LLP108-IoT
Coursework Report

Student ID: B929899

2020

Introduction

In the past two decades, the way humans interact with technology devices has drastically changed. From using buttons, transitioning to styluses and ending up with fingers as of nowadays. The innovation has been gradually removing the “middleman” of the interaction and this trend is unlikely to change in future. Hence, even the fingers will most likely become obsolete soon. One of the well known and applied techniques is a gesture based interaction. However, it is only yet to find it’s broader applications with consumer electronics. Up till now, it only found its place in gaming and VR.

This report outlines a coursework project that has been done to re-create a simple gesture-device interaction interface based on the provided sensor. The work utilizes IoT methodology with a combination of signal processing and machine learning to connect, configure the sensor and collect and process advertised data real-time.

Overall, the objective of the project was to simulate a wearable device such as a smartwatch with the sensor and re-create the user experience through the gesture-based interface which emulates a trivial control of the device, via gestures. The interface has been developed with a binary functionality - accept / decline type of interaction.

Solution

This project addresses a gesture-based interaction interface problem using only a 3 dimensional (x,y,z axes) accelerometer data.

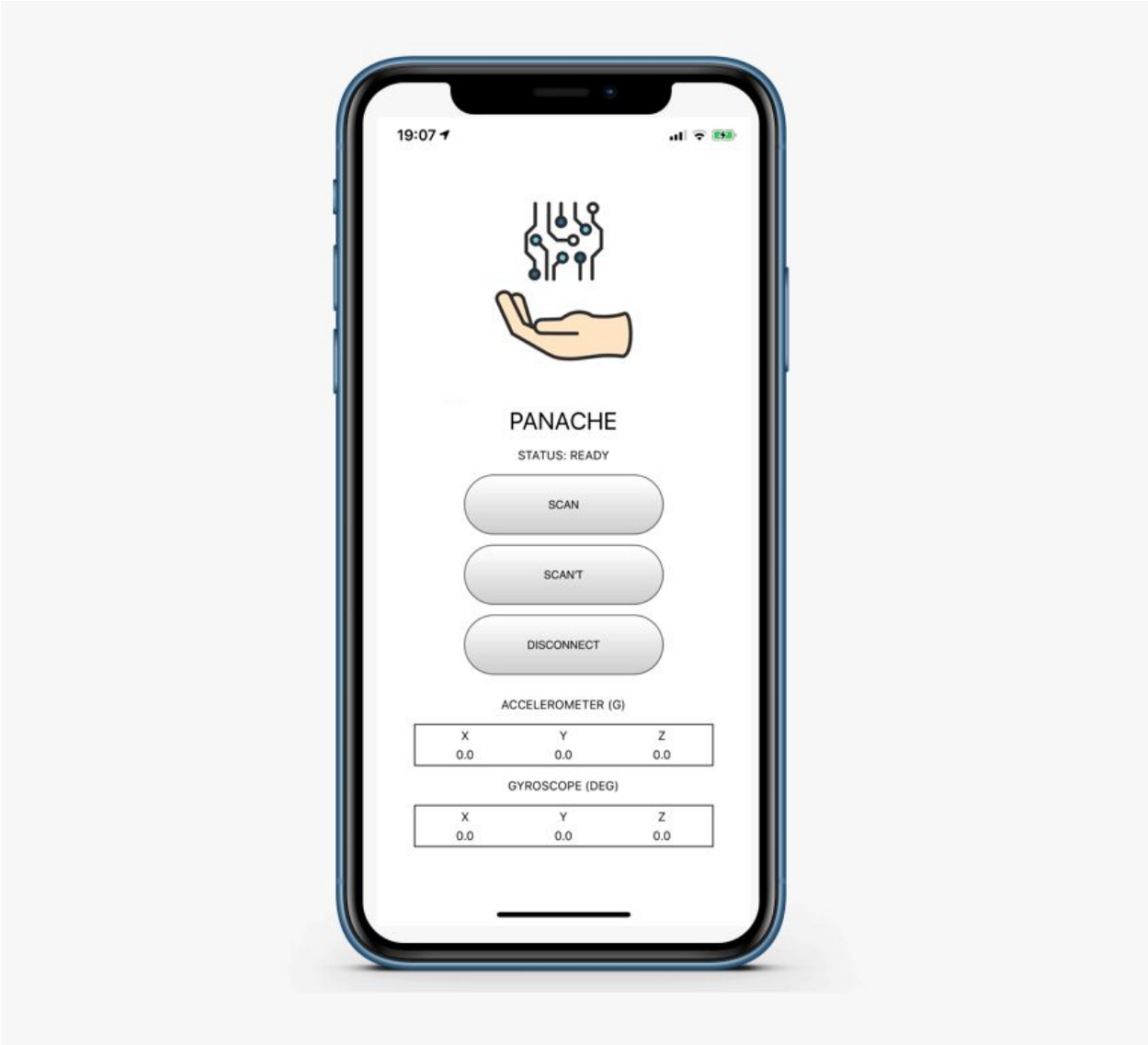
Outline

The provided hardware equipment is a piece of TI SensorTag CC2650STK. Moreover, the project as a whole is an extensive software engineering work that by-directionally interacts with the sensor and utilizes machine learning while doing so.

