

Getting Started with STM32CubeIDE and SensorTile

COMPSYS 704 Advanced Embedded Systems

Akshat Bisht and Kevin Wang

This lab tutorial aims to guide you through the process of installing the programming environment for the SensorTile embedded device and building example projects which will demonstrate the use of the XPRINTF function (to display data over USB on your laptop) and the provided Bluetooth interface (to send data over Bluetooth on a mobile app). While printing on laptop is not an essential part of the project, it can be useful for debugging and checking sensor data validity. To begin this lab, download the example project provided on Canvas. This project already contains the necessary abstraction layer you need for your project, so use this project as your starting point for this lab exercise and for the project.

Lab preparation

Before attempting this lab, you should install the following software and app:

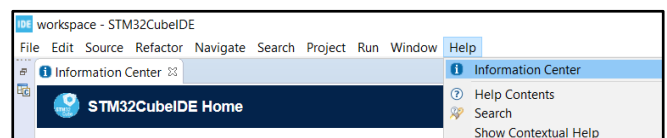
1. STM32CubeIDE: The software IDE for STM development kit, which can be downloaded from: <https://www.st.com/en/development-tools/stm32cubeide.html>
STM32CubeIDE is already installed in the MDLS lab machines. For your convenience, we encourage you to try install it in your own laptop.
2. CoolTerm: The terminal program for displaying the output from the XPRINTF function over USB. This can be downloaded from <http://freeware.the-meiers.org/>. This can be replaced by other terminal program, such as PuTTY.
3. ST BLE Sensor app: This is a mobile app that you can use to display data sent over the Bluetooth interface.
Android: <https://play.google.com/store/apps/details?id=com.st.bluems>
iOS: <https://apps.apple.com/it/app/st-bluems/id993670214>

Connecting the ST-LINK/V2 programmer with SensorTile

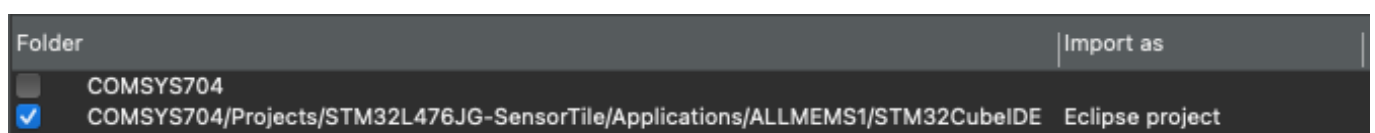
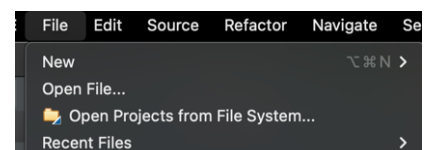
In the kit provided to you, there is a programmer board (the bigger) and the wearable SensorTile device. The SWD connector is for the ST-LINK/V2 programmer where the 5-pin ribbon cable can be used to connect the programmer with the SensorTile device. The first pin of the ribbon cable (with an arrow on the header) should be connected to the first pin of the SWD connector (which usually has a dot next to it. The same applies to both the programmer board and the SensorTile device). Two cables are provided in your project package. The USB mini cable is used to power the discovery board, whereas the USB micro cable is used to power/charge the SensorTile device. The following XPRINTF example is also using this USB interface. The SensorTile is equipped with a 100mAh Li-Po battery and can operate without any power supply for a short period of time.

Implement, Build, and Run a XPRINTF example

Once STM32CubeIDE is installed, open the IDE and create your own workspace directory. Once the workspace has been loaded, you should arrive at the **Information Centre** page, which is not needed and can be closed.



Download the provided template project (i.e. COMPSYS704.zip) from Canvas and unzip it to the workspace directory. You can open the project in CubeIDE by choosing “**Open Projects from File System**” under the **File** menu. When importing the project, make sure you only tick the following project:



Once the project is loaded, under the project navigation pane, go to Application -> User -> main.c, double click to open the file. In the main() function, the following functions are already provided to initialise the SensorTile device for you. You can go through them if you are interested, but you don't need to change anything in those functions.

```
HAL_Init();

// Configure the System clock
SystemClock_Config();

InitTargetPlatform();

// Initialize the BlueNRG
Init_BlueNRG_Stack();

// Initialize the BlueNRG Custom services
Init_BlueNRG_Custom_Services();

// Initialize timers
InitTimers();
```

