# Getting started with MotionVC vertical context library in X-CUBE-MEMS1 expansion for STM32Cube

## Introduction

The MotionVC is a middleware library part of X-CUBE-MEMS1 software and runs on STM32. It provides real-time information about vertical movement. The library is able to detect a change of altitude and distinguish the type of vertical movement: stairs, elevator, and escalator.

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 eases portability across different STM32 microcontrollers.

The software comes with sample implementation running on 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 MotionVC middleware library for X-CUBE-MEMS1 software expansion for STM32Cube

## 2.1 MotionVC overview

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

The library acquires data from the accelerometer and pressure sensor, detects changes of altitude and distinguish type of vertical movement: stairs, elevator, and escalator. The library is able to adapt to different noise floor of the pressure sensor data.

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 for the X-NUCLEO-IKS01A2 and X-NUCLEO-IKS01A3 expansion board, mounted on a NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L152RE development board.

## 2.2 MotionVC library

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

### 2.2.1 MotionVC library description

The MotionVC vertical context library manages data acquired from accelerometer and pressure sensor; it features:

- vertical movement detection: on floor, up/down
- type of vertical movement detection: stairs, elevator, escalator
- drift free altitude and vertical velocity with confidence parameter calculation
- inbuilt step detection mode
- automatic adaptation to different noise floor of pressure sensor data
- required accelerometer data sampling frequency of 50 Hz and pressure sensor sampling frequency of 10 Hz
- resources requirements:
  - Cortex-M3: 12.3 kB of code and 4.0 kB of data memory
  - Cortex-M4: 11.7 kB of code and 4.0 kB of data memory
  - Cortex-M7: 9.9 kB of code and 4.0 kB of data memory
- available for ARM Cortex-M3, Cortex-M4 and Cortex-M7 architectures

### 2.2.2 MotionVC APIs

The MotionVC library APIs are:

- `uint8_t MotionVC_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 MotionVC_Initialize(void)`
  - performs MotionVC library initialization and setup of the internal mechanism

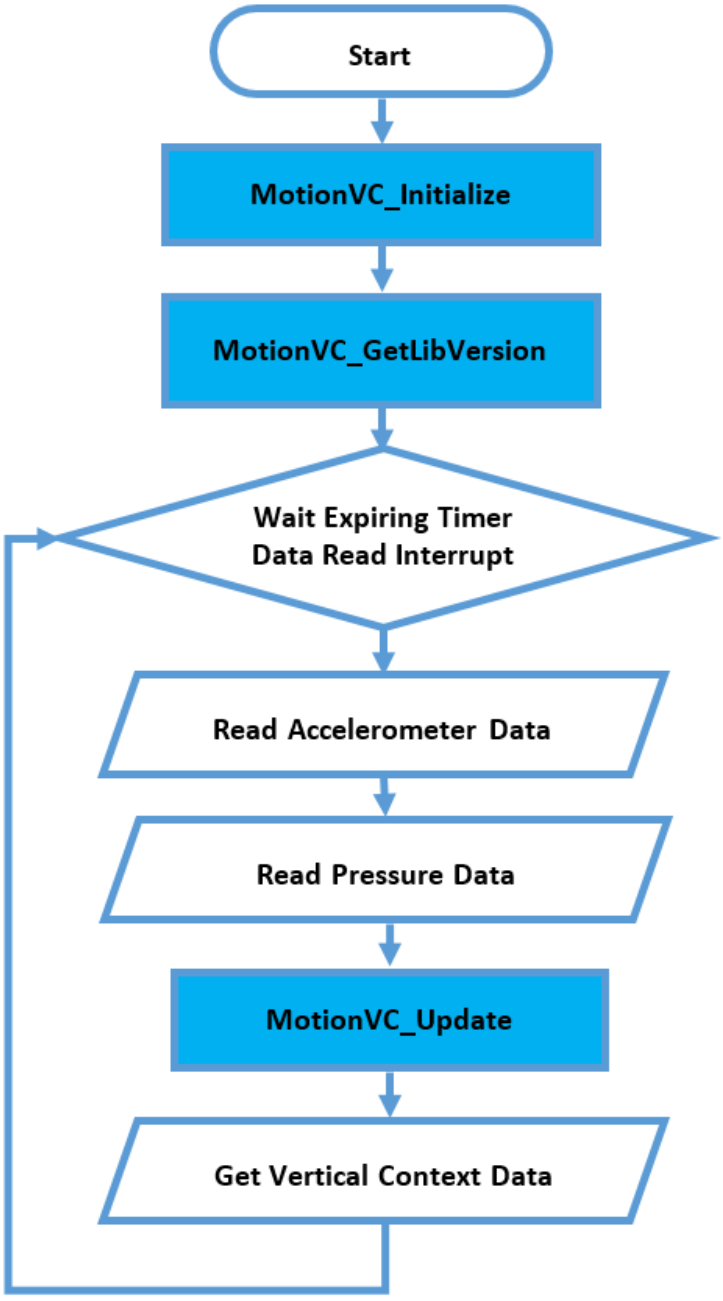
*Note: This function must be called before using the vertical context library*

