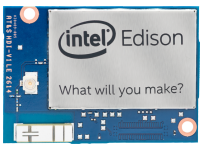


Intel® Edison Tutorial:

Upper Body Motion Classifier

Table of Contents

Introduction....3

List of Required Equipment and Materials....4

Setting up the Classifier....5

Before start: download zip file....5

9DOF Sensors Setup....6

1. Connecting to the clients....6
2. Configuring the sensor parameter....8

Server Edison Setup....10

1. Python libraries setup....10
2. Intel Edison Apache2 setup....10
3. Android Studio/Android smartphone setup....11
4. Collecting data from the sensors....13
5. Feature Extraction on the collected data....14
6. FANN classifier setup....15
7. Classifier Result Display....18

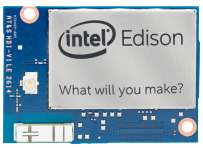
Action Standard....19

Potential Improvement....20

Troubleshooting Guide....21

References....22

Revision history		
Version	Date	Comment
1.0	12/12/2016	Final Draft v1.0

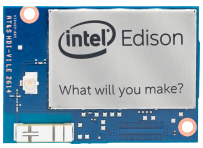


Introduction

This reference design aims to provide instructions on how to use provided code to build a embedded system to classify upper body motions. To do this, three Intel Edison kit will be attached to right elbow, and also left and right wrists by electric tape and set up as client nodes to collect data. The client Intel Edisons will collect data from a 9 Degrees of Freedom Inertial Measurement Unit (9DOF IMU). The 9DOF IMU produces three-dimensional (x, y, z) data collected from a micro-accelerometer, micro-gyroscope, and micro-magnetometer. This data will be transferred to a central Intel Edison, operating as a server. The server will log data read from the 9DOF IMU to a comma separated value (CSV) file. The server will spawn three separate threads to service each incoming TCP/IP client connection. Upon receipt of data from all three clients, each of the three threads servicing the client will append a data entry to a comma separated value (CSV) file associated with a specific client. This file name will be correlated with the sensor location assigned to the client Intel Edison. After data acquisition window of ten seconds, the CSV files representing the current time window will be closed and new CSV files will be created to record the collected data for the next ten seconds. A feature extraction program written in Python will start processing the collected data and prepare features for Fast Artificial Neural Network(FANN) to make classification decision. Then, a html webpage with be generated to reflect the classification result and the Android application will display the current action. The key concepts used to build this project are listed below.

1. Motion sensing.
2. TCP data transport systems.
3. Multi-threaded multi-node client-server architecture.
4. Data logging.
5. Extract features from logged data
6. FANN classifier training and testing
7. Classifier result presentation

Before proceeding with this reference design, users must ensure they are familiar with the Intel Edison, the SparkFun 9DOF kit and FANN. For more information, refer to the document labelled IoT Machine Learning System-Reference Design and IoT Coordinated Motion Sensing-Reference Design.



List of Required Equipment and Materials

1. 1x Intel® Edison
2. A PC or a Mac
3. 4x USB 2.0 A-Male to Micro-B Cable (micro USB cable).
4. 1x powered USB hub OR 1x external power supply.
5. 3x SparkFun 9DOF IMU kit:
    - 3x Base Block https://www.sparkfun.com/products/13045
    - 3x 9DOF IMU https://www.sparkfun.com/products/13033
    - 3x Battery Block https://www.sparkfun.com/products/13037
    - 3x Hardware Pack https://www.sparkfun.com/products/13187
6. 1x Roll of Electric Tape
7. Access to a network (Preferably: **eduroam**).
8. 1x Android smartphone (this tutorial uses Samsung S3)

Setting up the Classifier

Before start: download zip file

- Download zip file and unzip it, it should contain three folders including EE202C_Collect, EE202C_Classify and PreciseClassificationApp-v2.0, they should be arranged in the following ways:
    - EE202C_Collect should be put under Intel Edison directory **/home/root**.
    - EE202C_Classify should be put under Intel Edison directory **/home/root**.
    - PreciseClassificationApp-v2.0 should be put in PC.

