# STMicroelectronics SensorTile Tutorial: IoT Machine Learning Motion Pattern Classification

## Table of Contents

# 1. Introduction to This Tutorial

This Tutorial describes the development of a SensorTIle *self-learning* system that recognizes a motion pattern consisting of a sequence of motions. As for the previous SensorTile Tutorials, this applies the EmbeddedML system developed by Charles Zaloom.

In the previous Tutorial: STMicroelectronics SensorTile Tutorial: IoT Machine Learning Motion Classification, the objective was the development of a system that detected changes in orientation of the SensorTile.

The new objective for this Tutorial is the classification of a motion *pattern* that includes a sequence of motions.

The requirement now is to develop *features* that are derived from sensor signals and enable the differing motion patterns to be distinguished.

This Tutorial will include acquisition of data from motion sensors for both training of a neural

For more information regarding the SensorTile board, please open the following link.
www.st.com/sensortile

## *List of Required Equipment and Materials*

1) 1x STMicroelectronics SensorTile kit.
2) 1x STMicroelectronics Nucleo Board.
3) 1x Personal Computer with two USB type-A inputs OR you must have a powered USB hub.
4) 1x USB 2.0 A-Male to Micro-B Cable (micro USB cable).
5) 1x USB 2.0 A-Male to Mini-B Cable (mini USB cable).
6) Network access to the Internet.

## *Prerequisite Tutorials*

It is important to have completed Tutorials 1 through Tutorial 4 and Tutorial 10 and 11.

Your instructor will ensure you have the required background regarding motion sensing, digital signal processing, and basic physics of acceleration, velocity, and displacement.

## 2. Neural Network Motion Pattern Classification Objective

In the previous Tutorial, the objective for the machine learning system was to develop a machine learning system that could be trained to detect changes in the orientation of the SensorTile system.

There were six instances of motion. For Motion Classification, the computation of features for each instance was based on the processing of sensor signals that occur during the period of each instance.

This development is directed to a system that includes two motion states forming one pattern. The first motion state corresponds to a change in orientation, similar to that of the previous development. However, there will be a second motion state occurring in each instance. Specifically, in an instance, the SensorTile will start from an initial orientation and then move to a new orientation. Then, as a second step, the SensorTile will be returned to a level position, or rotated to an orientation where its Z-Axis is level.

## 3. Feature Data Acquisition for Motion Pattern Classification

The feature for this application will include sensor signals from the two motion steps:

State 0)

In the first state, State 0, this system will first measure the difference in acceleration between an initial orientation where the user holds the SensorTile in a starting position, and a final state where the user rotates the SensorTile and holds it in a final position.

As in the previous Tutorial, the initial state will occur with the SensorTile held level (or nearly level) and the final state will occur with the SensorTile tilted about a direction by an angle of about 45 degrees, for example.

Now, the features required for determining orientation include only the X and Y axis accelerations. Here, the projection of the gravitational acceleration vector on the sensor X and Y axis accelerometer axes provides information on the angles defining the orientation of the SensorTile.

The classification features will be derived then from these accelerations.

The features, $F_1$ and $F_2$, are the differences between the X and Y axis accelerations, respectively, between a final acceleration value when motion is complete and an initial value.

$$a_x = a_{final-x} - a_{initial-x}$$
$$a_y = a_{final-y} - a_{initial-y}$$
$$a_z = 0$$

Since no information is required from Z-Axis acceleration, this is set to zero.

State 1)

In the second state, State 1, the system will now acquire sensor signal data during a second motion.  In State 1, the user will either orient the SensorTile level (where its accelerometer Z-Axis is vertical) shown in Figure 1(a), or orient the SensorTile with its Z-Axis horizontal. This is accomplished by rotating the SensorTile to an orientation where its printed circuit board surface is oriented vertical shown in Figure 1(b).

