expresSynth

Final Project Report, Analysis and Evaluation

**Tom Holmes**

**Goldsmiths, University of London**

**Year 3 Music Computing**

**1. Introduction**

Despite my love for them, I’ve never not found myself frustrated at the number of controls on most synthesisers. Robert Moog’s original Modular synthesizers have a huge amount of nobs and buttons on them in their most basic form, and depending on configuration, the number could go into the hundreds; Goldsmiths University’s very own Roland 100m Modular has a figure worryingly close to that number. Even today, in the age of software synthesizers, where the possibilities of intelligent and interesting interfaces are essentially endless due to very limited programming and design constraints, today’s most popular current products, Rob Papen’s Albino and GForce’s Oddity, for example, have if anything increased the number of controls. I can’t help but feel that the dizzying array of controls on most synthesisers is frightening to those who are not familiar (and even those who are) and therefore a lot of people will never try to learn the basics of synthesis, music production or performance, and in the case of more experienced users, may be driven to give up before they discover the intricate details of sound synthesis using software applications.

The idea for this project is to create a synthesiser that takes away the dazzling on-screen controls and replaces them with gestures, familiar and simple gestures that will hopefully, through research and smart programming, be intuitive and consequently breakdown the very real barrier between confusing user interfaces and fantastically designed sound engines. It is also to create a synthesizer with a modern interface, less cluttered than most of the top synthesizers that are out there today, and also to create a tool that not only acts a tool in production and sound design, but can teach people what the controls are and what they do to sound, and do so in an enjoyable and straight forward manner. Further, there’s no reason that if the project works, the foundation of the gestural control system could be used with other applications.

I believe that what I am trying to achieve and the techniques required suits the skills that I have learnt on this course and integrates the aspects of software programming, digital signal processing and design that I have found myself both enjoying the most and succeeding at the furthest throughout my three years at Goldsmiths. I have consistently performed well and enjoyed DSP, particularly in creating and modelling analogue synthesis and sound design, while as a fairly hands-on person, adding a physical aspect to the project in the form of a handheld controller gives an opportunity to uses skills and thought processes, physical design and HCI for example, that wouldn’t be required in a purely code based project.

My long term vision is a piece of software that means players can go from a simple starting sound to whatever sound they can imagine, quicker than is currently possible, and in a way that is more enjoyable, more intuitive and more memorable than is currently possible. The technological world is evolving and developing at a rate that is only likely to increase in the future. The world’s biggest technology companies are now more than ever focusing on improving the cohesion and relationship between humans and computers, and having taken a programming degree with heavy focus on music and sound, creating a tool that offers a different way of interacting with and controlling musical software is a logical, interesting and fitting project to undertake.

In short, I believe that the current synthesiser user interfaces are both outdated and an obstacle, their cluttered displays are a hindrance to both creativity and expression and I feel sooner rather than later an alternative method of control needs to be developed.

**2. The Problem Posed**

Compared to more commercial applications, like mobile telephones, laptops and tablets, music production software has fallen behind in the race towards the perfect interface. In 2004 the most advanced telephone interfaces were slow, unintuitive, awkward and confusing. We are now in 2017 and the latest iPhone operating system is so familiar and intelligent to use that most people could probably operate them without looking. Unfortunately, the 2004 Logic Pro 7 interface is worrying similar to the current edition. The same can be said for the almost identical Sculpture synthesizer interfaces, seemingly unchanged despite 13 years of development. Rob Papen’s Albino software synthesizer has hardly changed interface despite 3 renditions. While many of the world’s essential digital devices have seen constant developmental changes and improvements to their human-computer interaction and interface, the same cannot be said for most musical software.

Scrolling down on a phone or laptop used to be the dull process of many button presses, the same could be said for keypresses on a laptop, whereas, nowadays you scroll with a flick of a finger. Increasing the pitch, or changing the waveform on a software synthesiser used to be the press of a button, well, it still is. Would a similar flick of a finger or wave of a hand not be a more modern and natural approach?

My intended approach to this problem is to look into the gestural systems that already exist, both in the musical sphere and outside, in areas such as gaming and television control, and analyse how they work, their successes and failures, and what kind of implementation might realistically work. I intend on building a hardware controller for a piece of software that includes the necessary gesture recognition algorithms along with a comprehensive synthesizer and the accompanying user interface.