9DOF Sensors Setup

1. Connecting to the clients

- Access the three 9DOFs respectively, plug the micro USB into the terminal marked "console" shown in Figure 1. This will turn the Edison on, even if the battery is off.
    - log in to Edison
    - **$configure_edison --wifi**
    - select "**eduroam**"
    - enter the network username(UCLA email address with username@ucla.edu)
    - enter UCLA network password
    - **$configure_edison --showWiFiIP**
    - write down the IP addresses on tapes and label the three 9DOFs with tape

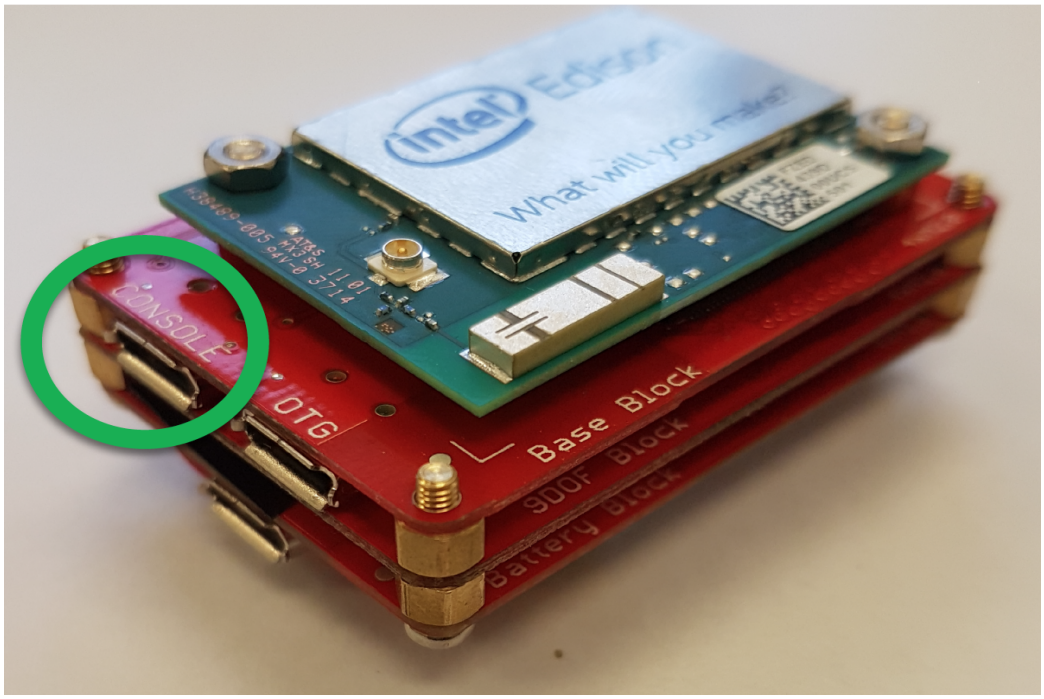


Figure 1: Client Edison Diagram

Note: UCLA **eduroam** network assigns constant IP address to the same device in a location, therefore it is suggested to remain in the same location every time while developing the project.

Figure 2: Back of Client Edison

- Attach the three 9DOFs to user's both wrists and elbow of right hand, respectively as Figure 3 shown.

