STMicroelectronics SensorTile Tutorial: Motion Pattern Recognition by Rotation Angle Classification with Machine Learning

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 that includes rotations measured by the SensorTile Rate Gyroscope.

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

We wish to acknowledge the important development by Shuxin Tan of UCLA on development of this novel system.

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 Tutorials 10, 11, and 12.

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.

Orientation was measured by exploiting the acceleration due to gravitational acceleration as detected by the SensorTile accelerometer systems.

Now, in this Tutorial, the SensorTile MEMS Microgyroscope systems will be applied as the signal source. Now, these are rate gyroscopes that measure rotation rate in angle change per unit time. These are calibrated to provide angle measurement in units of milli-degrees per second (mdps). Angular rotation rate is measured for the X, Y, and Z axes.

3. Feature Data Acquisition: Rotation Angle Computation

In order to detect rotation, it is required that we the angular rate be integrated. Considering angular rotation rate R_X about the X-axis, measured angle change about the X-axis, θ_X is

$$\theta(t) = \theta(t=0) + \int_{\tau=0}^{\tau=t} R_X(\tau)d\tau$$

Now, since the SensorTIle digital system samples at discrete time intervals, determined by the sampling period, T_{Sample} , the angle is determined by a sum, replacing the continuous time integral above. Also, time, t, now becomes nT_{Sample}, where n is the index value over all samples. Then, R_X is sampled also at times nT_{Sample}.

Now, computation of this definite integral over discrete values is accomplished by the trapezoidal rule. As an example, consider the function displayed in Figure 1. The continuous function, *R(t)*, is sampled at the time points *t = a* and *t = b*. An estimate for the definite integral between these points (the area under the function between the sample points) is

$$\int_{\tau=a}^{\tau=b} R_X(\tau)d\tau = (b-a)\left(\frac{R_X(b)+R_X(a)}{2}\right)$$

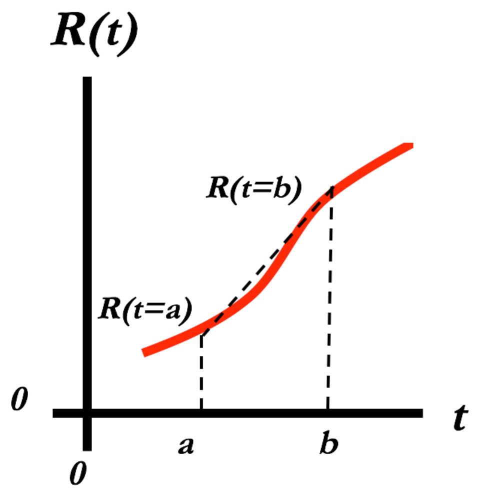


Figure 2. Example of function f(x) with the trapezoidal approximation estimating the area under the function between the interval between x = a and x = b.

Now, the difference, *(b – a)*, is simply the sampling period, T_{Sample}’

Thus, the angle computation for each axis is

$$\theta_X(n) = \theta_X(n=0) + \sum_{i=1}^{i=n} \frac{(R_X(i-1)+R_X(i))}{2} T_{Sample}$$

Similarly,

$$\theta_Y(n) = \theta_Y(n=0) + \sum_{i=1}^{i=n} \frac{(R_Y(i-1)+R_Y(i))}{2} T_{Sample}$$

$$\theta_Z(n) = \theta_Z(n=0) + \sum_{i=1}^{i=n} \frac{(R_Z(i-1)+R_Z(i))}{2} T_{Sample}$$

where T_{Sample} = 0.01 seconds.

4. Feature Data Acquisition: Compensation for Sensor Offset

The Rotation Rate sensor microgyroscope hosted on the SensorTile system is a state-of-the-art device and the most accurate system of its type ever developed or available.

This device is accurately calibrated to measure rotation rate. The sensor gain selected provides a measurement in units of milli-degrees per second. The sampling rate of this project application is set at 100 samples per second corresponding to a sampling period, T_{Sample} = 0.01 seconds.