Many of the most well-known gestural systems today, such as Microsoft Kinect, utilise the techniques of computer vision in combination with machine learning to achieve the goals, however in my opinion, despite the advantages of using computer vision system, a hardware controller offers a more achievable alternative given I will be able to choose the components and the respective data they’ll provide, rather than dealing with the unpredictability of a real life scene.

In order for it to be a product that can be evaluated to a degree where it can be concluded whether or not it is a genuinely viable alternative to the current available software synths, I intend it to be a prototype, not a finished product, with the focus **not** being on obtaining an extremely high level of accuracy in gesture recognition, or on a fully accomplished hardware controller and outstanding synth engine, but with all of the aspects and completed and combined to a level where it can be trialled and tested, and its intentions obvious enough to allow a view towards future developments and hopefully a final product.

**3. Research**

There are quite a number of fields that I need to consider to ensure I understand why and how products have been successful, or failed in the past and how I might shape the design of all aspects of my project. The key areas I’ve looked into are previously successful examples of and my own favourite implementations of software DSP synthesis and sound design, intuitive and ingenious user interface designs, the actual gestures that are found in existing products, to give me an insight into what is actually possible, the physical controller design, and the general interaction of gestural systems as a whole.

**3.1 Research into Gestural Systems and Controller Design**

**3.1.1 Method of Control – Handheld or Computer Vision?**

When I first began looking into the gestural systems that are commercially available today, the one that really stood out to me, perhaps unsurprisingly, was the Nintendo Wii. The Wii was, and still is an incredibly successful product, selling 101.63 million units since its 2006 release, with over 30 million units alone shipped in 2008 1. Its design hasn’t changed in almost a decade and its continued popularity and market dominance has meant no similar product has been able to take a significant hold in the gestural game sector. Microsoft’s Kinect add-on for their Xbox range of consoles has sold 24 million units since its release 4 years ago; 4 years after the Wii’s release it had shipped 84 million units.

The key difference between the Wii and the Kinect, and why the Wii stands out to me for this project is that where the Kinect uses Computer Vision, a technique that extracts and interprets data from a camera, essentially automating the ability of the human eye in the digital world, the Wii uses a less sophisticated method, utilising a physical accelerometer in combination with standard buttons and infrared communication to send raw acceleration data to its hub, where that data is computed and transformed into on screen reactions.

Computer Vision is fantastic for many application, and in examples that require object recognition, individual identification and facial recognition or image restoration for example, it is absolutely essential, however, in my opinion, for an application that only requires a few values for movement and speed it is too complicated. Using a camera that observes the real world means your data is subject to the randomness and variability of the real world, someone walking past the screen, different people having different hand sizes, brightness differences due to lighting, or different backgrounds behind the user can mean huge variability in data streams.

There are of course problems with the handheld accelerometer method, the key issue being that different users will inevitably move and hold the controller in different ways, giving varying data that needs to ultimately be mapped to the same function. This problem can however be broken down into smaller ones: device orientation, controller speed, gesture size and gesture shape.

Each of these variables can cause gesture recognition to fail if not accounted for, and the combination of more than one unaccounted for could render any gesture algorithm completely useless, however, when individually approached the potentially impact each variable component can be fairly easily neutralised.

There are advantages and disadvantages to both handheld controllers and Computer Vision methods, however for my intents and purposes, and given the simplicity of data required, the method that suits this project best is the Wii method.

**3.1.2 Handheld Controller Design**

The Wii isn’t however the only handheld controller to consider. Another piece of hardware that has received attention in recent years is mi.mu’s gesture controlled gloves, developed and most notably used by electronic artist Imogen Heap. The glove, like the Wiimote (Nintendo’s official name for the actual controller part of its Wii system) uses accelerometers and other data sensors that are then manipulated and mapped to different functions. The idea of a glove for music control doesn’t originate from mi.mu’s and Heap’s 2014 glove, one of the original commercially released examples was from Nintendo who brought out their “Power Glove” in 1989. The Power Glove however was considered a commercial failure, it was too big and uncomfortable for its possibilities4.

Imogen Heap’s glove is still in development stage, 28 years after the release of the Power Glove, showing that this approach to handheld controls is still a long from replicating the commercial viability and success of the Nintendo Wii. My personal opinion on gesture controlled gloves is that they prove too complicated to build and have too much potential to be ugly and uncomfortable. Further, the act of putting on and a taking off a glove can itself alone be quite a prohibitive task.