In the main() function, the **ReadSensor** flag is set every 100ms. In this exercise, the goal is to show how to use the XPRINTF function, so you can print a simple periodically incremented number example as follows:

```
if (ReadSensor) {
    ReadSensor=0; // reset flag

    //*****get sensor data*****
    readMag();
    readAcc();

    //*****process sensor data*****

    COMP_Value.x++;
    COMP_Value.y++;
    COMP_Value Heading++;

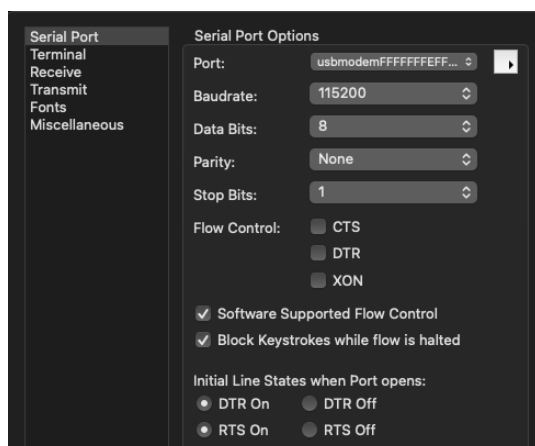
    XPRINTF("***STEP INCREMENTS = %d**\r\n", (int)COMP_Value.x);
}
```

The XPRINTF function works through the USB-CDC interface, which can be displayed through a terminal program.

Once complete, **build** , and then **run**  the project by selecting the relevant button from the toolbar.

In this exercise, we are using CoolTerm as the terminal program. As mentioned earlier, you can use any terminal program that you are familiar with, with the following settings. Once the terminal program is connected, the XPRINTF statements will be displayed.

On Windows the COM Port will appear as COMX, where X can be any number, but not 1, as COM1 is a hardware port on the computer itself. On macOS the UART port will appear with the name usbmodemFFFFFFFFFFFF, or something similar. The Baud rate can be 9600 or 115200. Data bits 8, 1 Stop bit and no parity bits.





Implement, Build, and Run a Bluetooth communication example

In this example, you will modify the project to display data values on the ST BLE Sensor App via the provided Bluetooth communication. The following global variables are defined to store the raw Accelerometer, Magnetometer sensor data and the processed data **COMP_Value** variables are defined to store the detected number of Steps, Heading angle, and Distance. These global variables (**ACC_Value**, **MAG_Value**, and **COMP_Value**) will be automatically sent over Bluetooth every 100ms by setting the **SendAccGyroMag** flag in the infinite event while loop.