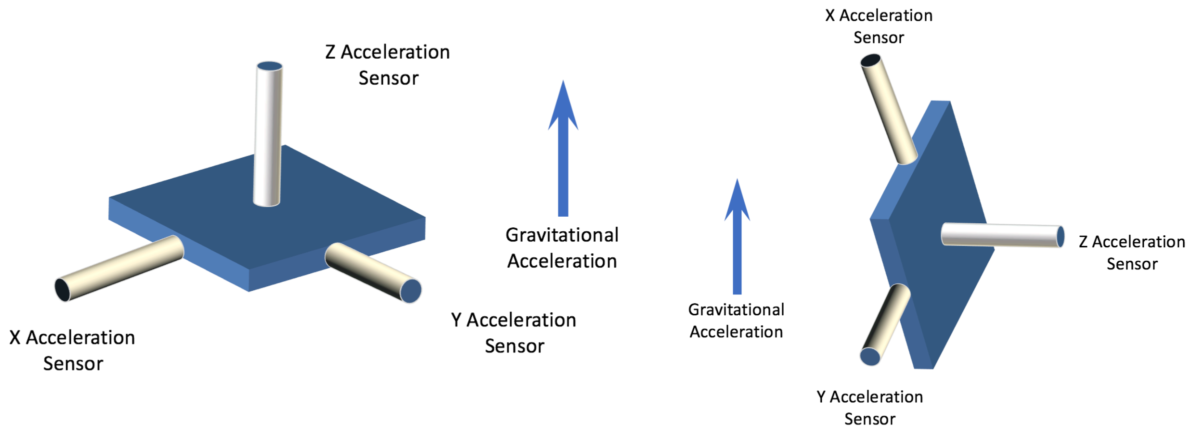


*Figure 1. SensorTile in Level orientation shown in (a) and rotated to a vertical orientation with Z-axis level shown in (b).*

## 4. Design of New Feature for Motion Pattern Classification

The objective of Motion Pattern Classification is to accurately classify motions that include two orientation change events.

New features must be designed that enable both classification of an orientation change in State 0 and the presence or absence of an orientation change in State 1. Consider these new features:

$$F_1 = \frac{a_x}{\sqrt{{a_x}^2 + {a_y}^2}}$$

$$F_2 = \frac{a_y}{\sqrt{{a_x}^2 + {a_y}^2}}$$

If Z-Axis acceleration in State 1 is less than Z_ACCEL_THRESHOLD:

$$F_3 = 0$$

If Z-Axis acceleration in State 1 is greater than Z_ACCEL_THRESHOLD:

$$F_3 = \frac{(F_1 + F_2)}{2}$$

## 5. Feature Computation for Motion Pattern Classification

The same Neural Network as included in Tutorial 10 is applied here. This has a topology of 3 input neurons, 9 hidden layer neurons, and 6 output neurons. Therefore, three features are required to be supplied to the three input neurons.

The features must now be normalized before being presented to the Neural Network classifier.

This normalization will ensure that all values corresponding to the X and Y axis acceleration values lie within the range of -1.0 to +1.0. Normalized feature values, $F_i$ , will be derived from acceleration values, $a_i$ , for $i = x \; or \; y$ with:

$$F_1 = \frac{a_x}{\sqrt{{a_x}^2 + {a_y}^2}}$$

$$F_2 = \frac{a_y}{\sqrt{{a_x}^2 + {a_y}^2}}$$

$$F_3 = \frac{(F_1 + F_2)}{2}$$

## 6. Acquisition of Training Data Features

The Neural Network training process is shown in Figure 2. This starts with a notification to the user (via both an LED illumination and a text message) to orient the SensorTile in its initial state and then in its final state. The process then continues with a measurement of feature values that correspond to training Ground Truth.

There will be six instances of orientation and therefore six sets of Ground Truth values.

After acquisition of feature values, these are normalized by the Softmax function. The process continues until all training is complete for each of the six instances.

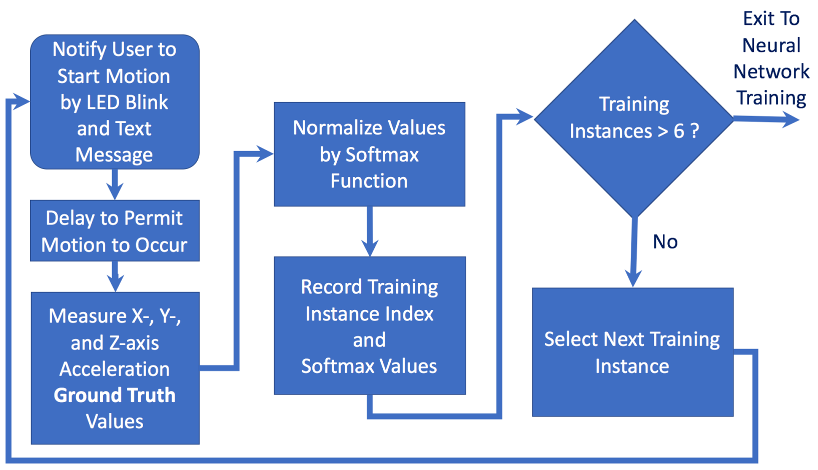

Notify User to Start Motion by LED Blink and Text Message
Delay to Permit Motion to Occur
Measure X-, Y-, and Z-axis Acceleration **Ground Truth** Values
Normalize Values by Softmax Function
Record Training Instance Index and Softmax Values
Training Instances > 6 ?
Exit To Neural Network Training
No
Select Next Training Instance

*Figure 2. The data acquisition and feature computation process. This system executes for 6 training instances and then exits to the Neural Network training process.*

## 7. Neural Network Training for EmbeddedML

After completion of acquisition and normalization of features, the Neural Network training may start. The EmbeddedML system follows the process of Figure 3.