Figure 3: Location of Client Edisons

- SSH into the 9DOFs
  - **$ssh root@IPaddress**
  - enter password
- Create a new directory labelled IoT.
  - **$ cd**
  - **$ mkdir EE202C_Collect**
- Transfer the folder labelled EE202C_Collect to the root directory of Intel Edison 9DOF sensors.
- The EE202C_Collect folder should now include the following files.
  - **9DOF.c**
  - **9DOF.h**
  - **LSM9DS0.c**
  - **LSM9DS0.h**
  - **main.c**
  - **server.c**
  - **server.h**
  - **Makefile**

2. Configuring the sensor parameter

- Configure parameters:
  - **9DOF.c** initializes the 9DOF sensor with predefined values as Figure 4 shown below:
    - e.g. sample rate, resolution for accelerometer, gyroscope and magnetometer.

```
//set scale
set_accel_scale(ninedof->accel, ninedof->a_scale);
set_gyro_scale(ninedof->gyro, ninedof->g_scale);
set_mag_scale(ninedof->mag, ninedof->m_scale);

//set sample rate setup
set_accel_ODR(ninedof->accel, A_ODR_800);
set_gyro_ODR(ninedof->gyro, G_ODR_760_BW_50);
set_mag_ODR(ninedof->mag, M_ODR_100);

//calculate resolution
ninedof->a_res = calc_accel_res(ninedof->a_scale);
ninedof->g_res = calc_gyro_res(ninedof->g_scale);
ninedof->m_res = calc_mag_res(ninedof->m_scale);
```

Figure 4: Sensor Initialization Function

- Predefined constants are listed in **LSM9DS0.h** (Figure 5).

```
// accel_scale defines all possible FSR's of the accelerometer:
typedef enum
{
    A_SCALE_2G, // 000:  2g
    A_SCALE_4G, // 001:  4g
    A_SCALE_6G, // 010:  6g
    A_SCALE_8G, // 011:  8g
    A_SCALE_16G // 100:  16g
} accel_scale_t;
```

Figure 5: Predefined Sensor Parameter Values

- Compile the source files into an executable binary file.
  - **$ make**

Server Edison Setup

1. Python libraries setup

- Install required Python libraries to Intel Edison
  - Numpy: **$pip install numpy** (**Note**: this should take around 20 minutes)
  - Matplotlib: **$pip install matplotlib**
- Checking if the libraries are installed correctly
  - entering python editor by entering **python** in terminal/putty
  - try to import the libraries by entering:
    - **import numpy**
    - **import matplotlib**
  - libraries are installed correctly if there are no error messages

2. Intel Edison Apache2 setup

- Apache2 installation on Intel Edison, choose either of the two methods below:
  - **Method 1**: Follow the steps below in sequence
    - **$ echo src/gz all http://repo.opkg.net/edison/repo/all > /etc/opkg/base-feeds.conf**
    - **$ echo src/gz all http://repo.opkg.net/edison/repo/all > /etc/opkg/base-feeds.conf**
    - **$ echo src/gz edison http://repo.opkg.net/edison/repo/edison >> /etc/opkg/base-feeds.conf**
    - **$ echo src/gz core2-32 http://repo.opkg.net/edison/repo/core2-32 >> /etc/opkg/base-feeds.conf**
    - **$ opkg update**
    - **$ opkg install apache2**
  - **Method 2**: Install apache2 which includes PHP 5 from AlexT's repo
    - **$ wget http://repo.opkg.net/edison/repo/core2-32/apache2_2.4.20-r0_core2-32.ipk**
    - **$ opkg install apache2_2.4.20-r0_core2-32.ipk**
- To enable Apache2, type the following command:
  - $ **systemctl reenable apache2**
- To check if Apache2 service is loaded, enabled and running correctly, type:
  - $ **systemctl status apache2**
- If things go wrong with apache2, type the following commands to reboot:
  - **$ systemctl restart apache2**
  - **$ reboot**

- After successfully installed Apache2, index.html should reside in folder /usr/share/apache2/htdocs/, any modifications to folder files will have effects on the website corresponding to the Intel Edison IP address
    - e.g. Modifying index.html will display itself on http://111.222.33.44/index.html
    - e.g. Adding new files prediction_results.html will display itself on http://111.222.33.44/prediction_results.html

3. Android Studio/Android smartphone setup

- Open Android Studio project
    - Install Android Studio from https://developer.android.com/studio/index.html
    - Open project "PreciseClassificationApp-v2.0" in Android Studio
    - Change the project structure to "Android" mode (Figure 6), and open fragment1.java