The design of the controller is very important for a lot of reasons, as Don Norman said in the preface of his seminal *The Design of Everyday things, “the poor design that gives rise to so many of the problems of modern life, especially of modern technology.*” First and foremost, way it is held will influence the orientation of the accelerometer. The amount of edges a controller has impacts its ability to be held in numerous ways and therefore orientations. If the user is capable of holding the controller differently, even in the slightest degree, every time they put it down and pick it up, the values from the inherently noisy accelerometers could be very different. For control devices that are held with two hands, for example Sony or Microsoft’s PS4 and xBox controllers, molding the controller to the shape of the left and right hand respectively means the controller can only be held in one way. The difference between these controllers and the Wiimote is that they aren’t made to be moved around and the Wiimote is only made for single hand usage. The way the combat this is to make the controller a rectangle, a rectangle with buttons only on the top and bottom. Giving a controller 4 right angles means that firstly, it is obvious for a user that they are holding it wrong, secondly, it cannot move around and change angle in the user’s hand, and lastly, it gives a reference of the controller’s orientation with respect to the ground, something that cannot be offered by a circle of a five or six or more sided shape.

The main criticism I have of the Wii remote is that it is quite big. As juvenile as it may sound, I believe that there is a certain magic when using Computer Vision or a very small controller because it appears as though you are simply using your hand to control a computer program. Although this may seem a small point, I believe some of the huge success of Apple Inc. can be put down to that weird feeling technologies like their Trackpad or Touch ID Fingerprint Recognition or Snapchat’s facial recognition give you.

**3.1.3 Gesture Design and Selection**

Another area that I had to consider, an area which is fundamental to success of any gesture based system, is the design and selection of the gestures. In a more obvious way than the design of the controller itself, the gestures chosen significantly influence whether a user enjoys the product, and whether it feels natural, intuitive and psychologically unobtrusive enough to want to use it again. Although there is an element of learning and therefore an understanding and familiarity of the system built up over time, the reason the Wii is so successful is that more often than not, you don’t need any type of instruction to know what gesture you need to perform to execute an action, it simply comes naturally to you. Now I appreciate there is some difference between how naturally the gestural action of hitting a golf ball and changing the release on the resonant filter envelope come to you, however, with a little of the built up understanding and familiarity as mentioned earlier, with careful gesture design and selection, these two very different examples could ultimately invoke the same natural user response.

So, I began to look at the gestures that these systems use and the physical equivalent of what they are depicting so I could then establish what kind of shapes might feel natural and memorable for certain controls of a software synthesizer.

Samsung’s Gestural TV Control Interface uses a wave to wake up the television, I feel this is fairly fitting and obvious, and also feels quite natural. It also uses a grab for when you want to select something on the screen. The interface also incorporates swipe left and swipe right for moving back a page and forward a page respectively. This type of gesture is very common across a lot of platforms, Apple’s iPhone Interface and Macbook Trackpads, as well as most smartphone interfaces utilize this function for moving between pages and screens.

In the Nintendo Wii Sports game, the actions are also just as natural; in the boxing game you jab forward to punch, in the Tennis game you stroke either with you hand in front of you or behind and in bowling you swing underarm.

Given how obvious these actions seem, I concluded the best way to establish which gestures suit which commands most appropriately was to ask a selection of friends to “draw” a command, or word, as it came to their head.

I asked three of my course mates and one individual outside of the course who I knew had never used a synthesizer before so get an understand of the thought processes of people who will both understand the commands and those who have no idea what most of the commands mean. The command/words I chose were Sinewave, Triangle, FX, Tremolo, Increase Speed and Add and Subtract. The results were as follows:

**Sinewave** **–** Fortunately, and to great relief, each of my three course mates as well as the other individual drew a sinewave, from left to right.

**Triangle** **–** Again, as expected everyone drew an equilateral triangle, however the start points did differ between the bottom left and bottom right corners and the top of the triangle.

**FX** **–** 2 of the 4 individuals drew an X, starting from the top left, with one moving from the bottom of the first line upwards to the top of the second line and down from there, while the second moved along the bottom of the shape from the end of the first to the bottom of the second line.