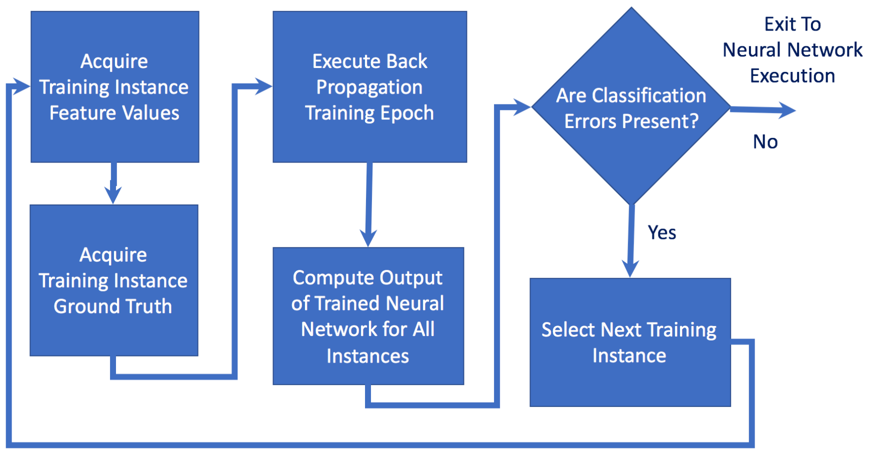

Acquire Training Instance Feature Values
Acquire Training Instance Ground Truth
Execute Back Propagation Training Epoch
Compute Output of Trained Neural Network for All Instances
Are Classification Errors Present?
Exit To Neural Network Execution
No
Yes
Select Next Training Instance

*Figure 3. The Neural Network training process executing with the features and Ground Truth obtained from the data acquisition process of Figure 2.*

## 8. Neural Network Execution

Finally, Neural Network execution proceeds. Here, as in the acquisition of training data, the user is notified to complete an orientation action. Then, feature values are computed by normalization. These are then presented to the Neural Network execution system.

The Neural Network execution system proceeds by forward propagation computation to determine output classifications based on input values and the trained Neural Network.

The user is notified of the predicted result by both a text message and an LED blink pattern.

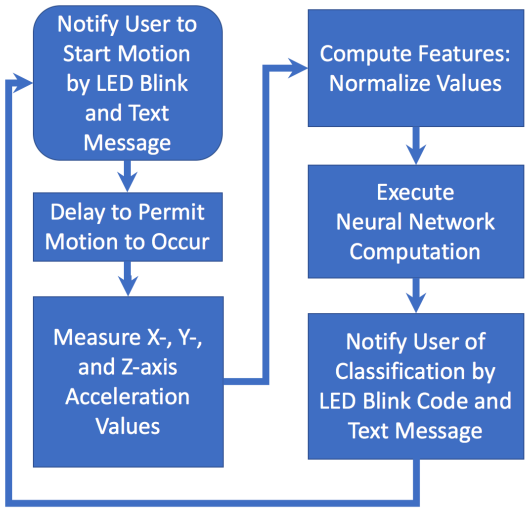

Notify User to Start Motion by LED Blink and Text Message
Delay to Permit Motion to Occur
Measure X-, Y-, and Z-axis Acceleration Values
Compute Features: Normalize Values
Execute Neural Network Computation
Notify User of Classification by LED Blink Code and Text Message

*Figure 4. The Neural Network execution process operating on data acquired and with the Neural Network trained by the process of Figure 3.*

## 9. EmbeddedML Motion Classification Software Systems

It is important to review the application software components now.

This system will include the primary data acquisition systems of Tutorial 11. This previous system provided data sampling with the Feature_Extraction_State_0 () function.