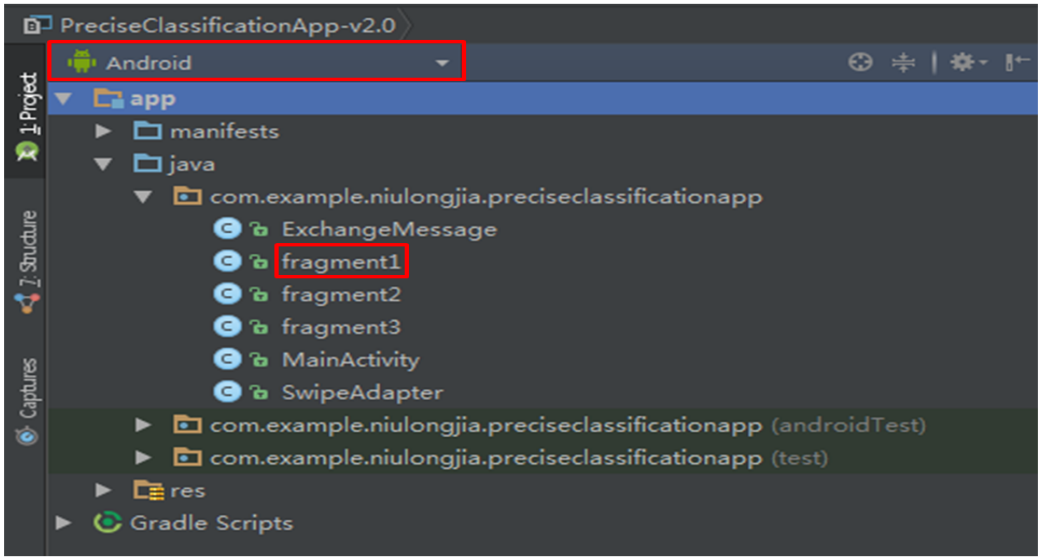


Figure 6: Android Application Project Directory

    - In fragment1.java (Figure 7), change the default IP address to the server Intel Edison IP address

```
fragment1.java
package com.example.niulongjia.preciseclassificationapp;

import ...

/**
 * A simple {@link Fragment} subclass.
 */
public class fragment1 extends Fragment {
    Handler uiHandler = new Handler();

    Document siteDoc;
    String stateString="";
    Elements p;

    Button button;
    TextView textView;
    ImageView imageView;

    // need to change this every time !
    // need to connect android phone with wifi !
    String URL="http://131.179.1.199/prediction_update.html";
```

Figure 7: Fragment.c File

- Configure android smartphone
  - Connect android smartphone with PC, using microUSB cable
  - Open Settings/More and search for **Developer options**
    - If "Developer option" is not available, Tap "**About Device**" and tap "**Build number**" for seven times
  - Return to **Settings/More** and tap **Developer Options**
  - Turn on **Developer option** and **USB debugging**
  - It may need some time to install driver for smartphone if the phone is connected for the first time.