Now, all motion sensor systems display a finite error, referred to as offset. Specifically, if the sensor is motionless, its rotation rate measurement output is nonzero. This nonzero value leads to computational errors since it is integrated over time thus indicates presence of a rotation.

The offset error signal may be removed directly. This is accomplished by measuring the offset signal for each sensor axis during a time when the sensor is motionless. Then, this offset signal is subtracted from all subsequent measurements. Some applications do not offer a knowledge of a time interval when the sensor system is motionless. This method is effective and sufficient for this rotation angle classification where there is known point in time where the user maintains the sensor motionless.

5. Feature Data Acquisition for Rotation Angle Classification

For this Tutorial, as in previous examples, there will be six instances of motion. For Rotation Angle Classification, the computation of features for each instance will be based on the processing of sensor signals that occur during the period of each instance.

This system will measure the difference in SensorTile angular position between an initial position where the user holds the SensorTile in a starting position, and a final state where the user has rotated the SensorTile and where the *magnitude* of angle change *exceeds a maximum value threshold.*

Angle *magnitude* is computed as

$$\theta_{Mag} = \sqrt{{\theta_x}^2 + {\theta_y}^2 + {\theta_z}^2}$$

The maximum value threshold is adjustable by design and is set to 30 degrees in this application example.

When angle magnitude reaches the maximum value threshold, the *individual* X, Y, and Z axis components of rotation are acquired.

These form the *features:*

$$\theta_x = \theta_{final\_x} - \theta_{initial\_x}$$
$$\theta_y = \theta_{final\_y} - \theta_{initial\_y}$$
$$\theta_z = 0$$

Since no information is required from Z-Axis rotation, this angle change is set to zero.

6. Feature Computation for Rotation Angle 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 rotation values, θ_i , for $i = x \; or \; y$ with:

$$F_1 = \frac{\theta_x}{\sqrt{{\theta_x}^2 + {\theta_y}^2}}$$

$$F_2 = \frac{\theta_y}{\sqrt{{\theta_x}^2 + {\theta_y}^2}}$$

$$F_3 = 0$$

7. Acquisition of Training Data Features

The Neural Network training process is shown in Figure 3. 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.

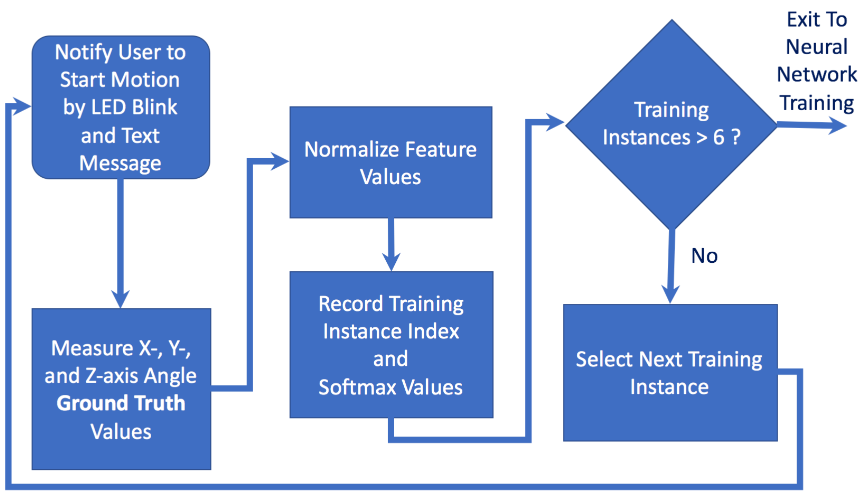

Notify User to Start Motion by LED Blink and Text Message
Measure X-, Y-, and Z-axis Angle Ground Truth Values
Normalize Feature Values
Record Training Instance Index and Softmax Values
Training Instances > 6 ?
Exit To Neural Network Training
No
Select Next Training Instance

Figure 3. The data acquisition and feature computation process. This system executes for 6 training instances and then exits to the Neural Network training process.