**Tremolo** **–** The overall response to tremolo was to move the controller up and down repeated, some quicker and some slower. The individual who didn’t know what tremolo was, moved it in circles repeatedly.

**Increase** **Speed** **–** Aggressive movements upwards and also horizontally to the right.

**Add** **–** Again, fast movements upwards and to the right.

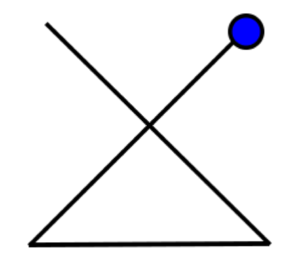
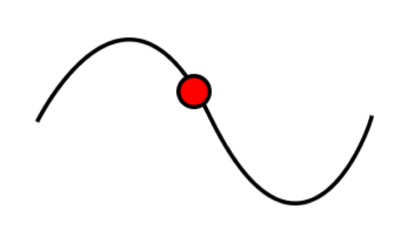
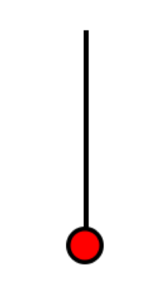
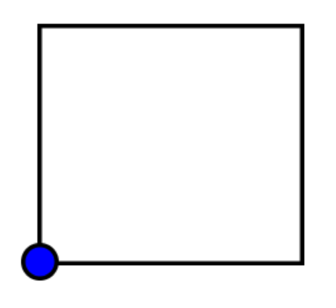
**Subtract** **–** The opposite of increase and add, fast movements downwards and left.

I was quite pleased with the outcome of this experiment as the gestures performed by most of my test subjects were on the whole very similar to what I had thought of myself, thus allowing me to conclude that, at least for people who have used synthesizers before, their idea of what gesture might be appropriate for particular commands and controls are fairly similar.

There were obvious variations in the speed and size of the gestures “drawn”, as well as difference in the number of repetitions in shapes like a sinewave where one individual drew up and down three times while another drew down and up just once.

Another interesting observation from the experiments was that I hadn’t yet considered dexterity and its potential implications; one of my course mates is left handed and unsurprisingly he drew all of the shapes with his left hand.

Following on from the initial gesture experiments I decided to mock up some images and ask again what shapes they thought they should draw. I further expanded these to Gifs that showed when a button might need to be pressed and let go at the beginning (see below, they’re aren’t moving below but when in Gif form they do!). The results of showing these visual commands to people very good, with everyone performing the gesture as I had expected.



**3.1.4 Possible Number of Gestures**

The problem I am trying to combat with this project is user interfaces with tens, if not hundreds of controls. This number can be brought down using improved interface design and better parameter selection, however as am I looking to use either Machine Learning techniques or a differencing algorithm, and given the already apparent difficulties in obtaining accurate gesture recognition using these techniques, bringing this number down to a number that I can realistically consider differentiating between will be difficult.

I looked into existing gestural systems to see how many gestures they use and therefore how many gestures I could realistically consider using:

***Microsoft Kinect –*** *Four main gestures: Wave, Guide (in four directions), Scroll and Hover.*

***Samsung’s Gestural TV Interface –*** *Twelve gestures: Wave, Move, Grab, Long Grab, Grab and Move, Thumbs Up, Two Hand Wave, Two Hand Grab & Narrow, Two Hand Grab & Widen, Two Hand Grab & Rotate, Flip Left and Flip Right[8].*

***Wii –*** *No gesture recognition, but acceleration recognition on the X, Y and Z axes, the Wii uses buttons as selection mechanisms.*

***Apple iPhone Touch Screen and Macbook/iMac Trackpads –*** *An incredible 15 different gestures (Trackpad only, iPhone interface is less due to its inability to detect more than one press at a time)[7].*

As you can see, other than Apple’s implementation on their Macbook and iMac Trackpads the highest number there is twelve, while the Nintendo Wii does facilitate any gesture recognizing actions, just control movements and buttons. From this, and depending on what method I use to distinguish between the maximum number of gestures I presume I can use is between 4 and 6. Therefore I will need to use certain gestures more than once to be able to achieve the level of control I need to achieve to make a comprehensive synthesizer. The other major advantage I identified in the Wiimote is that it uses two buttons to essentially double the amount of gestures you can perform. In Wii boxing “jabbing” forward with the remote while pressing the A button on the top produces a different result than if you perform the same gesture while pressing the B button found on the bottom of the controller.

