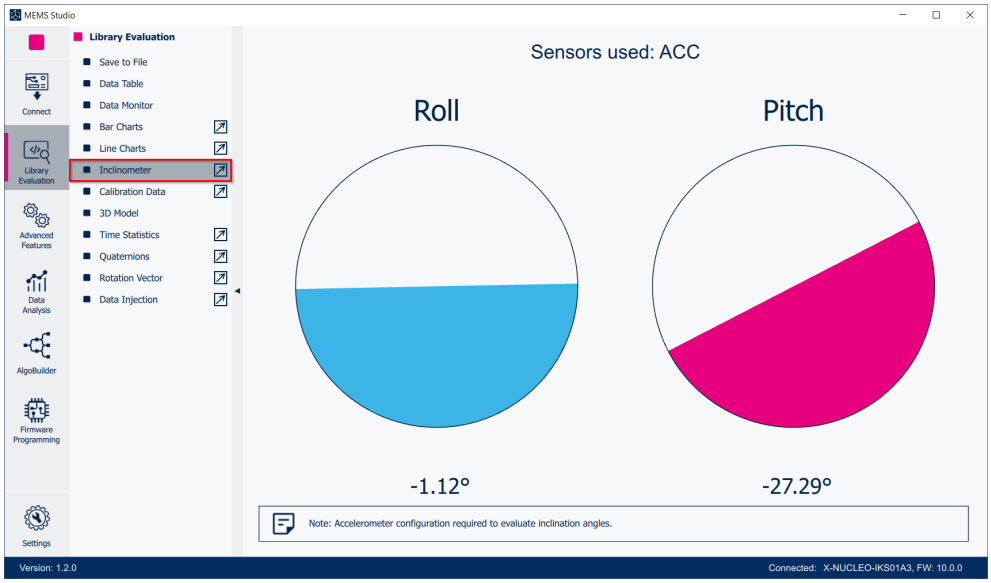
To align the vehicle model, point the STM32 Nucleo board towards the screen and press the Reset model button.

Figure 5. MEMS-Studio - Library Evaluation - 3D Model



Step 5. Click on the [**Inclinometer**] to open the application window displaying tilt angles.

Figure 6. MEMS-Studio - Library Evaluation - Inclinometer



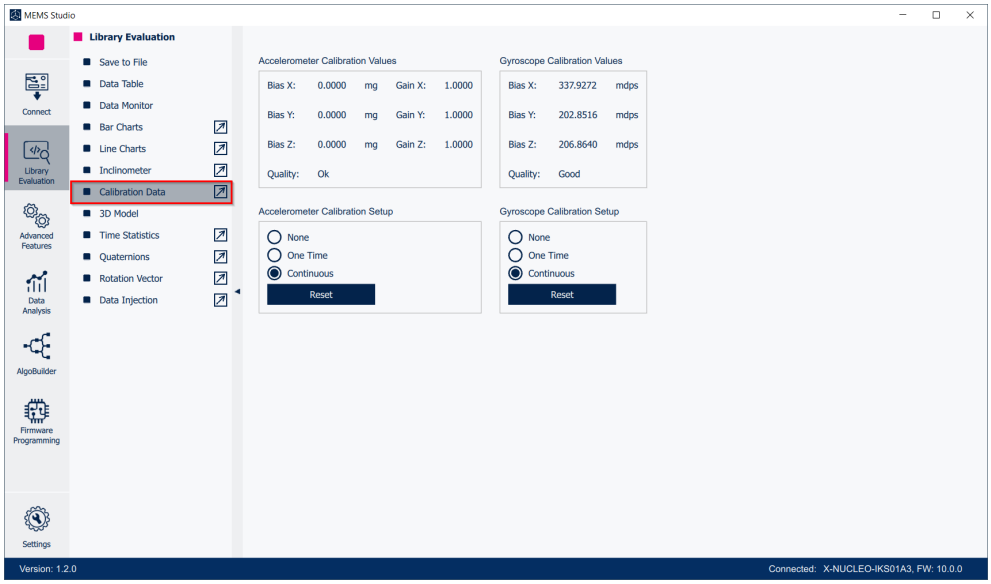
Step 6. Click on the [**Rotation Vector**] to open the application window displaying the rotation vector graph.

Figure 7. MEMS-Studio - Library Evaluation - Rotation Vector

Step 7. Click on the [**Calibration Data**] to open the application window displaying accelerometer and gyroscope calibration status.

