Flashing Jacket

Milestone Report

Alex Michon, Dinesh Parimi, Jack Wan

November 21st, 2017

Overview

The main goal of our project is to create a LED grid to make biking safer at night. Our device will detect arm movements of the cyclist and create light signals to indicate in which direction the cyclist wants to turn. The LED grid will be attached to the back of the cyclist's jacket.

We also thought about other potential applications of a wearable led grid, so we will implement another feature. Our device will be able to respond to ambient noise. For example, it should be able to display a spectrogram based on the current music.

Requirements

- Detect and distinguish the three cyclist arm signals: left, right and stop (see Figure 1)
- Detect when a cyclist brake
- Display easily idientifiable light signals
- Identify the ambient sound frequencies

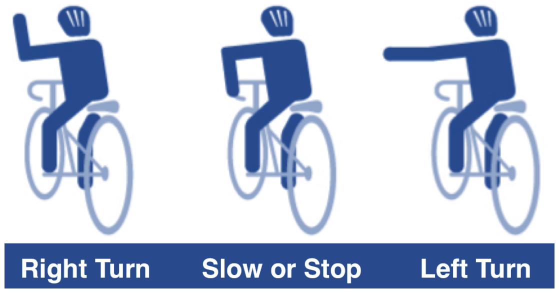

Figure 1: Arm signals

Hardware

Microcontroller

We need a microcontroller that should be as lightweight and as small as possible. But it should also have enough computing power to perform the movement classification (see Software).

We chose the Adafruit Feather-M0 basic proto. Although it is not the smallest microcontroller, it has several features that could help us in the design of our prototype. It has a 32MHz frequency, which we estimated to be enough to run our software. Moreover it integrates nicely with the other components through libraries provided by Adafruit.

LED strips

Adafruit manufactures sewable and non-sewable LED strips. Although sewing the LEDs is the easiest way to attach the LEDs to the jacket, it has one main drawback: the LEDs are not addressable, which significantly limits the number of LEDs. So we chose the NeoPixel LED strips with 60 LEDs per meter, which are not sewable but addressable using PWM. We plan to fix the LED grid to the jacket using velcro.

We created a smaller (4x4) model of our LED grid and were able to program it to create several distinctive indicators that we could use to signal braking, turning, and default states.

Gesture sensors

In order to detect the arm movements of the cyclist, we will use IMUs: one of the left arm, one on the left forearm, one on the back. The two IMUs on the left

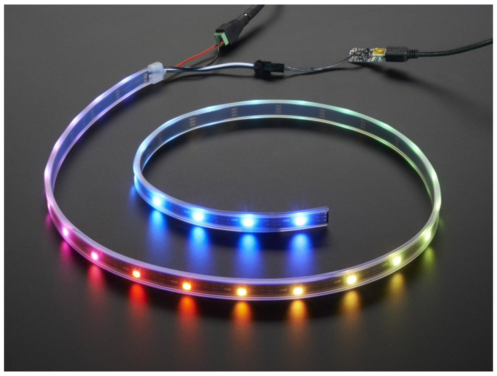
Figure 2: Adafruit NeoPixel

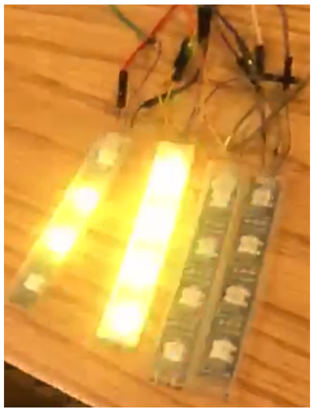
Figure 3: Demo of the signals with a small LED grid

arm will be used to detect the arm movements. The IMU on the back will be used to subtract the global bike acceleration to have the arm acceleration in the bike frame. It will also be used to detect a brake.

For this project, we will use the LSM9DS1 by Adafruit, a 9-DOF IMU. Specifically, we will use its 3-axis accelerometer and gyroscope to measure arm motion and detect hand signals.

Microphone

For the music mode, we will use the Adafruit I2S MEMS Microphone Breakout. Arduino provides a I2S library that is supported by the Feather M0. There are extension to these library that includes FFT to compute the frequencies from the samples given by the microphone.

Software

State Machine

In order to satisfy the requirements specified in the previous sections, we developed a state machine. In the MAIN state, we use the switch signal to determine whether the state machine should switch between the BIKE mode and the MUSIC mode or not, which will further generate LED control signals to the LED matrix. Eventually, we'll output the light signals to the real physical world.