I also believe I will need to split the gestures into control and selection gestures. Selecting which type of waveform you require is a selection with a discrete number of possible outcomes, increasing the pitch or phase of that oscillator is continuous with a numerical result. The responsiveness of these gestures is also something that needs to be considered when moving between actions with continuous and discrete outcomes. Regardless of what method I chose to recognize the gestures, be it the Euclidean distance or a Hidden Markov Model, it will require analysis and comparison, something that can only occur at the end of a shape. If you want to increase the speed of an LFO, it would feel more natural if the change in speed occurred in union with the change in acceleration of the controller, and thus the time taken to wait for that gesture to finish and then be compared with whatever database, may detract from how natural the gestural control feels.

**3.1.4 User Feedback**

The latest iPhone, the iPhone 7 features haptic feedback on certain commands, the Wiimote has a small motor inside it that provides vibrations and so does the

**3.2 Research into User Interfaces**

**3.2.1 Overall Design and Style**

The main aim of my project is to find an alternative to the cluttered and confusing user interfaces that are found on many of today’s most popular software synths. Graphical User Interface design and Human Computer Interaction are two huge topics of research and the quality of implementation on products, be it mobile phones, ATM’s, desktop computer applications or anything else is key to their success.

Apple’s design principles cite clarity, deference and depth as the three primary themes that differentiate their iOS design from other platforms 4. For clarity they further explain that text is legible at every size, negative space, color, fonts, graphics and interface elements subtly highlight important content, while for depth they say that distinct visual layers convey hierarchy, impart vitality and facilitate understanding 4.

The reason I cite Apple’s design principles is that as an Apple product user I feel their UI design is fantastically clear and understandable, and the above principles, if followed correctly and cleverly should lead to a successful interface. Of the example soft synths I pointed out earlier, Rob Papen’s Albino, G-Force’s Oddity and Apple’s Sculpture, along with hardware alternatives, in particular the modular synths, the Roland 100m for example, it can be said that Apple’s design principles are not adopted in their design. The one that stands out to be the most is layers. Almost all software synthesizers use only one layer and cram all of the controls into that one space. There are very few examples where a menu is available and you can delve in and out of the oscillator or filter sections for example, with that section filling the screen, allowing clear presentation of the available controls. Manipulating virtual knobs and sliders on a screen with a mouse or even with a multi-touch tablet is awkward. While the ‘virtual hardware’ look may be appealing, it’s too fiddly to perform with5, let alone if all of the synths controls are squashed into an 800x600 pixel space.

Regardless of what gestures I end up using, they need to be explained in a way that is unambiguous and clear. The depiction of each gesture and the way that depiction is interpreted is of paramount important to the success of the program. Nintendo and Microsoft both utilize simple, cartoon-like animations to show what action the user must perform. Other programs use real-life video clips while some others use worded descriptions. Given the resounding success of the Wii and Kinect platforms the obvious choice for me to use is the cartoon animations. User interfaces are often based on static presentations, a model ill-suited for conveying change. Consequently, events on the screen frequently startle and confine users. Cartoon animation, in constant is exceedingly successful at engaging its audience; even the most bizarre events are easily comprehended6. Cartoon animations, although they can be as intricate and elaborate as you wish, are most commonly fairly simple, including only the required details to depict what they require. This there, is the route that I feel would best suit my UI design. Additionally, stylistically I feel a UI with cartoon animations is more fun than the serious dark and metallic designs found on most of today’s soft synths.

**3.3 Research into Synth Design and Parameter Selection**

As much as I want this to project to find a new way of controlling a synthesizer, I also want the synthesizer to be a comprehensive and satisfying tool for music production. Through my own use of synthesizers along with some examples that have stood the test of time, the Minimoog for example, I think you can deduce a certain number of features you need to have a synth that can make an array of sounds extensive enough for please most users.

The two synths that I want to design my synth around are the before mentioned Minimoog and also the Roland Jupiter-6. Both of these synthesizers have two Voltage Controller Oscillators, one Voltage Controlled Filter, a single LFO and two Envelopes and develop a wide array of sounds from fairly few controls. Given there are restrictions on the number of parameters that can be used due to the confine of gesture differentiating, as well as the desire to keep the interface clean and concise, these two synthesizers, with their wide array of sounds and relatively small number of controls provide a sound starting point for parameter selection.