The workflow of the solution is briefly displayed in a flowchart - figure 2.

1. The AI-powered iOS Panache app is launched, and can be further controlled via developed front-end user interface (Figure 1.). A user can press the “SCAN” button and the app automatically scans for nearby BLE devices (Bluetooth Low Energy, 2020). The Panache App is automatically configured to connect to the provided TI SensorTag device and read the configured advertised data which are in this case accelerometer data.

Figure 1. Panache App UI



2. As soon as the app receives the data, it forwards it to a configured cloud server via an HTTP post method in batches of 10 readings.

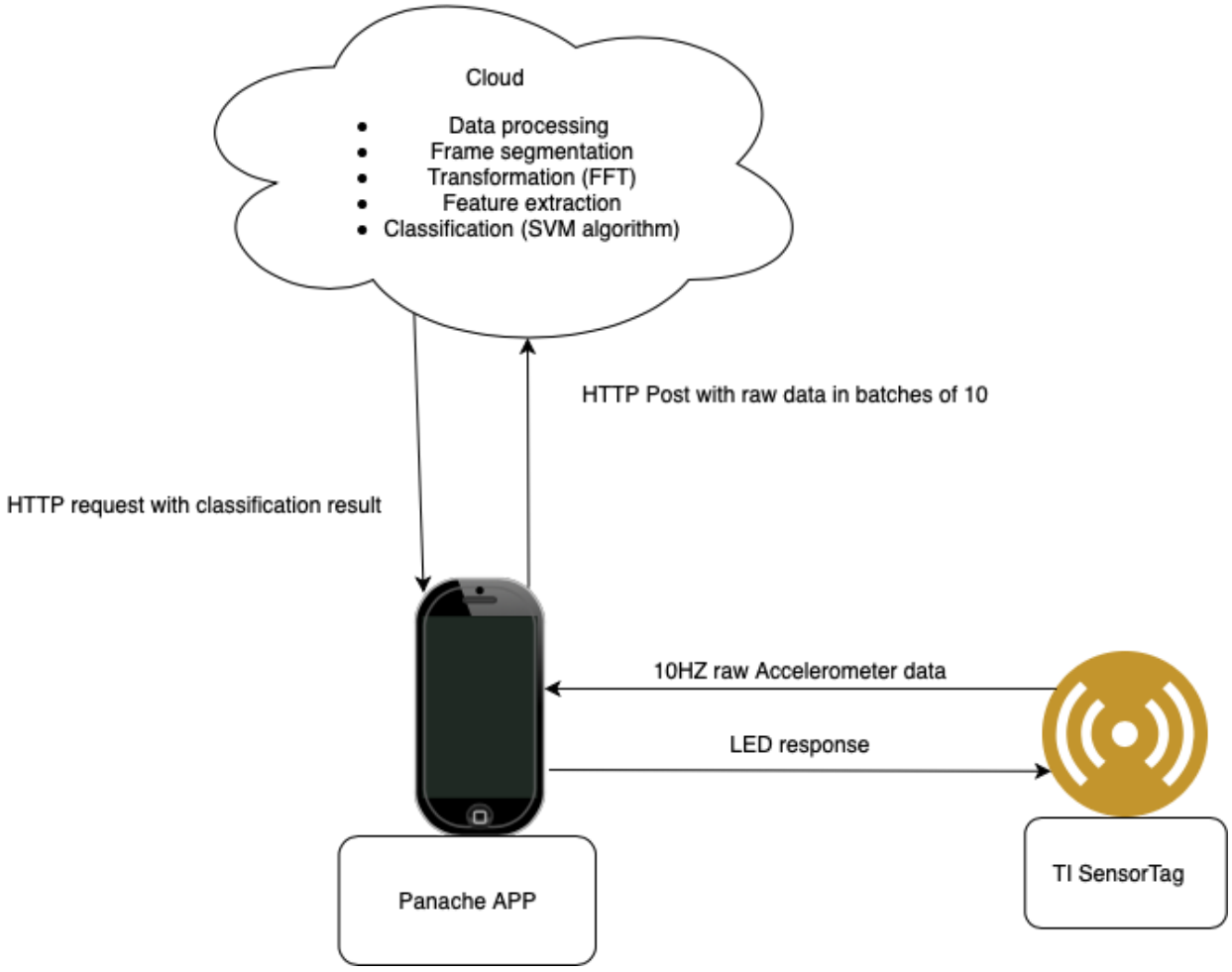
3. A server together with a micro-web framework contain an extensive software module that is responsible for collecting data. Afterwards the module conducts initial pre-processing, frame segmentation, Discrete Fourier Transform (DFT) transformation, feature extraction and finally a trained Support Vector Machine (SVM) classifier. As the sample is processed and transformed into a classifiable input, it runs through the