- `void MotionVC_Update(MVC_input_t *data_in, MVC_output_t *data_out)`
  - runs the vertical context and altitude determination algorithm
  - retrieves the latest vertical context results
  - the parameters for the structure type `MVC_input_t` are:
    - `AccX` is the acceleration in X axis in g
    - `AccY` is the acceleration in Y axis in g
    - `AccZ` is the acceleration in Z axis in g
    - `pressValue` is the pressure sensor data in hPa
  - the parameters for the structure type `MVC_output_t` are:
    - `Timestamp` is the timestamp
    - `Valid` is the flag that indicates if the result is valid or not
    - `Baro_Altitude` is the altitude in cm computed from pressure using the standard formula
    - `Cal_Altitude` is the calibrated altitude in cm with the drift correction
    - `Speed` is the structure (`MVC_speed_t`) with information about vertical speed
    - `Context` is the vertical context (`MVC_context_t`)
    - `Confidence` is the confidence in the context (`MVC_confidence_t`)
    - `Nsteps` is the number of detected steps
  - the parameters for the structure type `MVC_speed_t` are:
    - `Speed` is the vertical speed in cm/s
    - `Speed_Error` is the estimated error of the vertical speed in cm/s
  - the items for the enum type `MVC_context_t` are:
    - `MVC_UNKNOWN` value for no pressure data or reliable data
    - `MVC_FLOOR` value for walking on flat surface
    - `MVC_UPDOWN` value for significant change observed in height
    - `MVC_STAIRS` value for stairs
    - `MVC_ELEVATOR` value for elevator
    - `MVC_ESCALATOR` value for escalator
  - the items for the enum type `MVC_confidence_t` are:
    - `MVC_CONFIDENCE_UNKNOWN`
    - `MVC_CONFIDENCE_POOR`
    - `MVC_CONFIDENCE_MED`
    - `MVC_CONFIDENCE_HIGH`

*Note: This function has to be called periodically.*

### 2.2.3 API flow chart

Figure 1. MotionVC API logic sequence




### 2.2.4 Demo code

The following demonstration code reads data from the accelerometer and pressure sensor and calculates vertical context data.

```
[…]

#define    VERSION_STR_LENG    35

/*** Initialization ***/

char lib_version[VERSION_STR_LENG];

/* Vertical Context API initialization function */ MotionVC_Initialize();
/* Optional: Get version */
MotionVC_GetLibVersion(lib_version);

[…]

/*** Using vertical context algorithm ***/

Timer_OR_DataRate_Interrupt_Handler()

{

  MVC_input_t data_in;
  MVC_output_t data_out;

  /* Get acceleration X/Y/Z in g */
  MEMS_Read_AccValue(&data_in.AccX, &data_in.AccY, &data_in.AccZ);

  /* Get pressure in hPa */
  MEMS_Read_PressValue(&data_in.Pressure);

  /* Vertical context algorithm update */
  MotionVC_Update(&data_in, &data_out);

}
```

### 2.2.5 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 | 297 | 1961 | 2 | 158 | 1152 | 2 | 411 | 3005 | 8 | 747 | 6115 | 8 | 347 | 3498 | 8 | 510 | 6385 |

**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 | 73 | 504 | 2 | 68 | 470 | 2 | 102 | 649 |

## 2.3 Sample application

The MotionVC 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.

The application detects vertical context in real-time. Data can be displayed through a GUI or stored in the board for offline analysis.