First, acceleration values are returned as arguments of the getAccel function. It takes an argument corresponding to a pointer address to the accelerometer system handler data structure.

```
void getAccel(void *handle, int *xyz) {
        uint8_t id;
        SensorAxes_t acceleration;
        uint8_t status;

        BSP_ACCELERO_Get_Instance(handle, &id);
        BSP_ACCELERO_IsInitialized(handle, &status);

        if (status == 1) {
                if (BSP_ACCELERO_Get_Axes(handle, &acceleration) == COMPONENT_ERROR) {
                        acceleration.AXIS_X = 0;
                        acceleration.AXIS_Y = 0;
                        acceleration.AXIS_Z = 0;
                }
                xyz[0] = (int) acceleration.AXIS_X;
                xyz[1] = (int) acceleration.AXIS_Y;
                xyz[2] = (int) acceleration.AXIS_Z;
        }
}
```

Second, the notification to the user and acquisition of features is supplied by the Feature_Extraction_State_0 () function.

```
void Feature_Extraction_State_0(void *handle, int * ttt_1, int * ttt_2,
                int * ttt_3, int * ttt_mag_scale) {

        int ttt[3];
        int ttt_initial[3];
        int axis_index;
        float accel_mag;
        char  msg[128];

        /*
        * Acquire acceleration values prior to motion
        */
```

```
        getAccel(handle, ttt_initial);

        sprintf(msg, "\r\nStart Motion to new Orientation and hold while LED On");
        CDC_Fill_Buffer((uint8_t *) msg, strlen(msg));
        BSP_LED_On(LED1);
        HAL_Delay(3000);

        /*
        * Acquire final state acceleration values after motion
        * Notify user of motion complete
        */
        getAccel(handle, ttt);
        sprintf(msg, "\r\n Motion Complete");
        CDC_Fill_Buffer((uint8_t *) msg, strlen(msg));
        BSP_LED_Off(LED1);

        /*
        * Compute acceleration magnitude including only X and Y axes
        */

        accel_mag = 0;
                for (axis_index = 0; axis_index < 3; axis_index++) {
                   accel_mag = accel_mag + pow((ttt[axis_index] - ttt_initial[axis_index]), 2);
                }
        accel_mag = sqrt(accel_mag);

        /*
        * Compute feature values including only X and Y axes and
        * setting Z-axis feature to zero
        */
        *ttt_1 = ttt[0] - ttt_initial[0];
        *ttt_2 = ttt[1] - ttt_initial[1];
        *ttt_3 = ttt[2] - ttt_initial[2];
        *ttt_mag_scale = (int)(accel_mag);
        return;
}
```

Please refer to Figure 2 to while reviewing this function.

However, as an extension to the system of Tutorial 11, this Motion Pattern Classification system adds a new feature.

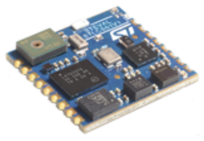
This is provided in the Feature_Extraction_State_1 () function.  Note this highlighted below.

Then, you may examine the TrainOrientation() function.  An excerpt here is shown below.

```
                sprintf(msg1, "\r\n\r\n\r\nTraining Start in 2 seconds ..");
                CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
                BSP_LED_Off(LED1);
                HAL_Delay(2000);

                num_train_data_cycles = 1;

                for (k = 0; k < num_train_data_cycles; k++) {
                        for (i = 0; i < 6; i++) {
```



```
                                        sprintf(msg1, "\r\nMove to Start Position - Wait for LED On");
                                        CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
                                        HAL_Delay(START_POSITION_INTERVAL);

                                        switch (i) {
                                        HAL_Delay(1000);

                                case 0:
                                        sprintf(msg1, "\r\nMove to Orientation 1 on LED On");
                                        CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));

                                        Feature_Extraction_State_0(handle, &ttt_1, &ttt_2, &ttt_3,
                                                                        &ttt_mag_scale);


                                        Feature_Extraction_State_1(handle, &ttt_1, &ttt_2, &ttt_3,
                                                                        &ttt_mag_scale);

                                        ttt_initial_max[0] = ttt_1;
                                        ttt_initial_max[1] = ttt_2;
                                        ttt_initial_max[2] = ttt_3;

                                        break;
```

Feature_Extraction_State_0() will return the values of acceleration that are derived from the first orientation change state. Then, Feature_Extraction_State_1() will measure the change in Z-axis acceleration and compare this with a threshold, Z_ACCEL_THRESHOLD.

This will then construct the third feature as described in Section 5.

The Feature_Extraction_State_1() function arguments include the values of acceleration that were acquired by Feature_Extraction_State_0(). These are also returned by Feature_Extraction_State_1(). In addition, Feature_Extraction_State_1() returns the value of the third feature, ttt_3. The function is shown below.