8. 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 4.

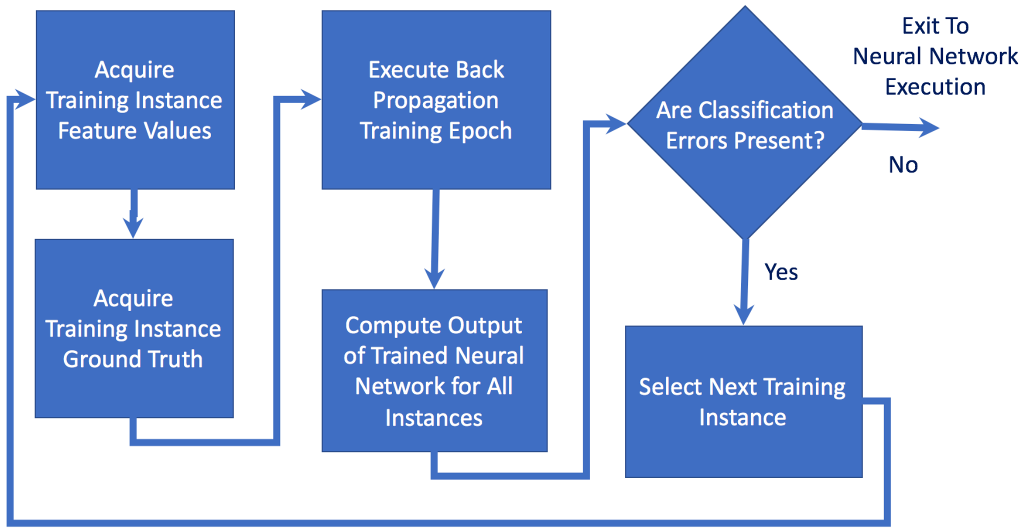

Acquire Training Instance Feature Values
Acquire Training Instance Ground Truth
Execute Back Propagation Training Epoch
Compute Output of Trained Neural Network for All Instances
Are Classification Errors Present?
Exit To Neural Network Execution
No
Yes
Select Next Training Instance

Figure 4. The Neural Network training process executing with the features and Ground Truth obtained from the data acquisition process of Figure 3.

9. 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.

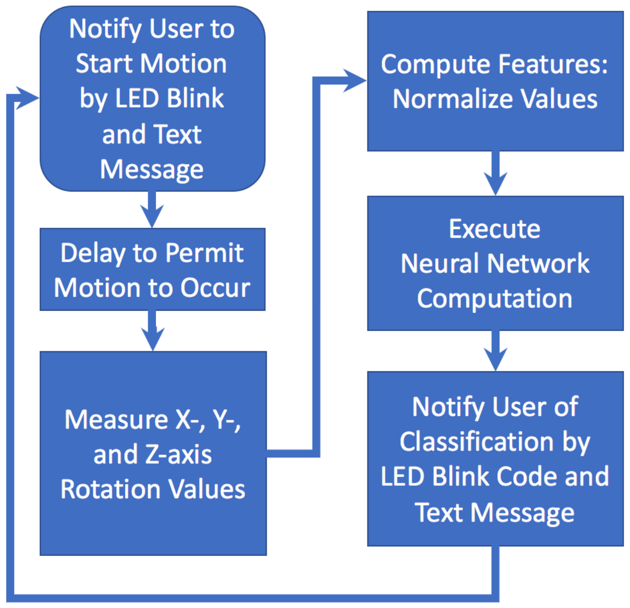

Notify User to Start Motion by LED Blink and Text Message
Delay to Permit Motion to Occur
Measure X-, Y-, and Z-axis Rotation Values
Compute Features: Normalize Values
Execute Neural Network Computation
Notify User of Classification by LED Blink Code and Text Message

Figure 5. The Neural Network execution process operating on data acquired and with the Neural Network trained by the process of Figure 4.