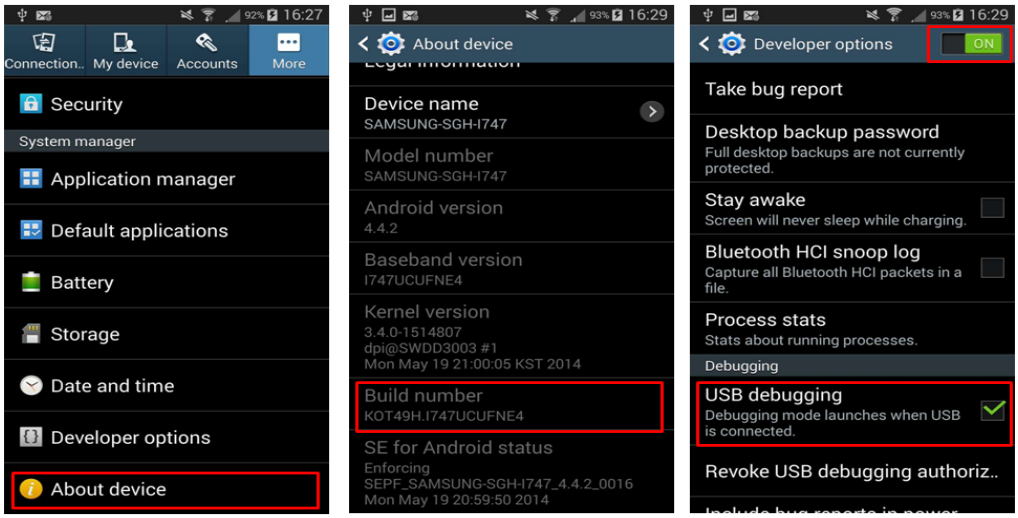


Figure 8: Android Smarting Settings

4. Collecting data from the sensors

- After downloading project zip file, unzip it and place EE202C_Collect folder and EE202C_Classify folder under Intel Edison directory /home/root. For collecting data from three 9DOFs, there following files should be in /home/root/EE202C_Collect/server:
    - **9DOF.c**
    - **9DOF.h**
    - **LSM9DS0.c**
    - **LSM9DS0.h**
    - **main.c**
    - **server.c**
    - **server.h**
    - **Makefile**
- Connection establishment:
  To establish a client-server connection, each client needs to know the server's IP ahead of time. Then execute the following commands to establish the connection:
    - Server side: **$ ./server <PORT_NO>**
    - Client side: **$ ./client <IP_ADDR_SERVER> <PORT_NO>**
- A global variable **num_clients** is introduced to keep track of the number of connected clients. The data recording process will not start unless the number of connected clients reaches three.
- To distinguish every sensor with each other, the very first line each client sent to the server is its location. For example, the first line the sensor at right wrist sent to the Edison board is "NINEDOF_1_RIGHT_WRIST". Then the server is able to tell which client is at which location.

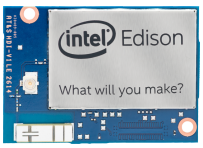
- The real-time effect is achieved by predicting the motion in the past 10 secs. To achieve that, **alarm(10)** function is called every 10 second to trigger an alarm signal SIGALRM. Every time a **SIGALRM** signal is received, the csv files recording raw data from the past 10 second are closed and three new csv files for the current 10 seconds are created for writing. Therefore, after a number of seconds, the directory should look like Figure 9:

```
NINEDOF_1_RIGHT_WRIST_1.csv      NINEDOF_2_RIGHT_ELBOW_6.csv
NINEDOF_1_RIGHT_WRIST_10.csv     NINEDOF_2_RIGHT_ELBOW_7.csv
NINEDOF_1_RIGHT_WRIST_11.csv     NINEDOF_2_RIGHT_ELBOW_8.csv
NINEDOF_1_RIGHT_WRIST_2.csv      NINEDOF_2_RIGHT_ELBOW_9.csv
NINEDOF_1_RIGHT_WRIST_3.csv      NINEDOF_3_LEFT_WRIST_1.csv
NINEDOF_1_RIGHT_WRIST_4.csv      NINEDOF_3_LEFT_WRIST_10.csv
NINEDOF_1_RIGHT_WRIST_5.csv      NINEDOF_3_LEFT_WRIST_11.csv
NINEDOF_1_RIGHT_WRIST_6.csv      NINEDOF_3_LEFT_WRIST_12.csv
NINEDOF_1_RIGHT_WRIST_7.csv      NINEDOF_3_LEFT_WRIST_2.csv
NINEDOF_1_RIGHT_WRIST_8.csv      NINEDOF_3_LEFT_WRIST_3.csv
NINEDOF_1_RIGHT_WRIST_9.csv      NINEDOF_3_LEFT_WRIST_4.csv
NINEDOF_2_RIGHT_ELBOW_1.csv      NINEDOF_3_LEFT_WRIST_5.csv
NINEDOF_2_RIGHT_ELBOW_10.csv     NINEDOF_3_LEFT_WRIST_6.csv
NINEDOF_2_RIGHT_ELBOW_11.csv     NINEDOF_3_LEFT_WRIST_7.csv
NINEDOF_2_RIGHT_ELBOW_2.csv      NINEDOF_3_LEFT_WRIST_8.csv
NINEDOF_2_RIGHT_ELBOW_3.csv      NINEDOF_3_LEFT_WRIST_9.csv
NINEDOF_2_RIGHT_ELBOW_4.csv      server_test_data.csv
NINEDOF_2_RIGHT_ELBOW_5.csv
```