**Stand-alone mode**

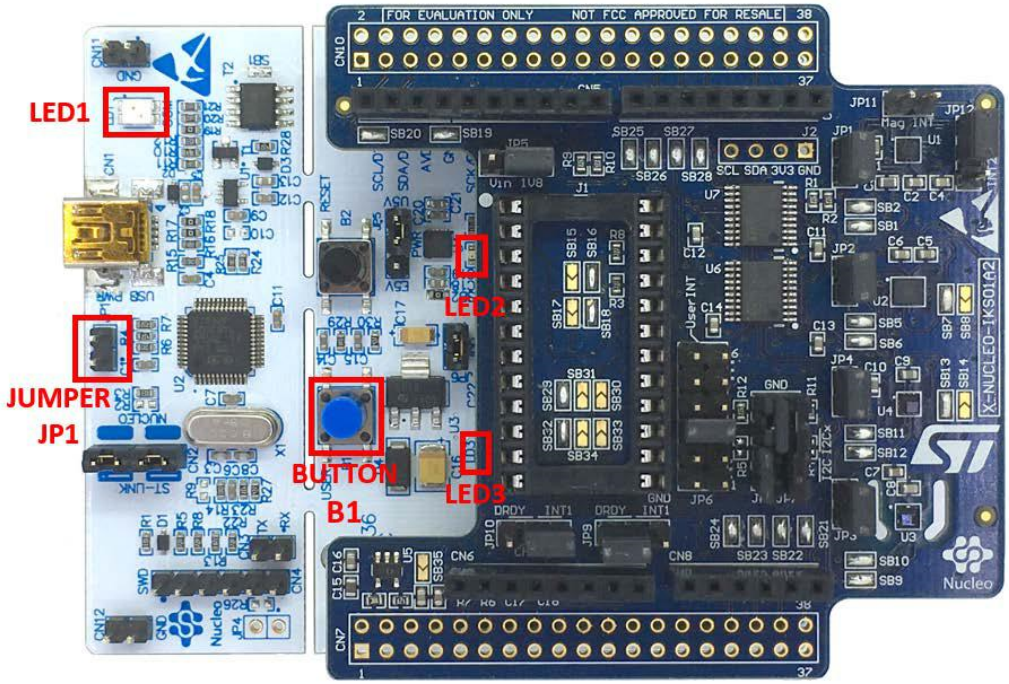
In stand-alone mode, the sample application allows the user to detect vertical context and store the information (vertical context and its confidence) in the MCU flash memory.

The STM32 Nucleo board may be supplied by a portable battery pack (to make the user experience more comfortable, portable and free of any PC connections).

**Table 4. Power supply scheme**

| Power source | JP1 settings | Working mode |
|---|---|---|
| USB PC cable | JP1 open | PC GUI driven mode |
| Battery pack | JP1 closed | Stand-alone mode |

**Figure 2. STM32 Nucleo: LEDs, button, jumper**




The above figure shows the user button B1 and the three LEDs of the NUCLEO-F401RE board. Once the board is powered, LED LD3 (power) turns ON and the tricolor LED LD1 (COM) begins blinking slowly due to the missing USB enumeration (refer to UM1724 on www.st.com for further details).

*Note: After powering the board, LED LD2 blinks once indicating the application is ready.*

When the user button B1 is pressed, the system starts acquiring data from the accelerometer and pressure sensor and detects the vertical context. During this acquisition mode, fast LED LD2 blinking indicates that the algorithm is running; the detected value is stored in the MCU internal flash memory. Data are automatically saved every 5 minutes to avoid excessive data loss in case of an unforeseen power fault.

Pressing button B1 a second time stops the algorithm and data storage, and LED LD2 switches off.

Pressing the button again starts the algorithm and data storage once again.

The Flash sector dedicated to data storage is 128 KB, allowing memorization of more than 8,000 data sets.

To retrieve these data, the board has to be connected to a PC, running Unicleo-GUI. When stored data is retrieved via the GUI, the MCU flash sector dedicated to this purpose is cleared.

If the LED LD2 is ON after powering the board, it represents a warning message indicating that the flash memory is full.

*Note: Optionally, the MCU memory can be erased by holding the user push button down for at least 5 seconds. LED LD2 switches OFF and then blinks 3 times to indicate that the data stored in the MCU have been erased. This option is available only after power ON or board reset while LED LD2 is ON indicating the flash memory is full.*

When the application runs in stand-alone mode and the flash memory is full, the application switches to PC GUI drive mode and LED LD2 switches OFF.

The flash memory must be erased by downloading data via the Unicleo-GUI or the user push button (as mentioned in the above note).

**PC GUI drive mode**

In this mode, a USB cable connection is required to monitor real-time data. The board is powered by the PC via USB connection. This working mode allows the user to display the activity detected, accelerometer data, time stamp and eventually other sensor data, in real-time, using the Unicleo-GUI.