The Minimoog is a prime example of where less controls can lead to an enhanced experience, the fixed synthesis method of the Minimoog closed down sonic possiblities but also made the process much less daunting for musicians. The way this was laid out on the Minimoog has become the default design workflow for synthesisers both in hardware and software[9]. It is a synth that has featured of countless hit records since the 1970’s with its distinctive filter sweeps, LFO and monophonic sounds. However despite my admiration for the Minimoog, the reason why problem of cluttered interfaces exists is because not everyone is satisfied with simplicity of the Minimoog.

**4. Implementation**

In this section I will detail and explain how I brought together the initial ideas and desires that I had for this project and the research that I conducted, with the influences that the research had on my decisions in building the synthesizer and controller.

I had to use a number of technologies and processes to achieve the task, with the technologies and processes changing over the course of the project to overcome the problems and constraints that I encountered.

Having spent the majority of my time at Goldsmiths learning and improving my skills in C++, much of the time using the openFrameworks libraries and its affiliated addons, using OF in C++ was the obvious choice of language to complete the bulk of the programming. These following sections document how I constructed the project.

**4.1 Initial Controller Design and Build**

All of my research pointed towards the Nintendo Wiimote as a platform around which to design my controller. The Wiimote’s shape helps to eliminate variability in orientation while its accelerometer driven data capture and infrared connection allow free movement and a stream of simple X, Y, Z values for manipulation.

I chose to use an ADXL 345 Accelerometer along with an Adafruit Bluefruit Feather 32u4 as a starting point for my controller. The Wiimote features an almost identical ADXL 330 Accelerometer that also gives X, Y, Z acceleration values. The Wiimote also incorporates a gyroscope and magnetometer, however looking ahead I didn’t see that these features were essential. If it became apparent a gyroscope and/or magnetometer would really help the project, then I could add them later on. The Adafruit Feather offers ample inputs and outputs as well as a Bluetooth Low Energy Module for wireless connection.

Given I wasn’t certain about buttons, haptics or other connections to the controller I left it as it was, with the ADXL 345 connected to the Adafruit via the SCL and SDA connections, giving me enough to start analyzing the data input and creating the software necessary for gesture recognition.

The ADXL 345 offers 4 acceleration sensitivity settings and the ability to change the data rate. The sensitivity options are 2G, 4G, 8G and 16G, while the available data rates range from 0.1Hz to 3200Hz. I initially chose 2G sensitivity and 50Hz data rate; selecting 2G was arbitrary, subject to later experimentation, while 50Hz was due to Adafruit’s note that rates above 100 Hz will exhibit increased noise. Rates below 6.25 Hz will be more sensitive to temperature variations.[10]

**4.2 Connection and Data Stream**

The first problem to overcome was passing the accelerometer data from the Adafruit into openFrameworks. One of the biggest problems I had with the project was incorporating Bluetooth LE into it and getting the controller to communicate and send data wirelessly.

Apple’s Core Bluetooth documentation and Adafruit’s Bluefruit examples both contain Heart Rate Monitor examples. Having studied the code for both, I learnt about the relationship between Central Managers and Peripherals and how to use UUIDs to identify individual characteristics. I got the Heart Rate Monitor example to function properly, changed the data the Adafruit was sending to the X, Y and Z values and was able to pick up these values in the Heart Rate Monitor example.

However, my inexperience in the Objective-C language meant that after a couple of weeks I was still unable to transfer the accelerometer data into my openFrameworks applications, and I ultimately decided to move to a serial, wired connection for the time being. This is one of the areas I would like to focus on the most once the project is finished, particularly given what I had said earlier about the “magic” of appearing to control a computer with just your hand, and the obvious damage having a thick, black USB cable in between your hand and computer does to this illusion…

The Arduino IDE that the Adafruit is designed to be programmed on, as well as the openFrameworks main library both have built in serial communication functions. After some issues and confusion with how the Arduino IDE and OF functions send and receive information in byte sizes and formats I was able to get a stream of X, Y and Z values into my OF application using the following code.