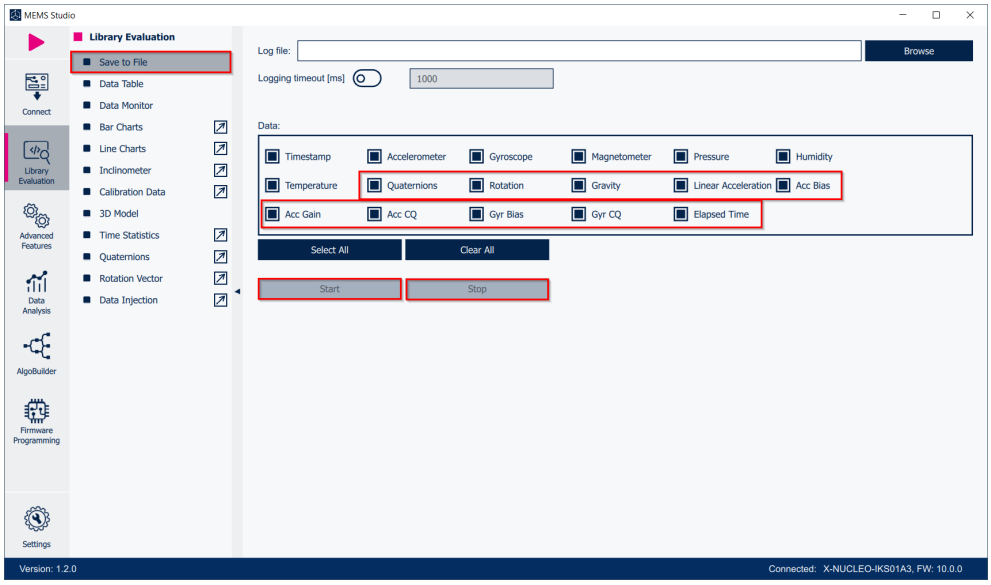
To switch between calibration modes click on the appropriate button.
To reset calibration press the [**Reset**] button.

Figure 8. MEMS-Studio - Library Evaluation - Calibration Data



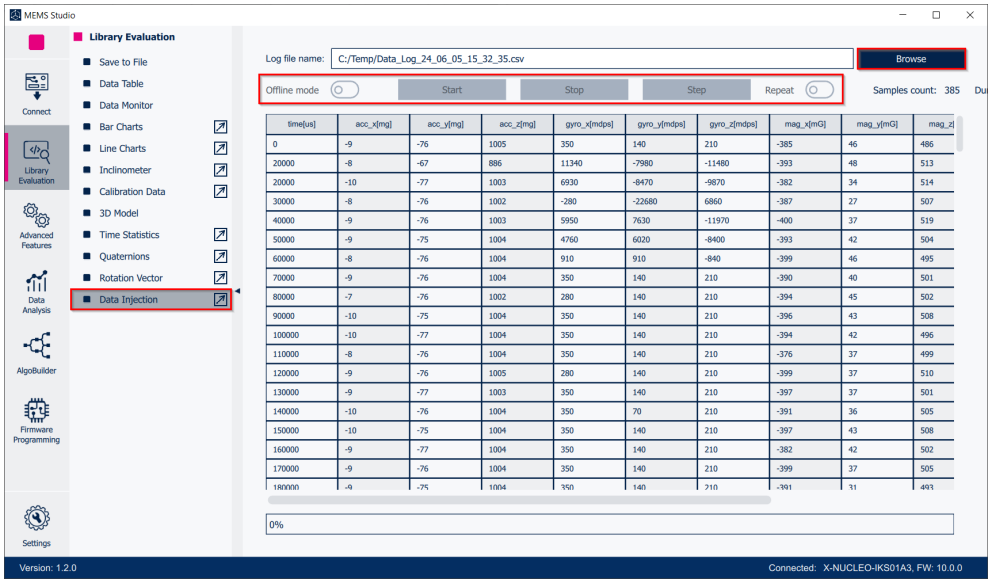
Step 8. Click on the [**Save To File**] to open the dataloging configuration window. Select the sensor and dynamic inclinometer data to be saved in the file. You can start or stop saving by clicking on the corresponding button.

Figure 9. MEMS-Studio - Library Evaluation - Save To File



Step 9. Data Injection mode can be used to send the previously acquired data to the library and receive the result. Select the [**Data Injection**] tab on the vertical tool bar to open the dedicated view for this functionality.

Figure 10. MEMS-Studio - Library Evaluation - Data Injection



Step 10. Click on the [**Browse**] button to select the file with the previously captured data in CSV format.

The data will be loaded into the table in the current view.
Other buttons will become active. You can click on:

– [**Offline Mode**] button to switch the firmware offline mode on/off (mode utilizing the previously captured data).

– [**Start**]/[**Stop**]/[**Step**]/[**Repeat**] buttons to control the data feed from MEMS-Studio to the library.

3 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. UM3233: Getting started with MEMS-Studio

Revision history

Table 4. **Document revision history**

Date	Version	Changes
12-May-2020	1	Initial release.
17-Sep-2024	2	Updated Section Introduction, Section 2.1: MotionDI overview, Section 2.2.1: MotionDI library description, Section 2.2.3: MotionDI library parameters, Section 2.2.10: Algorithm performance, Section 2.3: Sample application, Section 2.4: MEMS-Studio application

Contents

List of tables

List of figures

IMPORTANT NOTICE – 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 acknowledgment.

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. For additional information about ST trademarks, refer to www.st.com/trademarks. 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.

© 2024 STMicroelectronics – All rights reserved