In this working mode, data are not stored in the MCU flash memory.

### 2.3.1 Unicleo-GUI application

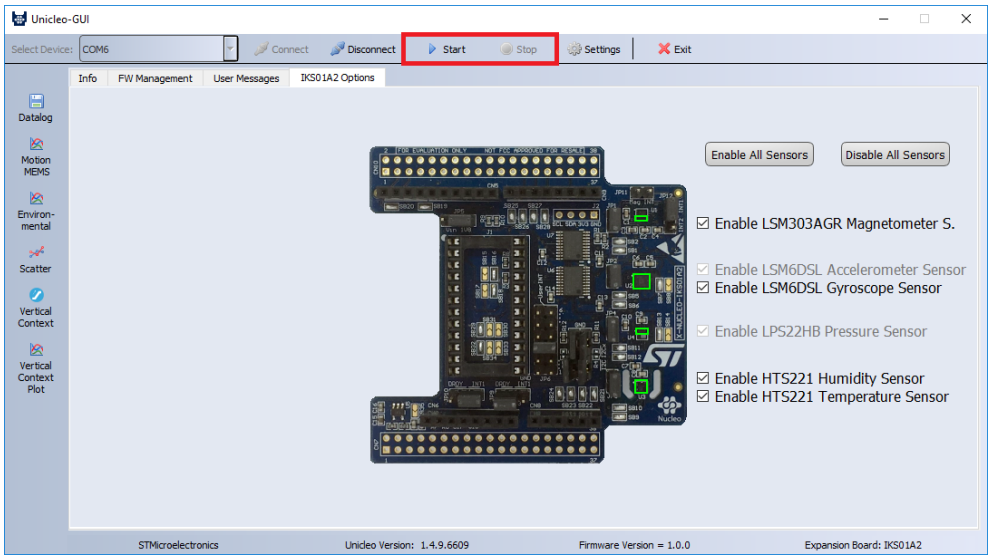
The sample application uses the Windows Unicleo-GUI utility, which can be downloaded from www.st.com.

**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.

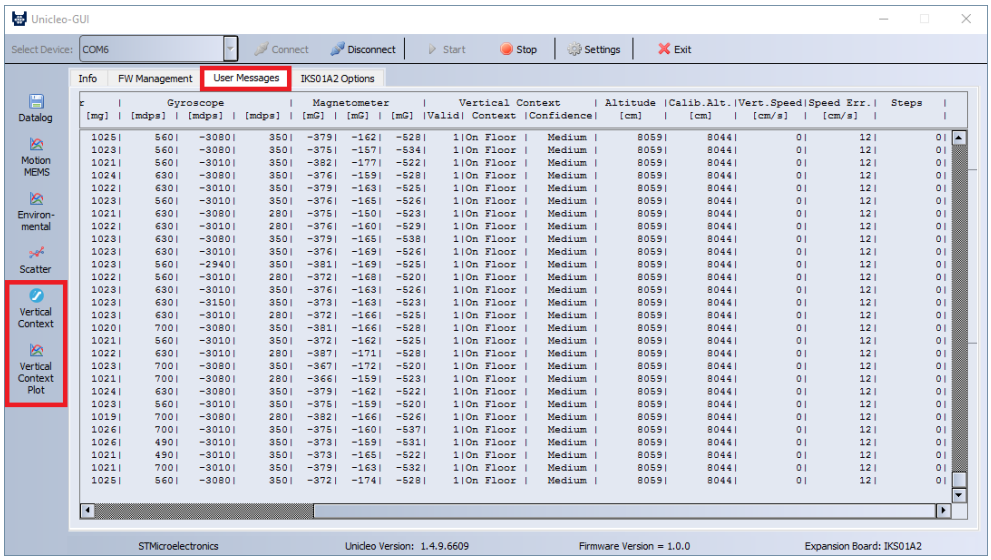
If an STM32 Nucleo board with supported firmware is connected to the PC, it is automatically detected and the appropriate COM port is opened.

