PROGRESS REPORT ON VIPS (formerly known as MIMES)

Period of commitment
March 2018 to June 2018

CONTRIBUTION 1

SYNOPSIS

Development of an alternative gesture recognition algorithm in smartwatch. Used a 1st derivative threshold method on the Z axis of the Gyroscope instead of the X axis of the accelerometer,(as done in Andrea's algorithm).

DETAIL AND ALGORITHM

As decided in the weekly Audio Lab 2 meetings, we settled to implement a simple snare hit gesture, that could **work with any tempo**. So I wanted to analyse and see the key components and patterns the smartwatch sensors go through while the gesture is performed. For this,

- I had recorded 96 samples on the smartwatch. The respective graphs and ARFF files of every sample was classified into 6 configurations-

1.Fast tempo, light intensity	4.Slow Tempo, light intensity
2.Fast tempo, medium intensity	5.Slow tempo, medium intensity
3.Fast tempo, hard intensity	6.Slow tempo, hard intensity

  "Slow" means 55 beats-per-minute
  "Fast" means 220 beats-per-minute.
  "Intensity" means the force applied in the hit, that causes the volume to vary.

  This classification was intended to **encompass all the possible configurations** of a hit being performed. For each configuration, I've sampled 16 recordings, divided into 2 separate recording sessions to avoid bias. Each recording consists of a series of non-stop hits performed, with that specific configuration.

- Studied and noticed the important trends in the various axes of the Gyroscope and Accelerometer, **developed a Behaviour Profile** for analysis (https://drive.google.com/drive/folders/1dM4VyJf1PvIW2Tr9D-ju7AtvGTeytJgq).All the graphs and respective ARFF files are also located here.

  Additionally, the knowledge acquired from the data analysis helped me suggest **a clustering approach to recognise** when a gesture is performed, which could help us base our future machine learning approaches on.
  However, after analysis, Boyd told me to look further beyond for more forms of representations of the graphs and data by breaking it down via fast FFT, or cleaning the signals from noise, after which we could consider an appropriate ML algorithm.

- Based on my findings in the aforesaid Behaviour Profile, among all of the patterns I found, the most useful was the **Z axis of the gyroscope**. It first showed **a trough** (-ve ω) which represented the hand was being raised to perform the hit, and then **a consequent peak**(+ve ω), which indicated the hand moving down to perform the hit. This was of course the smoothest, cleanest and most distinct of all axes, which motivated me to base my algorithm off this as it was so distinguishable every time a hit was performed.

The algorithm has the following key elements -

1. The **time window for evaluation** for a potential hit is set for **only when z gyroscope is positive**, which signifies the hand is now moving downwards and about to make the hit.
2. **A prediction factor of +35**, meaning that if the 1st derivative crossed this value, it is final confirmation that a hit is being performed.
3. Have set **the relaxation time at 170 milliseconds** once a hit is made,

RESULTS

The objective was to make the smartwatch sensitive and accurate enough to able **to recognise every performed beat** and hence be able to **play at all tempos without error**. My approach and algorithm has been able to perform this . The prototype works and can recognise the percussive gestures at almost all tempos.
Mick has fused this code/algorithm of mine with his latency adjustments for the sound reproduction and the final result is displayed in the video as recorded by Mick and I. (https://drive.google.com/file/d/1hsNEWYOG6JwoBQigP7EyNGQCzNiLlYgx/view)

The code to this is located on a secret link on my Github profile (https://gist.github.com/dipster1997/dfcc7d6c82ebeb51fa1ecae38fe6b0a1)

Limitations, Further Motivations & Objectives

- This model does not incorporate volume intensity variation yet and only considers tempo.
- While this works for a simple working prototype/demo, a machine learning approach would make it more robust and flexible in
  1. Learning more gestures and
  2. More accuracy and performing less mistakes in recognition
  3. Can make room for more classifications in a single gesture and hence allow us to measure volume intensity as well in the process.

  So after this, Boyd has advised us to look into some better forms of representation of our data so it will be easier for machine learning algorithms to interpret and learn from.