As for the state refinement of the BIKE mode, we have two classifiers, the gesture detector and the main gesture classifier. The former one is used to detect whether the inputs are the gesture signals or just random motions while the second determine which exact gesture it is. The reason for doing so is that this will generate a more accurate classification model because the two classifiers are more specialized than just a general one.

Since brake can happen at any time in BIKE mode, we use parallel composition to model this feature. And it is in the state refinement of FORWARD state that the transitions between turn left, turn right and stop take place.

Gesture classifier

An important part of this project is to detect and classify the arm signals of the cyclist. In order to do this, we used the Scikit-Learn library in Python. We want to use one of the classifier of this library, train it and then implement only the classification algorithm on the microcontroller.

Test data Due to the delay of the equipments, we only managed to get one IMU. We generated data by putting the IMU on our left arm and doing arm signals while standing still. We also did some random movements to distinguish real movements from noisy data. In this case, we are only able to get a 6-dimensional data set (3-axis accelerometer and 3-axis gyroscope) while the final version should be 18-dimensional input data.

We first set a fixed sized sliding window, and a fixed overlapping size. For example, the following figure shows the situation where the sample rate is

5Hz, the window size is 5, and the overlapping size is 4:

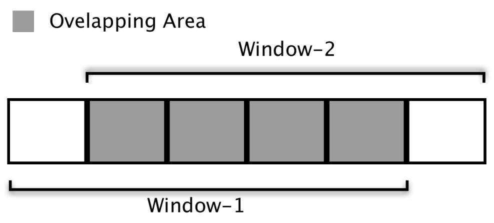

Figure 4: Data overlap

Dimensionality reduction: In order to process the data quickly on the microcontroller, it is useful to reduce the dimensionality of the data. We conducted LDA (Linear Discriminant Analysis) on the raw data and reduce its dimensionality to 2. We did many analysis on our model to develop a robust classifier. We considered PCA (Principal Component Analysis) as a candidate for dimensionality reduction but the result was not very good compared to LDA:

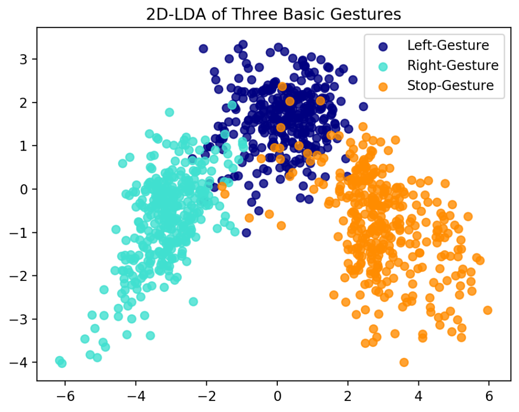

Figure 5: 2D LDA

The reason why PCA can not separate the data from different classes is that PCA looks for dimensions that can explain the overall variance of the whole data set while LDA looks for dimensions that can explain the most significant variance between classes. In other word, LDA takes advantage of the labels on the data while PCA doesn't. In this case, even if we keep the third dimension in PCA, the data set is still not separable:

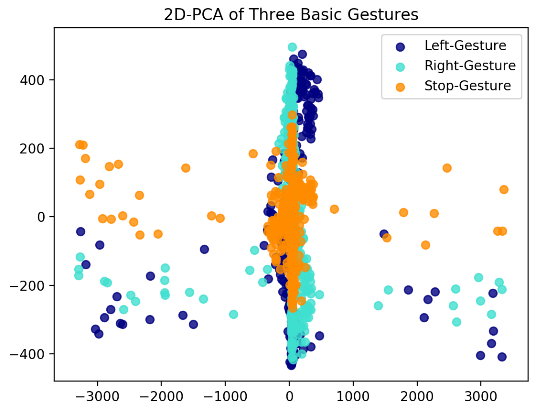

Figure 6: 2D PCA

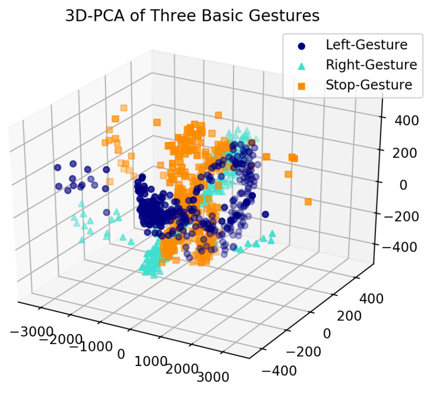

Figure 7: 3D PCA

Classifier: First, we tested a k-NN classifier on the reduced data set. As for the k-NN model, we finally achieved 95%-97% accuracy, which is based on that k equals 13. We divided the whole data set into 10 parts, 5 for training, 3 for tuning and 2 for testing. We change the value of k in k-NN to see the variation of the result:

From this graph we can see that the tuning error is very large though the training error is very small when k's value is around 2, we call this overfitting, can when k is around 10, we got a local minimum of both training error and tuning error, which is our final choice of the value k. Though it doesn't make a big difference to the situations where k 15, considering the computation complexity, the optimal solution

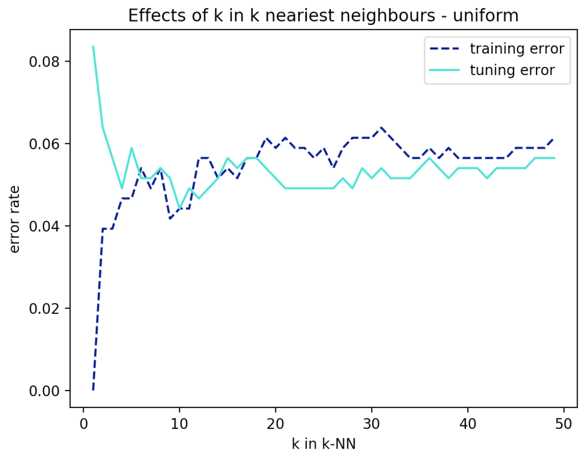

Figure 8: Error with kNN classifier

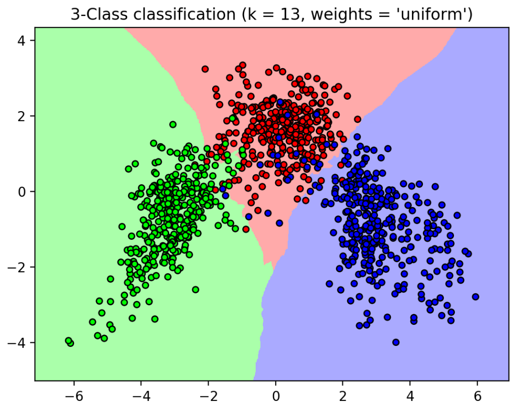

Figure 9: Classification with kNN

still falls in around 10. Following is an overview of our final k-NN model:

We then tried to use a Gaussian Naive-Bayes classifier. The accuracy stabilized at around 95%, which is similar to the k-NN. But the computation and implementation is much easier to realize than k-NN. So this will be the model that we will implement on the real microcontroller.

Simulation

In order to accelerate the integration process between the hardware components and the classifier, we designed a simulation to create signals on a LED grid. We will use it to validate the implementation of our classifier in C before testing it on the microcontroller. We chose to create the simulation with Qt so that we can write code in C that we would reuse for the real program on the microcontroller.

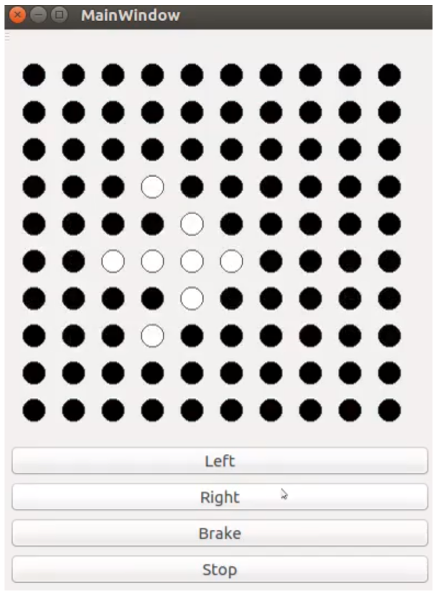

Figure 10: Screenshot of the simulation

Next steps

As said before, our first objective is to code the classifier in C to test it both on the simulation and the microcontroller. Then, we need to train our model with more realistic data that we would get by actually doing the signals on a bike. Now that we received the components, we will be able to generate data with the 3 IMUs.

And of course, we need to build the final product: we need to create the full LED grid integrate every parts and test it on ourselves.