Please note the highlighted code sequence that produces this third feature.

```
void Feature_Extraction_State_1(void *handle, int * ttt_1, int * ttt_2,
int * ttt_3, int * ttt_mag_scale) {
int ttt[3];
char msg1[128];

/*
 * Notify user to initiate second motion to rotate vertical up or down
 */

LED_Notification_Blink(12);

sprintf(msg1, "\r\nNow Rotate to New Angle when LED On");
CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
BSP_LED_On(LED1);
HAL_Delay(3000);
```

```
/*
 * Retain initial values of acceleration for X and Y axes
 */

/*
 * Detect SensorTile in inverted position
 *
 * Compute new feature according to value of Z-axis acceleration
 * and orientation
 *
 * X and Y axis accelerations remain unchanged.  However, Z axis
 * acceleration is assigned either to the average of X and Y
 * axis acceleration or to zero.
 */

getAccel(handle, ttt);

if (abs(ttt[2]) > Z_ACCEL_THRESHOLD){
       *ttt_3 = (*ttt_1 + *ttt_2)/2;}
else {
       *ttt_3 = 0;
}

BSP_LED_Off(LED1);

sprintf(msg1, "\r\nMotion Complete, Now Return to Next Start Position");
CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
HAL_Delay(2000);
return;

}
```

Observe that there will be 6 training instances (enumerated 0 through 5). For each instance, the Feature_Extraction_State_0() function acts to measure Ground Truth.

After each instance is complete, normalization occurs and a training dataset is computed.

```
XYZ[0] = (float) ttt_1;
XYZ[1] = (float) ttt_2;
XYZ[2] = (float) ttt_3;

motion_softmax(net->topology[0], XYZ, xyz);

training_dataset[i][k][0] = xyz[0];
training_dataset[i][k][1] = xyz[1];
training_dataset[i][k][2] = xyz[2];
```

The normalization is computed by motion_softmax, below, as described in the previous Tutorial 11.

```
void motion_softmax(int size, float *x, float *y) {
	float norm;
	norm = sqrt((x[0] * x[0]) + (x[1] * x[1]) + (x[2] * x[2]));
	y[0] = x[0] / norm;
	y[1] = x[1] / norm;
	y[2] = x[2] / norm;
}
```

The Neural Network is trained with its Neural Network defined by the data structure, net, training Ground Truth data, and an indicator of the specific Orientation instance corresponding to Ground Truth, _Motion_N, where N ranges from 1 to 6.

Finally, the Neural Network, described by the net data structure, is tested for accuracy by executing the Neural Network with test data:

```
run_ann(net, training_dataset[m][k]);
```

Training occurs over all instances for a maximum of 2000 epochs. The results of training are presented to the user at each of 100 epochs.

## 10. EmbeddedML Motion Classification Execution

After training is complete, the EmbeddedML system is now capable of classification of unknown inputs and predicting actual orientation.

This occurs in the Accel_Sensor_Handler() function. This function takes the arguments of a pointer address to the handler data structure associated with accelerometer measurement. This also includes the pointer address to the net data structure describing the complete Neural Network. This also returns a value, prev_loc, that corresponds to the actual prediction.

Focusing on the code segment that enables execution, the first step, shown below, is the call to the `Feature_Extraction_State_0()` function returning data from one orientation change. Next, the `Feature_Extraction_State_1()` function executes returning data from the second motion and second orientation change.

This is followed by motion_softmax() normalization.

Finally, run_ann() executes forward propagation computation based on the newly trained Neural Network.

Finally, the identification of the most likely prediction of the Neural Network is made. Specifically, the six output neuron values appear as net->output[i] where i ranges from 0 to 5 including the six values. The greatest of these values corresponds to the prediction of the correct classification among the six possibilities.

Given a value for this prediction, loc, the user notification may occur including blinking of the LED with an identifying code and printing of this value.

Please also note that this execution cycle will occur 10 times, permitting testing, after which the system will enter retraining.

```
Feature_Extraction_State_0(handle, &ttt_1, &ttt_2, &ttt_3,
            &ttt_mag_scale);


Feature_Extraction_State_1(handle, &ttt_1, &ttt_2, &ttt_3,
                        &ttt_mag_scale);


XYZ[0] = (float) ttt_1;
XYZ[1] = (float) ttt_2;
XYZ[2] = (float) ttt_3;

motion_softmax(net->topology[0], XYZ, xyz);

run_ann(net, xyz);


for (i = 0; i < net->topology[net->n_layers - 1]; i++) {
      if (net->output[i] > point && net->output[i] > 0.1) {
            point = net->output[i];
            loc = i;
      }
}
```

# 11. EmbeddedML Motion Classification Project

## *SensorTile EmbeddedML System Installation*

1. **Download the project from google drive**

   [https://drive.google.com/open?id=1zr5Oy8jKoSc3VQlLrm7yiWLkCRUfZB5A](https://drive.google.com/open?id=1zr5Oy8jKoSc3VQlLrm7yiWLkCRUfZB5A)

and decompress it to your working directory.