10. EmbeddedML Rotation Angle Classification Software Systems

It is important to review the application software components now.

This system will include the primary data acquisition systems of Tutorials 11 and 12. This previous system provided data sampling with the Feature_Extraction_State_0 () function.

For this Rotation Angle classification, the `Feature_Extraction_Gyro()` function provides sensor signal acquisition and computation of features.

First, rotation rate values are returned as arguments of the getAngularVelocity function. It takes an argument corresponding to a pointer address to the gyroscope sensor system handler data structure. It returns the pointer address to an array containing three array elements that are the current rotation rates for the X, Y, and Z axes.

```
void getAngularVelocity(void *handle, int *xyz) {
        uint8_t id;
        SensorAxes_t angular_velocity;
        uint8_t status;
```

```
        BSP_GYRO_Get_Instance(handle, &id);
        BSP_GYRO_IsInitialized(handle, &status);

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

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

```
        angle_mag_max_threshold = ANGLE_MAG_MAX_THRESHOLD;

        Tsample = (float)(DATA_PERIOD_MS)/1000;
```

Please refer to Figure 3 to while reviewing this function.

```
        getAngularVelocity(handle, ttt_offset);
```

Notify user to initiate motion

```
        BSP_LED_On(LED1);
```

```
for (sample_index = 0; sample_index < MAX_ROTATION_ACQUIRE_CYCLES; sample_index++) {

                for (axis_index = 0; axis_index < 3; axis_index++) {
                        ttt_initial[axis_index] = ttt[axis_index];
                }

                HAL_Delay(DATA_PERIOD_MS);

                getAngularVelocity(handle, ttt);
                for (axis_index = 0; axis_index < 3; axis_index++) {
                        ttt[axis_index] = ttt[axis_index] - ttt_offset[axis_index];
                }
```

The Z-Axis rotation rate information is suppressed for this system (this will be modified later).

```
ttt_initial[2] = 0;
ttt[2] = 0;
```

After the acquisition of rotation rate returned as ttt, the rotate angle is computed by integration using the trapezoidal rule:

```
for (axis_index = 0; axis_index < 3; axis_index++) {
        rotate_angle[axis_index] = rotate_angle[axis_index]
                + (float)((ttt_initial[axis_index] + ttt[axis_index]) *
                Tsample / 2);
}
```

Then, the squared magnitude of angle is computed as the sum of the squared angle values for each of the three axes:

```
angle_mag = 0;
for (axis_index = 0; axis_index < 3; axis_index++) {
        angle_mag = angle_mag + pow((rotate_angle[axis_index]), 2);
}
```

Compute angle magnitude and convert from milli-degrees to degrees

```
angle_mag = sqrt(angle_mag)/1000;
```

First, note the definition of a threshold:

```
#define ANGLE_MAG_MAX_THRESHOLD 30
```

This defines the maximum rotation angle magnitude (angle rotation in any direction) that satisfies the condition

```
angle_mag_max_threshold = ANGLE_MAG_MAX_THRESHOLD;
```

Now, compare the measured angle with the ANGLE_MAG_MAX_THRESHOLD. Break from and exit the measurement loop when angle magnitude exceeds the threshold.

```
if (angle_mag >= angle_mag_max_threshold) {
        BSP_LED_Off(LED1);
        break;
}
```

Now, set the Z-axis feature value to zero (since this is not required for rotation angle recognition about the X and Y axes).

```
        rotate_angle[2] = 0;
```

Then, convert angle (measured in millidegrees) to degrees.

Then, assign feature values

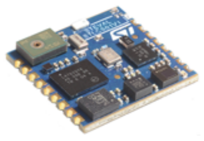
```
        *ttt_1 = rotate_angle[0] / (1000);
        *ttt_2 = rotate_angle[1] / (1000);
        *ttt_3 = rotate_angle[2] / (1000);

        *ttt_mag_scale = (int) (angle_mag * 100);
```

Finally, notify the user that the motion has been completed by turning the LED off.