classifier which returns a result. The result can be “0”,”1” or “2” depending on the classification. “1” represents the gesture 1 - “accept”, “2” represents gesture 2 - “decline” and zero stands for none of those.

4. The results are requested by the app web framework and are rendered back to the smartphone. Depending on the value of the result, the app sends a command that configures the LED to flash. A green flash represents the gesture 1 (“accept”) while red flash represents the “decline” gesture. If the result of the classification is not within the classification value thresholds, neither of the LED flash.

Every reading represents 1 request which translates into 1 classification trial and output request.

Figure 2. Workflow



Implementation & back-end workflow

The Panache interface has 2 main components - 1) iOS App and 2) Machine Learning, which divide into their respective subcomponents.

The implementation of the ML component - gesture classification was implemented based on the work conducted in 2009 which achieved 99%+ accuracy (Wu, Jiahui & Pan, Gang & Zhang, Daqing & Qi, Guande & Li, Shijian. (2009).

(The list of all resources used for the project can be found under “Project Resources“)

Panache App

There is a pre-made TI app called Starter that connects to the provided SensorTag, reads the advertised data, even offers some configuration and has a convenient way of sending data to the default IBM cloud solution. However, this set-up is proprietary and lacks the flexibility that is required for creating a gesture-machine interface that can seamlessly interact with the sensor in more of a programmatic way. Therefore I decided to make an app which offers me a full flexibility and configuration without the hassle of using a stack of closed proprietary solutions. For the iOS app, I have primarily used Xcode (IDE for development of Apple based software) where I have developed a Cordova based app in JavaScript. The Cordova BLE library offers a full range of functions that can be conveniently integrated to operate a remote BLE device/sensor. Together with the official SensorTag documentation, it enabled me to fully configure the sensor according to the needs of the project.

The app is accompanied with a light html based front-end which offers a basic UI to the user. On the back-end, the app firstly scans for nearby BLE devices and if the device ID matches the configured ID of the sensor, the app automatically connects to it and configures the characteristics of the accelerometer service ID so the accel data are advertised.

The sampling rate of the advertised data was configured in the “Period” characteristics of the device and was set to the 10Hz (10 readings / 1 sec) which is the maximum the sensor offers. Normally, gesture-based classification is implemented with 80Hz+ sampling rate. The significantly lower 10Hz rate might be one of the reasons for the reduced accuracy of the classifier.

According to the mentioned paper, the optimum input for the classification is 10 frames of readings. Hence the app has been configured to collect 10 readings before posting them to the server.

The overall implementation of the sensor was done on a low-level basis which resulted in learning about a beautiful low-level way of configuring such hardware.

Response

After the app receives a response value from the classification model hosted on cloud (heroku service) it decides whether and which LED to flash. The business logic is implemented as in the screenshot below:

```
const { data } = await axios.post('https://panache1.herokuapp.com/track_data', ACCEL_DATA)
// IOv1 activating / deactivating

if(data == "1" && IN_TIMEOUT==false){
    ble.write(DEVICE_ID, IO_SERVICE_ID, IO_DATA, activ_GREEN.buffer)
    IN_TIMEOUT = true
    setTimeout(function(){
        ble.write(DEVICE_ID, IO_SERVICE_ID, IO_DATA, de_activ_GREEN.buffer)
        IN_TIMEOUT=false
    }, 2000);
    }
if (data == "2" && IN_TIMEOUT==false){
    ble.write(DEVICE_ID, IO_SERVICE_ID, IO_DATA, activ_RED.buffer)
    IN_TIMEOUT = true
    setTimeout(function(){
    ble.write(DEVICE_ID, IO_SERVICE_ID, IO_DATA, de_activ_RED.buffer)
    IN_TIMEOUT=false
    }, 2000);
    }
```

Since the logic is refreshed with every request, a flag variable called “IN_TIMEOUT” was implemented to ensure the flash of the led would duration for 2000miliseconds regardless of a new request refresh. If the classification is successful, it sets the flag variable to **true** and executes the LED flash after which the IN_TIMEOUT variable is reset back to **false** again.