Figure 9: CSV file list in working directory

5. Feature Extraction on the collected data

- EE202C_Classify/FANN_system should contain two .py files:
    - prepare_train_data_normalize.py
    - prepare_test_data_normalize.py
- Prepare training data offline for FANN(prepare_train_data_normalize.py):
    - Run the program:
        - **$ python prepare_train_data_normalize.py**
    - The program will search through the current directory for csv files corresponding to three different sensors and extract features to put the results to **coefficient_data_train.txt**.
    - Five features are currently used:
        - standard deviation
        - mean
        - Range(max-min)
        - Root mean square(RMS)
        - Above threshold percentage (number of data entries that is greater than 1.25*mean divided by the total number of data entries)
    - Format of coefficient_data_train file is outlined as following:
        - Each line is constructed by 90 feature values



            - 5 features per sensor axis * (3 accelerometer + 3 gyroscope) * 3 sensors in total
        - Each line is ended with state indication
            - 6 single-bit binary numbers indicating the action:
            - e.g. **1 0 0 0 0 0** means the data belongs to **COOK** action
            - more of this is explained in the next section: **FANN classifier setup**
        - Last line has three entries separated by space:
            - number of data entries
            - number of input features
            - number of output states
- Prepare test data offline for comparison for FANN:
    - Two modes of operation are supported for preparing the test data: **multi-dataset mode** and **real-time mode**
    - **Multi-dataset mode**(prepare_train_data_normalize.py):
        - Change **multi_dataset_test** variable to **1** to enable this mode.
        - Run the program: **$ python prepare_train_data_normalize.py**
        - The program will search through the current directory for csv files corresponding to three different sensors and extract features to put the results to **coefficient_data_test.txt**.
        - This mode is used when several datasets are collected from multiple time windows and the user wants to test all these datasets together (as known as the **non-realtime** mode).
    - **Real-time mode**(prepare_test_data_normalize.py):
        - Run the program: **python prepare_test_data_normalize.py**
        - The program will start listening for incoming data files. If new files are found, files from the previous time window will be processed and **coefficient_data_test_update.txt** will have one line consisting the extracted features from the sensor data files.

6. FANN classifier setup

- After downloading the project zip file from Github, unzip it and place EE202C_Collect folder and EE202C_Classify folder under Intel Edison directory /home/root. For FANN classification, the following files should be in /home/root/EE202C_Classify:
    - Shell scripts to install Git, VIM and FANN
        - **install_fann**
        - **install_opkg**
    - Neural network C/Python code under the subfolder **FANN_system**. Make sure previous steps are completed and include coefficient_data_train.csv

(mandatory for neural-network training) and coefficient_data_test.csv (optional for offline testing) in the current folder.