```
        BSP_LED_Off(LED1);
        return;
```

Then, you may examine the TrainRotation() 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\nPerform Rotation 1 on LED On");
                                CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));

                                Feature_Extraction_Gyro(handle, &ttt_1, &ttt_2, &ttt_3,
                                               &ttt_mag_scale);
                                break;
                        case 1:

                                sprintf(msg1, "\r\nPerform Rotation 2 on LED On");
                                CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));
```



```
                    Feature_Extraction_Gyro(handle, &ttt_1, &ttt_2, &ttt_3,
                                   &ttt_mag_scale);
                    break;
```

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.

```
        sprintf(msg1, "\r\nAngle Values %i\t\%i\%i", ttt_1, ttt_2, ttt_3);
        CDC_Fill_Buffer((uint8_t *) msg1, strlen(msg1));

        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, test_NN);
```

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.

11. EmbeddedML Rotation Angle Classification Execution

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

This occurs in the Gyro_Sensor_Handler_Rotation( ) function. This function takes the arguments of a pointer address to the handler data structure associated with gyroscope 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 `Gyro_Sensor_Handler_Rotation()` function returning data from one 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_Gyro(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;
            }
    }
```

12. EmbeddedML Rotation Angle Classification Project

As in other EmbeddedML projects, this project, is based on an extension of the DataLog application to include EmbeddedML.

SensorTile EmbeddedML System Installation

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

https://drive.google.com/open?id=1w3bu480y-EZP38ml3zJgVXVO52aVxb46

and decompress it to your working directory.

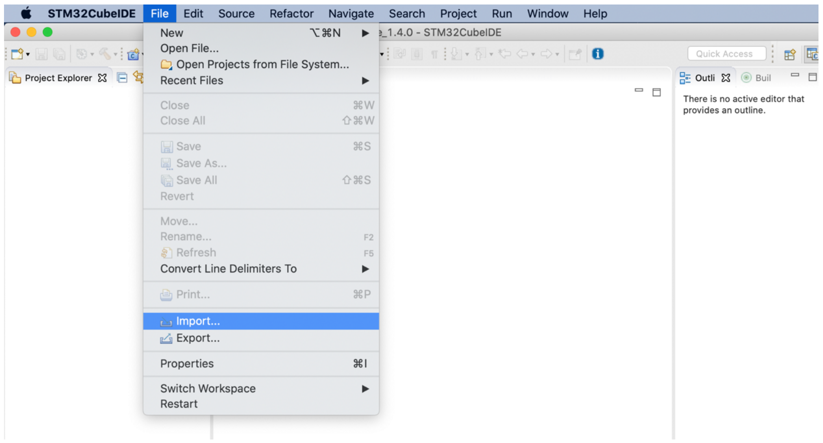
Note: Please download this project above. Please do not use the Projects from previous Tutorials.

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

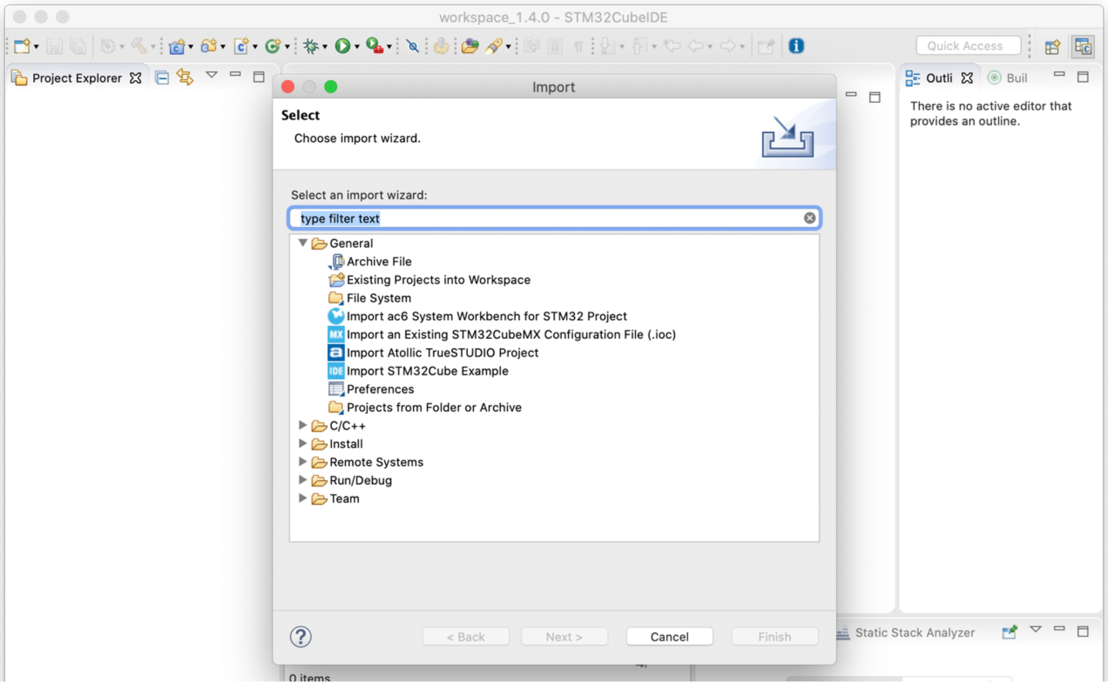
3. Once the IDE is open, first remove your current project.

4. Then, import the new Tutorial 10

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

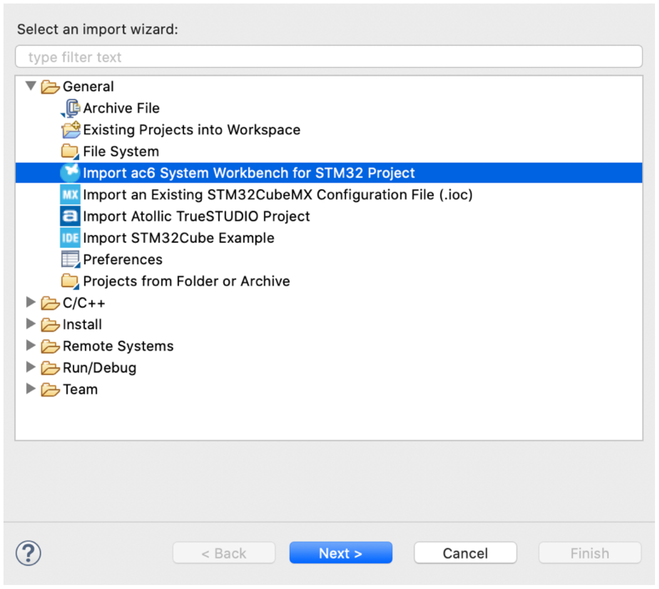


6. Then, Select General


workspace_1.4.0 - STM32CubeIDE
Project Explorer
Quick Access
Outli
Buil
There is no active editor that provides an outline.
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
Static Stack Analyzer
0 items

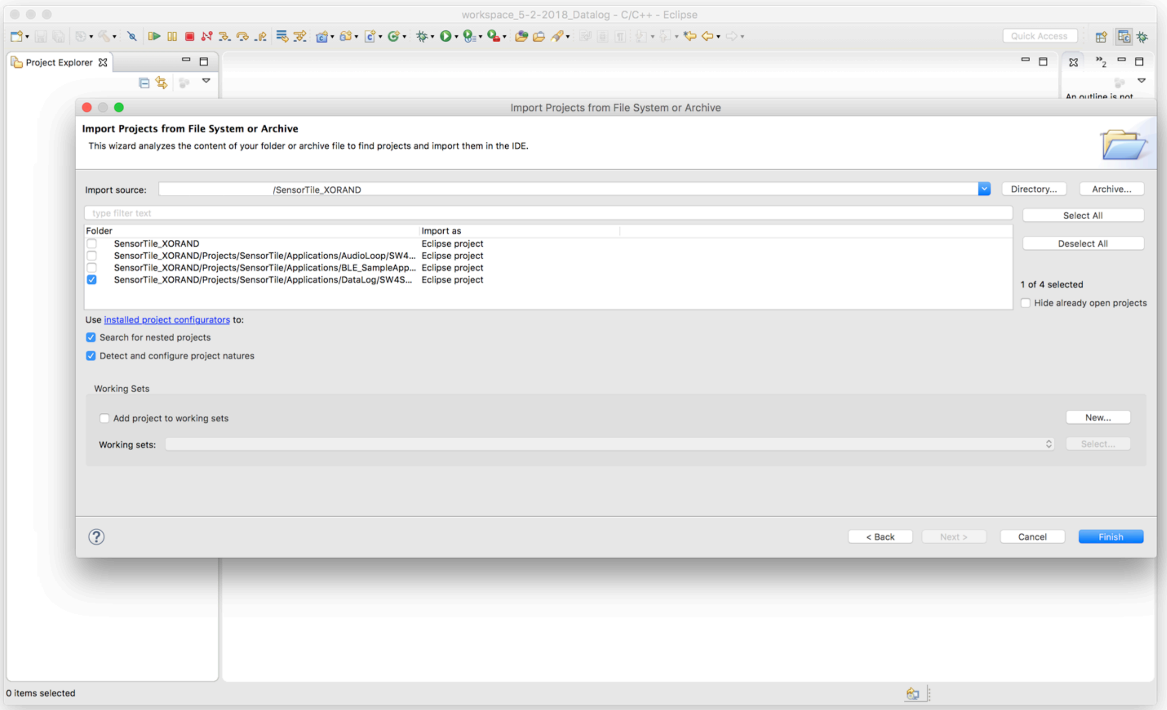
7. Select Import ac6 System Workbench for STM32 Project

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



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++

13. Assignment

The system described above, enables recognition of motion based on rotations that exceed a change in angle from a starting orientation to an angle of 30 degrees for an angle.

The individual angles, θ_x and θ_y, are computed by *integration* of the rotational velocity (rotation rate) for each axis.