CONTRIBUTION 2

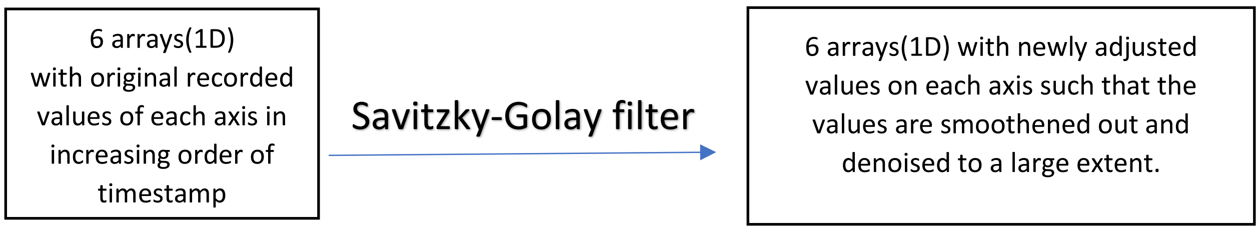
SYNOPSIS

After contribution 1, Boyd asked us to work on better forms of representation, so I began focusing on Digital Signal Processing to see how we could **produce cleaner and smoother signals**. While learning concepts like Convolution, DFT and Fast FFT, I focused a lot on digital filters and implementing them through Python SciPy libraries. While the **moving average filter** seemed a doable option, I was dissatisfied and wanted to find a better one. This is when I came across the already **existent and popular Savitzky-Golay filter**, which would do more justice than the moving average filter. In fact, this IIR (Infinite Impulse Response) smoothing filter is so popular that it is exclusively the only built API in the SciPy library which I been implementing.

The concept of Savitzky-Golay filter is that it **forms a best fit polynomial with least squares** on the data sub-sets provided to it. More information about it can be found in the following link- (https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter)

<u>DETAIL AND ALGORITHM</u>

We consider the recording of the values displayed on 6 axes(3 from accelerometer and 3 from gyroscope) taken during a performed gesture and stored in an ARFF file. The ARFF files are then extracted and placed into 6 1D arrays respectively and the Savitzky-Golay filter is applied **on each 1D array** through the SciPy API to produce a new 1D array **with adjusted values** so that the graph is **removed from a lot of noise**.


6 arrays(1D) with original recorded values of each axis in increasing order of timestamp
Savitzky-Golay filter
6 arrays(1D) with newly adjusted values on each axis such that the values are smoothened out and denoised to a large extent.

The **newly filtered 6 arrays** are now taken and plotted on a graph and displayed to show if there has been any improvement.

The data on which this filter has been tested is **a random combination of 20 hand gestures**, **varying intensity and varying tempo**, recorded on the smart watch, which involves the right hand performing a snare hit. The randomisation is intentional to exhibit that the data can originate from anyone and hence the filter should prove to be independent if it can work on all 20 graphs.

The code to this application is located on my github profile – (https://gist.github.com/dipster1997/f7668db20bd3614c1e0e3a1cf6df4116)
I have passed on this code to Mick for his implementation of it via the Raspberry Pi.

<u>RESULTS</u>

The results over this filter have been really good and can be evidently seen through the comparison graphs I have generated and uploaded based on my experiments(https://drive.google.com/drive/folders/1So4M6nTQGJItWyKX2dhU8RsJ3gwOIpnW)

The Savitzky-Golay filter is a smoothing filter. The following have been observed in my findings:

- It doesn't really cancel a lot of noise. But **it does smoothen out** whatever passes through it, be it noise or a signal. It works excellent on signals with a low frequency. The best example is the **Z axis of the Gyroscope.** Other axes that it has worked well on are the **Gyroscope X axis**, **Accelerometer Y** and **Z axis**. The thing common between all these 4 axes is that they were always **very distinct in their peaks and troughs** even before the filtering. So essentially the filter has made their peaks and troughs **stand out more distinctively**, while smoothening and cleaning most around them. This, of course, is helpful, especially when we are to train machine learning for more accuracy in prediction.