- **coefficient_data_train.csv**
- **coefficient_data_test.csv**
- **parameters_listener.h**
- **parameters.h**
- **prepare_train_data_normalize.py**
- **prepare_test_data_normalize.py**
- **train_neural_net.c**
- **test_neural_network.c**
- **test_neural_network_listener.c**
- **Makefile**

- Install FANN library
  - Navigate to the folder that contains the files labelled "install_fann" and "install_opkg"
  - Make the install scripts executable by running the following command:
    - **$ chmod +x install_***
  - If Git is not installed, run the install_opkg script as follows:
    - **$ ./install_opkg**
  - Install the FANN library by running the following command (This step may take about 10 minutes)
    - **$ ./install_fann**

- Compile and run FANN classifier **offline**
  - Navigate to the folder that contains the .c/.py files and the Makefile
  - **Parameters.h** can be modified to change the following property of FANN:
    - input neurons
    - hidden neurons
    - hidden layers
  - Clean previous executive files and compile the code by typing make
    - **$ make clean**
    - **$ make**
  - Train neural-network offline with coefficient_data_train.csv to produce Test.net
    - **$ ./train_neural_net**
  - Evaluating features' weight by checking Test.net file:
    - bigger weight means this input is more significant to the classifier
    - dropping unnecessary features might result in better result
  - Test neural-network offline with coefficient_data_test.csv to examine prediction results
    - **$ ./test_neural_network**

- After that prediction_results.txt will appear in the current folder and shows the prediction results for each row of coefficient_data_test.csv

- Compile and run FANN classifier **online**
  - Navigate to /home/root/EE202C_Collect/Server and run the server to take in data from three 9DOFs. The last script will be running continuously.
    - **$ make clean**
    - **$ make**
    - **$ ./server**
  - Open a new screen and ssh to the directory /home/root/EE202C_Classify/FANN_system and run python to generate **coefficient_data_test_update.txt** in real time by listening to three incoming 9DOF data. The following script will be running continuously.
    - **$ python prepare_test_data_normalize.py**
  - Open a new screen and ssh to the directory /home/root/EE202C_Classify/FANN_system, run c file to test neural-network online with Test.net and "coefficient_data_test_update.txt" generated from previous steps, and write results to /usr/share/apache2/htdocs/prediction_update.html to be ready for website/Android display. The following script will be running continuously.
    - **$ ./test_neural_network_listener**

7. Classifier Result Display

- Run the Android App
  - Open **PreciseClassificationApp-v2.0** project in Android Studio, click button
  - Android ADB will pop up and Android phone should appear as connected Devices

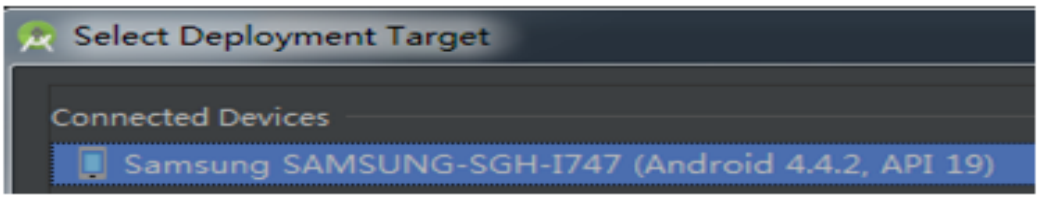


Figure 10: Connected Device List

  - Click OK and wait for APK installing onto the Android smartphone
  - Once installation finished, app will automatically open up
  - Unplug USB cable and use the app with WiFi connected