Classification model

The classification model is a server-side implementation which firstly collects the input coming from the app in batches of 10 readings every second and classifies it. The model is an extensive work and includes numerous sub components described below. As mentioned, the methodology of the implementation is based on the referenced paper however since the paper only explains methodology not the actual implementation, numerous measures have been implemented to handle the data inputs and outputs accordingly.

1. **Data processing**

The input is initially cleaned up to be processed further. This involves getting rid of noise data and making sure the sample is based on 10 readings.

2. **Frame segmentation**

The processed input data is segmented into frames where each frame F is composed of N reading combined with N+1 readings identical in length. This creates 10 overlapping frames each composed of 2 readings. The frame segmentation reduces the effect of intra-class variation and noise.

3. Signal processing

According to the signal processing theory, a frequency domain is rich in features which contain information about the signal. In order to transform the data from the time-based domain to frequency domain, a Discrete Fourier Transform has been implemented on each frame per axes using the Scikit learn python library.

4. Feature extraction

The final feature set is composed of features in both time-space domain and frequency domain. For the time-space domain, standard deviation of the axes in the components of the frame and a correlation between the respective axes have been extracted. In frequency-domain of the transformed sample, mean and energy have been computed.

Finally, All 4 of the extracted features have been concatenated into a feature vector where each axis in a frame has 6 features, which translates into 180 features for every sample where sample is based on 10 frames.

5. Classification model

Lastly, the feature vector is being run through the pre-trained Support Vector Machine model. The model evaluates the input vector and outputs a prediction value.

Since SVM is a binary classifier dealing with high dimensional feature space (the model was trained on nearly 100 000 features) it requires some further classification algorithm to classify between thresholds of the output values. If the thresholds are too low, the model will classify the gesture as 0 - no gesture. The return value is back propagated through the http request to the app and back to the sensor. The business logic is based on the IF statements to turn on and off respective LED lights accordingly to the response value.

Figure 3. Threshold classifier

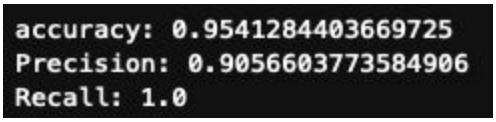
```
def clfout(sample):
    sample = sample.reshape(1,-1)
    if clf.decision_function(sample) <=(-1.2):
        return 1
    elif clf.decision_function(sample) >= 1.0:
        return 2
    else:
        return 0
```

Furthermore, the model was trained on over 540 samples (1 sample = 10 frames = 10 readings) including both of the gestures. The data has been collected manually over the period of a few minutes interacting with the sensor. The interaction / data collection involved a repetitive motion for both of the gestures. For the “decline” gesture, the wrist was repetitively twisted left and right. The “accept” gesture required a “rock-paper-scissors” movement with the forearm in vertical

directions. The reduced accuracy of the model is likely to be caused by the lack of data as well. According to the current research on similar projects, the datasets normally include weeks of continuous accelerometer motion data. This project has used just minutes which is a significant reduction in the overall train dataset.

It did result in high binary accuracy - 95%, however introducing the 3rd output option - 0 / no gesture has likely declined the overall practical classification accuracy.

Figure 4. Model metrics



Conclusions & future considerations

The project shows a simplicity and effectiveness in using accelerometer data for gesture classification which is substantial enough to create a gesture-machine interface.

In the testing phase, the classification model achieved a decent above 90% accuracy however for practical applications this would need to be slightly higher to be usable in real-world consumer devices.

However, at this stage of the project there is still a room for improvement. Few considerations are outlined below to increase the accuracy of the model, hence overall success of the project as a whole.

- Larger training dataset - thousands of samples instead of hundreds
- More qualitative feature extraction - experimentation of different and new features
- Decision threshold optimisation - this has not been done appropriately since the lack of time scope for the project.
- Stronger data processing - data normalization could have made the SVM model more accurate
- Overall UI and product use case implementation

References

Wu, Jiahui & Pan, Gang & Zhang, Daqing & Qi, Guande & Li, Shijian. (2009). Gesture Recognition with a 3-D Accelerometer. 5585. 25-38. 10.1007/978-3-642-02830-4_4.

En.wikipedia.org. 2020. *Bluetooth Low Energy*. [online] Available at: <https://en.wikipedia.org/wiki/Bluetooth_Low_Energy> [Accessed 14 April 2020].

Project Resources

Cordova BLE library
https://github.com/don/cordova-plugin-ble-central#write

Official SensorTag CC2650STK documentation
https://processors.wiki.ti.com/index.php/CC2650_SensorTag_User%27s_Guide#Movement_Sensor

Axios - http client for Node JS based applications
https://github.com/axios/axios