  The Most Promising graph – the graph displaying all axes of the accelerometer(https://drive.google.com/drive/folders/1mBWXRz7B2Hjgvdk-d27lNk3jdWSbo5q2)

  Oher graphs in the provided link - have shown improvement, particularly
    - The graph consisting of only the X and Z axis of the Gyroscope
    - The individual graphs Ay, Az, Gx and Gz inside the "individual Axes" folder.

- For every axis, there are 2 important parameters of

  1) **<u>The Window Size</u>** : I have found, in general, with a good window size, the better smoothening gets. But going too far with that begins to skew and **diminish the size and shape of the curves**. The more we increase the window size beyond a certain cluster of values(7 to 13), the more is **the interference of other points** outside a range we don't like or want involved.

  2)**<u>Order of Polynomial</u>**: This polynomial structuring is where the smoothening happens. I have found, it is wise to keep a low value in the order of the Polynomial. The more the value of the order of the Polynomial, **the more accommodating and flexible** it is to be closer and **fit with all the original points** given in the 1D array. The problem is that, a lot of **those original points constitute noise** which we wanted to eliminate. So it's better to keep a small value and allow the polynomial to reduce its accommodation for noise

  Finding the correct input parameters to get the filter to perform well involved experimentation.
  For the Gyroscope x axis : windows size of 12, polynomial order of 1
  For the Gyroscope y axis : windows size of 7, polynomial order of 1
  For the Gyroscope x axis : windows size of 9, polynomial order of 1
  For all the accelerometer axes: I've implemented a window size of 9 units and order of polynomial of 1.

- For **very high frequency signals**, where peaks and troughs are sharp, the results of it are not very good where the filtered output is either the same as the input or drastically distorted to fit the filter's adjustments.

<u>Considerations, Limitations, Further motivations and objectives</u> -

- We still need to design a filter that can **remove noise more effectively** and not just smoothen.
- This filter involved best-fit polynomial calculations and the method of linear least squares. This means a lot of calculations might **increase time complexity and latency**, especially if done on the Smartwatch, if not the Raspberry Pi. There might be better and efficient filters than this.
- Can we **tweak and redesign our own version of the the Savitzky filter** algorithm so that it is faster and better for the inputs we feed it?

<u>CONTRIBUTION 3</u>

<u>SYNOPSIS</u>

In the beginning, when the MIMES project has officially kicked off, a few things need to be clarified, which I wished to address in my proposed Percussion Model. Because this project was aimed to be a **percussion-based music ensemble**, the purpose of my model was to state which percussion systems are really possible to emulate and which ones are not.
I have to admit, while this model was unexpected, the musician and artistic side in me was very eager to explore this discussion so we could decide and design our final percussion model accordingly.

IN MY PROPOSED MODEL -

All percussion systems have something in common. To produce a beat, they involve **a singlehanded up-down stroke** with the **major movement in a 2D plane**. For every one of them,

- The intensity measurement is same.
- The tempo variation is same.

Hence if we can recognise that singlehanded up-down stroke, we can replicate most percussion instruments like a Snare, Chinese Gong, Hihat, Bass Drum, Cymbal, Toms. We can go for simpler percussion sounds as well, like simple taps from the Bongo or Cajon or the Dholak.
However, there are exceptions -

- Table cannot be emulated (since **it is played by fingers**. VIPS can only play percussion instruments that use the forearm)
- Chimes, Xylophone cannot be emulated. (VIPS can only play instruments that produce **sounds of a single frequency**. These instruments offer different sound frequencies for the same beat)
- A **multiple collection** of percussive instruments (like a drumkit) cannot be emulated as that would require an **actual spatial recognition** as well (the actual relative positioning of the instruments in real physical space in front of the percussionist), which the smartwatch cannot perform. Hence this means, **only 1 instrument** can be emulated at a time **by 1 smartwatch**.
- The smartwatch cannot be worn on foot as the belt of the watch is too short and will not fit everyone. I have tried and personally seen this myself.

Hence, I establish the fact that In Real-Time, VIPS, using only a Samsung smartwatch gear S3, can only emulate percussive instruments that

1. Produce a sound of only a particular sound frequency (the sound frequency can be altered beforehand, like tuning a snare or muffling a bass drum)
2. Involve a **distinct motion from the forearm** at every beat.