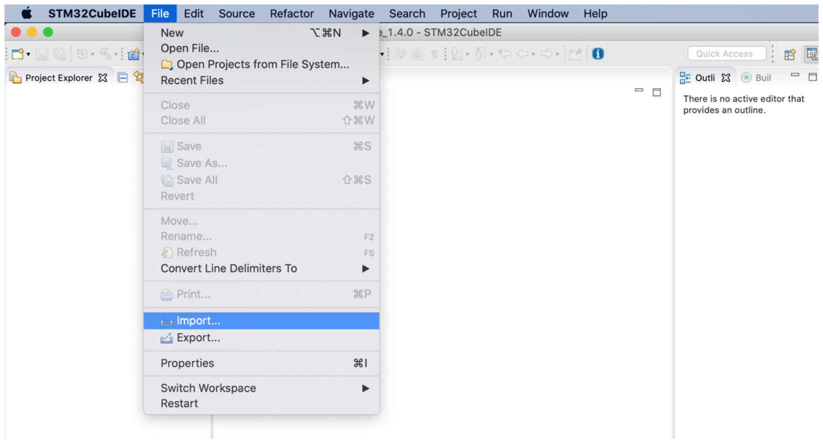
***Note: Please download this project above. Please do not use the Projects from previous Tutorials.***

2. Open the STM32CubeIDE, as instructed in the Tutorial 1**.** Select the same workspace as in Tutorial 1.

3. Once the IDE is open, first remove your current project.

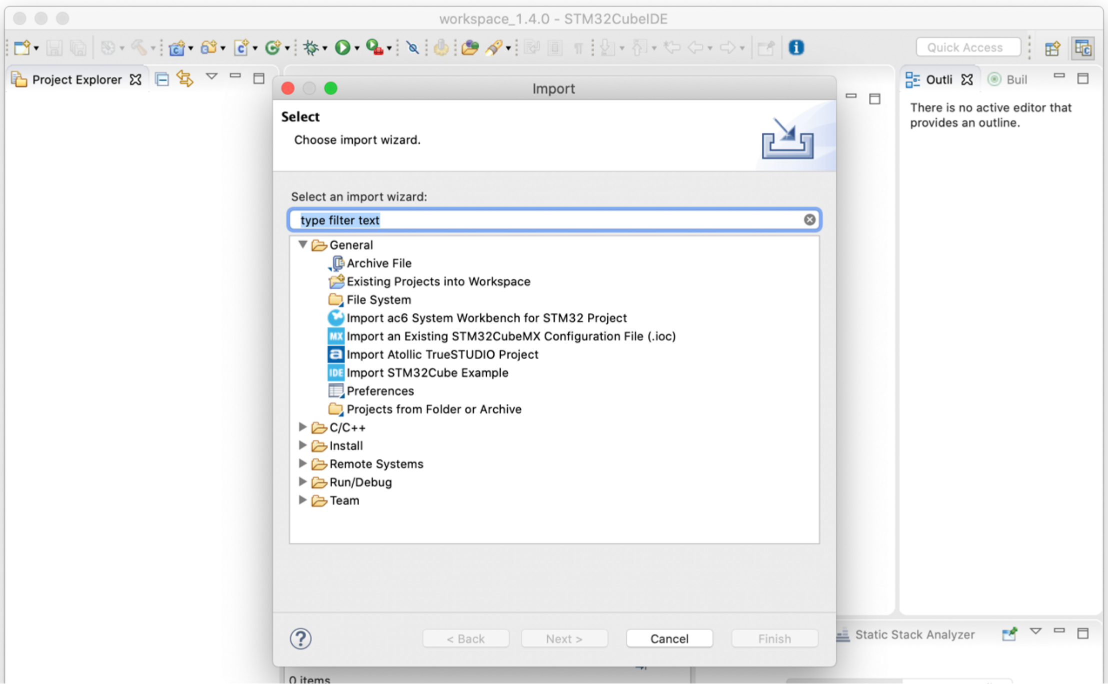
4. Then, import the new Tutorial 12

5. Now, start the STM32CubeIDE Application. Select File > Import



There is no active editor that provides an outline.

6. Then, Select General



workspace_1.4.0 - STM32CubeIDE

Project Explorer

Import

**Select**

Choose import wizard.