Then, the computed angle is the *magnitude* of combined X and Y axis angles:

$$\theta_{Mag} = \sqrt{{\theta_x}^2 + {\theta_y}^2}$$

Motion classification features are *extracted* when θ_{Mag} exceeds 30 degrees.

Motion classification features are the angles, θ_x and θ_y, that are measured at the time when θ_{Mag} exceeds 30 degrees.

In this assignment, modify this to create a system that measures a motion change that includes a change in the magnitude of angle of greater than 30 degrees occurring in any direction that includes rotation angle change of the **X and Z axes**. The new angle magnitude will be

$$\theta_{Mag} = \sqrt{{\theta_x}^2 + {\theta_z}^2}$$

Motion classification features are the angles, θ_x and θ_z, that are measured at the time when θ_{Mag} exceeds 30 degrees.

Now, develop a new system by designing an approach and modifying the appropriate source code in Feature_Extraction_Gyro(). This will require attention to how the three features are computed by Feature_Extraction_Gyro(). Note, that the third of the three features is always set to zero for this system. Thus, the features associated with θ_x and θ_z, must be assigned properly.

Now, it will also be required to select new orientations of the SensorTile such that 6 different motions induce 6 angle changes yielding features that enable recognition of motion. Then, proceed with training and testing of this new system.

Submit screen captures of 1) The source code modifications that enable this new system, and 2) the test results showing successful training.