One more additional and exceptional percussive instrument we can emulate without a downward stroke is the Tambourine and the Shaker, simply by **twisting our wrist** and **using the gyroscope** to read the angular velocity movements and produce the sound accordingly.

It was exciting to go through a complete comprehensive list of every possible type of percussion instruments and understand their exact functioning through my experience as a musician. I personally feel, this was a great forum where I could combine my skillsets in both Music and Computing to formulate exactly what we were chasing in VIPS.

RESULT AFTER DISCUSSION OVER PROPOSED MODEL

After a thorough discussion with Boyd, Prof Wang Ye, Mick and Tiera, keeping in mind we want to emulate **Drum Circles** and **basic Music Ensembles**, we have finally decided to focus on a downward stroke snare hit since that is the only gesture involved in such activities. This is good because once the snare movement is perfected, we **offer the same percussion via a different component** like a Tom, Bass pedal, Cymbal, Hihat by just **altering the sound we store in the smartwatch's sound library**.
The following questions arise to build MIMES -

- What range of gestures are we meant to recognise when performing in a music ensemble?
  How do we perform bass notes, accentuated notes, snare hits, cymbals?
- What are the important parameters to display when performing in an ensemble?
  Volume? Tempo? Reverberation? Sustain?

After the meeting, we have decided to focus on the following to get a realistic and accurate percussion system -
1. Tempo flexibility(1st priority)
2. Volume intensity (2nd priority)

CONTRIBUTIONS WITH ANDREA (BEFORE THE MIMES KICKOFF) (March to April 2018)

This was when I had just joined the MIMES project and joined Andrea in working with the Smartwatch. As I was new to the Tizen Studio environment, coding apps in embedded systems, this was a phase of intense learning and picking up the essential skills and was probably the most challenging phase for me to cope with a very steep learning curve, especially since it was during the school term and I was juggling 5 core modules with my music career. Nonetheless, I managed to coordinate well with Andrea on the following –

1. **Decided our approach** - Record the sensor data of the accelerometer and gyroscope installed inside the smartwatch. For that, we must activate the sensors and **save them in a file** with the respective timestamp. On completing a gesture we are exploring (like a basic snare hit), extract the file and **plot the coordinates on a graph** and understand the motion and behaviour of it. (intuition says we do this at a maxima/minima curve moment when the derivative is zero for the coordinates, which signifies the moment a particular gesture is done)

2. **First assignment** - Andrea has assigned me the basic job of developing a button application in the smartwatch where I develop a basic button written “record” and then upon demand “stop”. This is a trivial and easy task of me getting acquainted with a system. I managed to create the code to run the basic button changing app through Evas labels and object.

   A challenging difficulty I faced in later advanced tasks was that Tizen studio is difficult to debug and poorly documented in API, which left me stuck at a lot of places. While this made me spend overnight efforts and lose a lot of sleep for solutions that I finally worked out, it made me learn and adapt a lot to the intense learning curve and definitely a more experienced programmer.

3. **Bluetooth Application** - Using the Bluetooth Speaker provided by Prof Wang Ye, Andrea assigns me to develop a Bluetooth connection with the smartwatch via a native app using API. I manage to do the following –

   - A media player which can play any mp3 file
   - Create a Bluetooth socket in the smartwatch
   - Successful discovery of Bluetooth source devices in the smartwatch at the click of a button. The devices are listed and cycled through in an 8 second interval each. Device to be chosen is clicked on via button immediately while displayed.
   - Successful bonding with the chosen device.

But the following problem arises
When I play the music file in the bonded device, the sound still chooses to come out of the smartwatch and refuses to come out of the bonded device speaker. Andrea and I find no solution to the current approach and decide to abandon the Bluetooth API method and instead resort to original Operating System Bluetooth connection.
That finally produces the sound, but still, the Latency remains a challenge.
The code I wrote for these initial developments is here –
(https://gist.github.com/dipster1997/d53a1abc4775c4cd0fdb9783d9038677)