Select an import wizard:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Import ac6 System Workbench for STM32 Project
  - Import an Existing STM32CubeMX Configuration File (.ioc)
  - Import Atollic TrueSTUDIO Project
  - Import STM32Cube Example
  - Preferences
  - Projects from Folder or Archive
- C/C++
- Install
- Remote Systems
- Run/Debug
- Team

< Back | Next > | Cancel | Finish

Outli | Buil

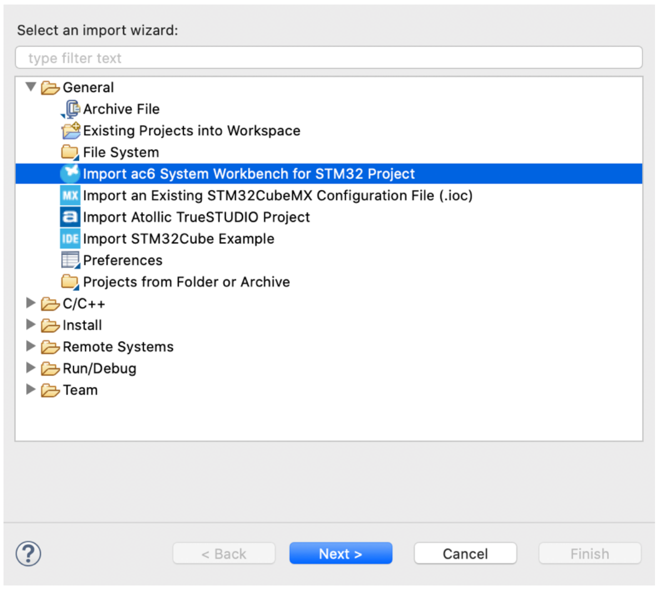
There is no active editor that provides an outline.

Static Stack Analyzer

0 items

7. Select Import ac6 System Workbench for STM32 Project

NOTE THIS SELECTION IS CRITICAL

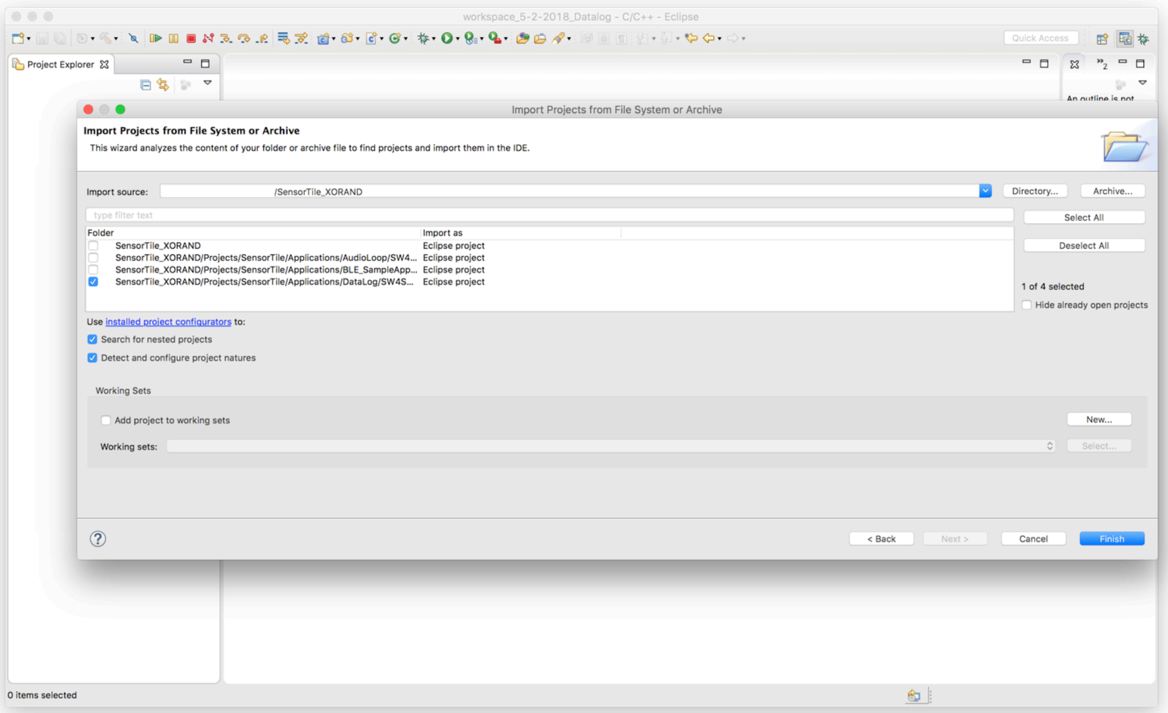


Select an import wizard:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - **Import ac6 System Workbench for STM32 Project**
  - Import an Existing STM32CubeMX Configuration File (.ioc)
  - Import Atollic TrueSTUDIO Project
  - Import STM32Cube Example
  - Preferences
  - Projects from Folder or Archive
- C/C++
- Install
- Remote Systems
- Run/Debug
- Team