**Figure 3. Unicleo main window**



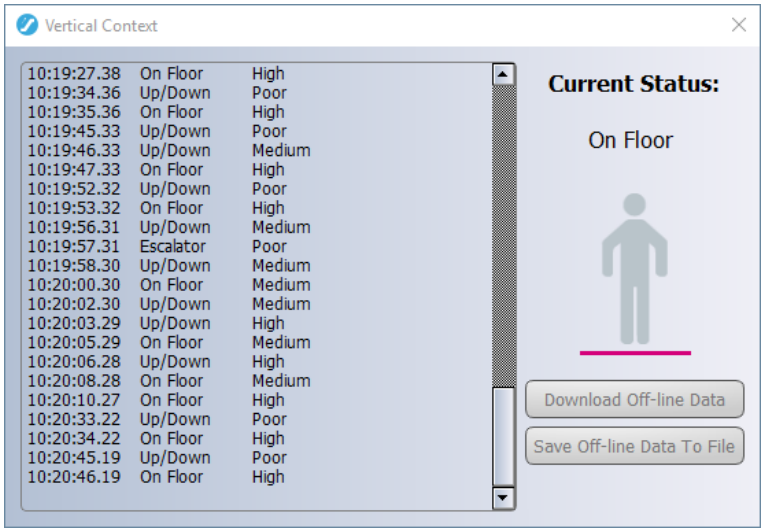
**Step 3.** Start and stop data streaming by using the appropriate buttons on the vertical tool bar. The data coming from the connected sensor can be viewed in the User Messages tab.

**Figure 4. User Messages tab**




**Step 4.** Click on the Vertical Context icon in the vertical toolbar to open the dedicated application window, where you can see detected vertical context and its confidence.

**Figure 5. Vertical Context window**




If the board has been working in stand-alone mode and the user wants to retrieve stored data, press **Download Off-line Data** button to upload the stored vertical context data to the application. This operation automatically deletes acquired data from microcontroller.

*Note:* ***Download Off-line Data*** *button is not available while data streaming is active.*

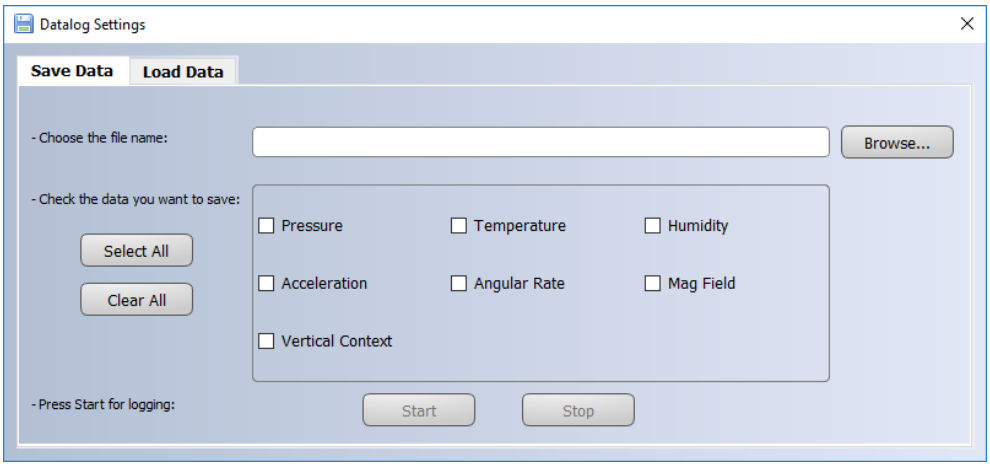
Press the **Save Off-line Data to File** button to save the uploaded data in a .tsv file.

**Step 5.** Click on the Vertical Context Plot icon in the vertical toolbar to open dedicated widows, where you can see all the outputs from the vertical context algorithm. The window is split into three section: the first one is the graph of standard and calibrated altitude, the second one is graph of vertical speed and the third section contains all the outputs from the algorithm (data valid flag, vertical context, context confidence, altitude, calibrated altitude, vertical speed, vertical speed error, number of steps) in text form.

**Figure 6. Vertical Context Plot window**

**Step 6.** Click on the Datalog icon in the vertical toolbar to open the datalog configuration window: you can select the sensor and vertical context data to be saved in the files. You can start or stop saving by clicking on the corresponding button.

**Figure 7. Datalog window**




## 2.4 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 5. **Document revision history**

| Date | Version | Changes |
|---|---|---|
| 17-Sep-2018 | 1 | Initial release. |
| 15-Feb-2019 | 2 | Updated Table 2. Elapsed time (µs) algorithm.<br>Added X-NUCLEO-IKS01A3 expansion board compatibility informaton. |
| 26-Mar-2020 | 3 | Updated Introduction, Section 2.2.1 MotionVC library description and Section 2.2.5 Algorithm performance. |

# Contents

# List of tables

# List of figures