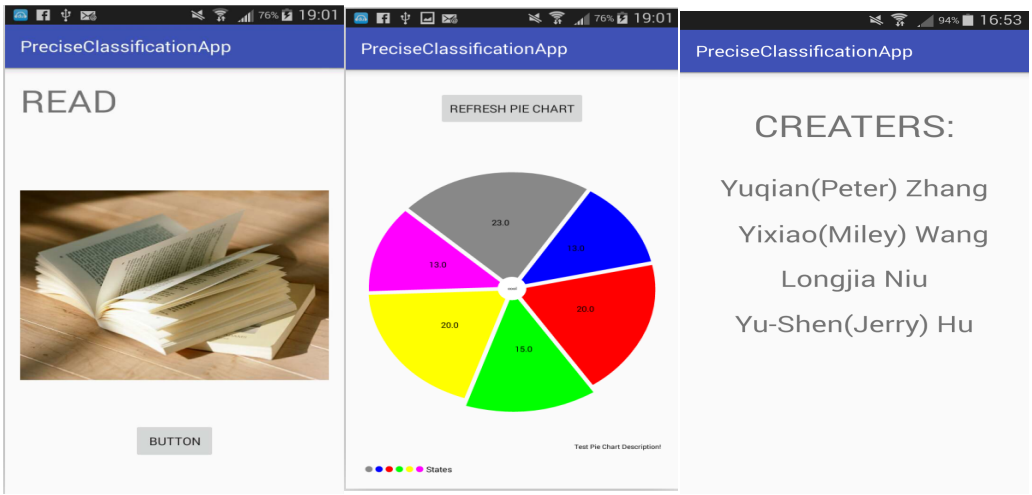


Figure 11: Android App Windows

Action Standard

1. Cooking
2. Reading
3. Resting
4. Typing
5. Walking
6. Grooming

Please refer to the video in the following link to see the standard actions.
https://www.youtube.com/watch?v=wgE2bz41uwE

Potential Improvement

1. Buffer the data before sending to the server Edison
    - Currently the data buffer is 256 byte and client transfers collected sensor data after one sensor reading to the server Edison.
    - Efficiency can be improved by transferring dataset consisting multiple sensor reading together.
2. Preprocessing the data by applying appropriate filters to the collected data and discover more features both in time and frequency domain.
    - Modify prepare_train_data_normalize.py and prepare_test_data_normalize.py:
        - The section that is commented out in line 395 to 425 is an example of mapping the data to frequency domain and extract top frequencies and their intensities as features.
        - Applying filters to data before processing the data can be useful because the collected data might involve significant noises
3. Support Multi-user with more training data
    - Currently the classifier is trained with 150 datasets of each action from a single user. Multi-user action classification can be enabled by adding more training dataset collected from different user doing those pre-defined actions.

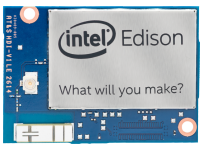
Troubleshooting Guide

1. Unable to access to client/server Edison
    - Log into the Edison and enter "**rfkill unblock all**". If the problem is not solved, proceed to next step.
    - Enter "**ifconfig wlan0 down**" and then "**ifconfig wlan0 up**". If the problem is not solved, proceed to next step.
    - Enter "**vi /etc/wpa_supplicant/wpa_supplicant.conf**". Delete everything except the first eight lines as shown in Figure 12 below.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
config_methods=virtual_push_button virtual_display push_button keypad
update_config=1
fast_reauth=1
device_name=Edison
manufacturer=Intel
model_name=Edison
```

Figure 12: wpa_supplicant.conf

    Then, reboot the Edison and configure WiFi again. If the problem is not solved, proceed to next step.
    - Refer to "Intel Edison Troubleshooting Guide" and proceed to "**Updating Firmware**" section.
2. Unable to install numpy/matplotlib Python library
    - update pip by: **$easy_install --upgrade pip**
    - Then try:
        - **$sudo pip install numpy**
        - **$sudo pip install matplotlib**
3. In **real-time mode** make sure to wait for 10 seconds before executing:
    - **$python prepare_test_data_normalize.py**
4. Please check Intel® Edison Troubleshooting Guide for Edison's unexpected behavior/error



References

1. Intel® Edison Tutorial: Introduction and Connectivity
2. Intel® Edison Tutorial: IoT Coordinated Motion Sensing Reference Design
3. Intel® Edison Tutorial: IoT Machine Learning System Reference Design
4. Intel® Edison Troubleshooting Guide
5. https://developer.android.com/reference/android/os/AsyncTask.html
6. https://developer.android.com/training/multiple-threads/communicate-ui.html