< Back | Next > | Cancel | Finish



workspace_5-2-2018_Datalog - C/C++ - Eclipse

Import Projects from File System or Archive

**Import Projects from File System or Archive**

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: /SensorTile_XORAND

| | Folder | Import as |
|---|---|---|
| ☐ | SensorTile_XORAND | Eclipse project |
| ☐ | SensorTile_XORAND/Projects/SensorTile/Applications/AudioLoop/SW4... | Eclipse project |
| ☐ | SensorTile_XORAND/Projects/SensorTile/Applications/BLE_SampleApp... | Eclipse project |
| ✓ | SensorTile_XORAND/Projects/SensorTile/Applications/DataLog/SW4S... | Eclipse project |

1 of 4 selected

☐ Hide already open projects

Use installed project configurators to:

✓ Search for nested projects

✓ Detect and configure project natures

Working Sets

☐ Add project to working sets

Working sets:

0 items selected

8. Then, select Project > Clean

9. Then, select Project > Build All

10. Then select Run > Debug As > STM32 Cortex-M C/C++

## 12. Assignment

Now, in this application, two motion states are detected. Features for classification are then defined.

The first motion state, State 0, includes an orientation change. The second motion state, State 1, includes detection of whether the SensorTile is oriented at a level position or oriented at a vertical position after completion of the first state.

There is a first video showing experimental training of six *motion patterns* is here. The motion patterns may or may not include the second motion state. The EmbeddedML system will be trained to recognize these motions in any order and defined with a second motion state of a level condition or an orientation of the system presenting the Z-axis in a near horizontal state.

Please carefully note how motion is performed as commanded by the LED illumination pattern.

Then, please also observe that after completion of the sixth motion, training occurs. The LED flashes appear at the end of each 500 training epochs. When these cease the system is trained.

There is a video showing experimental training of six motion patterns is here:

**https://youtu.be/TV2ErcPCkyo**

The assignment for this Tutorial is to develop a new system that first detects the first orientation change state exactly as above.

However, the second motion state will be a new development of State 1.

This will detect whether the SensorTile remains motionless in the same orientation that occurs at the end of State 0 or whether there is a change in any direction greater than a threshold. This will be computed by measuring not Z-axis acceleration, but rather the magnitude of acceleration in the X-Y plane. Specifically, $\sqrt{{a_x}^2 + {a_y}^2}$

To accomplish this system you may only need to make a minor modification to the `Feature_Extraction_State_1()` function with the following characteristics.

1) This function compares the Z-axis acceleration with a reference value here:

```
if (abs(ttt[2]) > Z_ACCEL_THRESHOLD){
      *ttt_3 = (*ttt_1 + *ttt_2)/2;}
else {
      *ttt_3 = 0;
}
```

2) If the Z-axis acceleration is greater than a threshold, Z_ACCEL_THRESHOLD, the third feature, ttt_3 is set to the average of the X and Y accelerations. Otherwise it is set to zero.

3) Now, introduce a modification in this code above.

4) Introduce code that compute the magnitude of acceleration in the X – Y plane.

5) Note that $a_x$is measured with ttt[0] and $a_y$ is measured with ttt[1].

6) You may also note that acceleration magnitude is computed elsewhere in the recent tutorials and you may refer to this code.

7) Now, replacing the role of abs(ttt[2}) with this acceleration magnitude will change the operation of this system.

8) You may use the same Z_ACCEL_THRESHOLD or you may wish to adjust this.

9) Note that the system will normalize features and no change in this is required for this assignment. Specifically,

$$F_1 = \frac{a_x}{\sqrt{{a_x}^2 + {a_y}^2}}$$
$$F_2 = \frac{a_y}{\sqrt{{a_x}^2 + {a_y}^2}}$$
$$F_3 = \frac{(F_1 + F_2)}{2}$$

*Please note that this sequence is included in the TrainOrientation() function and need not be modified.*

10) As a hint, to accomplish this assignment, only a minor change is required in the Feature_Extraction_State_1() function.

11) Finally, perform testing and collect a screen capture of output test results.

12) Also, collect a screen capture of your new Feature_Extraction_State_1() function.

13) Submit both screen captures for this assignment.