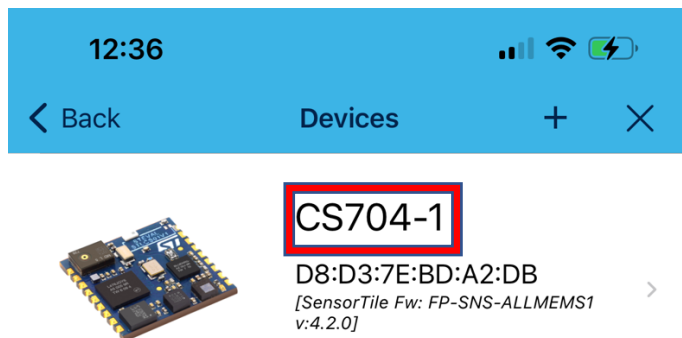
ACC_Value.x, ACC_Value.y, ACC_Value.z &

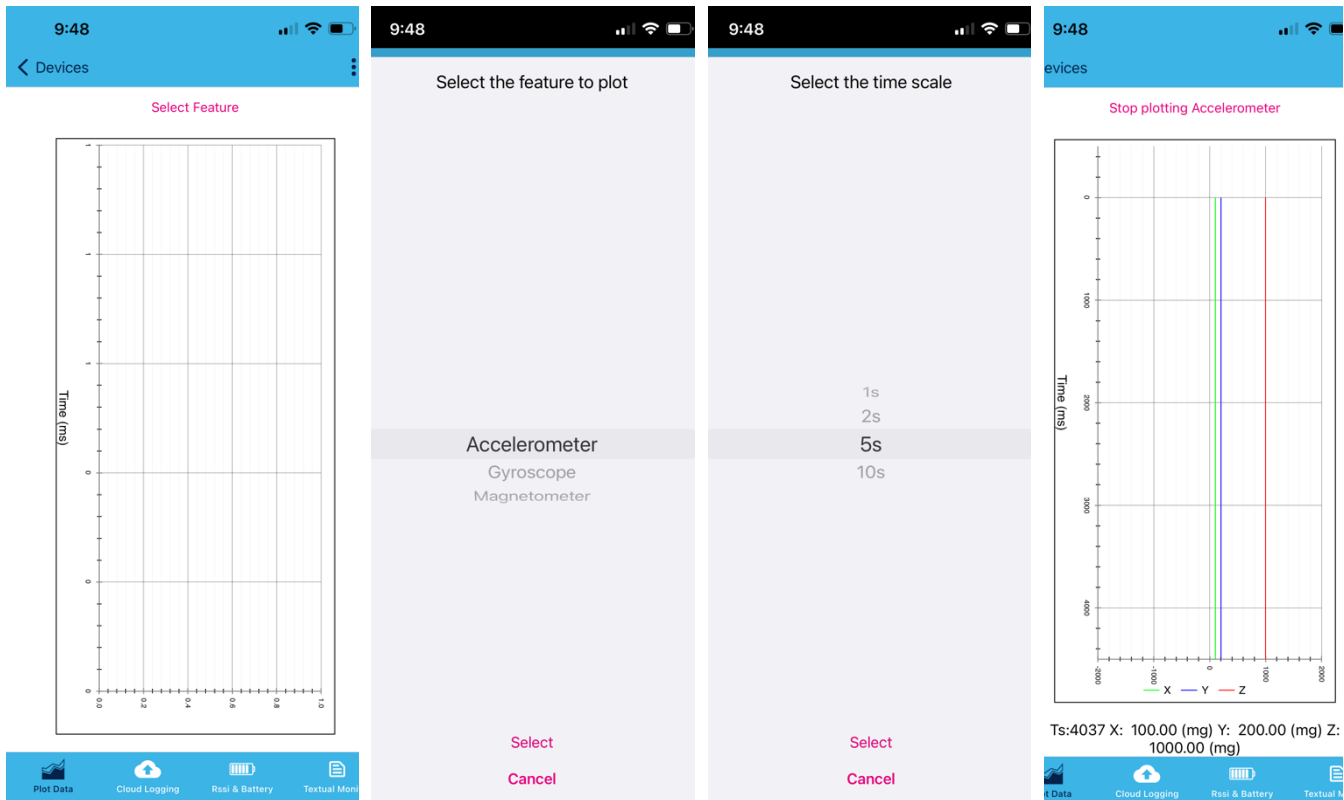
MAG_Value.x, MAG_Value.y, MAG_Value.z &

COMP_Value.x, COMP_Value.y, COMP_Value.Heading

In addition to the raw sensor and processed data value, the device name can be changed through the **NodeName[8]** variable. To emulate the raw sensor and processed data, you can simply try incrementing the **ACC_Value**, **MAG_Value**, and **COMP_Value** variables. once the code is complete, **build** , and then **run**  the project by selecting the relevant button from the toolbar.

Once you started running the project, open the ST BLE Sensor app and select “Connect to a device”. There should be a device with the same name defined by the **NodeName** variable showing up as in the figure below. Click on the device to connect to it. The data received by the mobile BLE app will be plotted separately by selecting a **feature and time scale** (i.e. Accelerometer (ACC_Value), Magnetometer (MAG_Value), or Gyroscope (COMP_Value)) respectively as shown in the figures next page.





Once you are done with the lab, it is time for you to dig in into the datasheet to understand how to configure the LSM303AGR (the integrated accelerometer and magnetometer sensor). The SPI bus interface functions are provided to you. You will need to initialise the sensor and apply the necessary compensation/calibration (if any), and figure out how to interpret the sensor data such that it can be used to implement your pedestrian dead reckoning algorithm (with step counting and heading angle detection).

Implement the following functions to initialise and read data from the Magnetometer and Accelerometer.

```
static void startMag() {
    //CS704 - Write SPI commands to initiliasie Magnetometer
}

static void startAcc() {
    //CS704 - Write SPI commands to initiliasie Accelerometer
}

static void readMag() {
    //CS704 - Read Magnetometer Data over SPI

    //CS704 - store sensor values into the variables below
    MAG_Value.x=100;
    MAG_Value.y=200;
    MAG_Value.z=1000;
}

static void readAcc() {
    //CS704 - Read Accelerometer Data over SPI

    //CS704 - store sensor values into the variables below
    ACC_Value.x=100;
    ACC_Value.y=200;
    ACC_Value.z=1000;
}
```

SPI Functions:

There are the following SPI Read and Write functions for the accelerometer and magnetometer.

- BSP_LSM303AGR_WriteReg_Mag()
- BSP_LSM303AGR_ReadReg_Mag()
- BSP_LSM303AGR_WriteReg_Acc()

- `BSP_LSM303AGR_ReadReg_Acc()`

Example usage of these functions is shown in [InitLSM\(\)](#)

More details about the various sections of `main.c` code are explained in the comment block at the beginning of `main.c`.