Arduino Code:

Serial.print(xAccel, 4);

Serial.print(" ");

Serial.print(zAccel, 4);

Serial.println(" ");

OF Code:

for(int i = 0; i < bufferSz; i++) {

buffer[i] = (char)adafruitOne.readByte();

xBuf = strtof(buffer, &pEnd);

yBuf = strtof(pEnd, NULL);

}

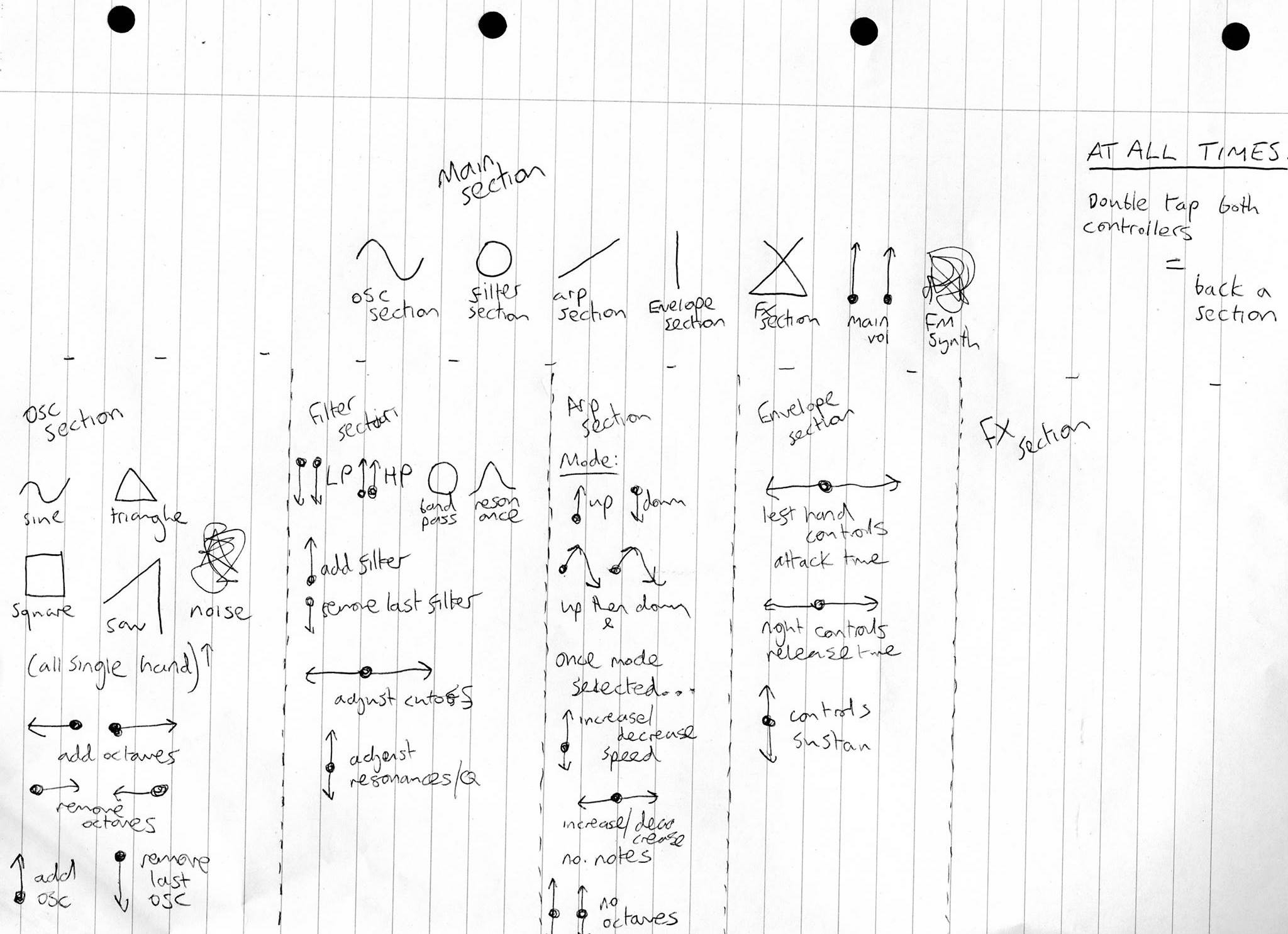
xAccel = xBuf;

yAccel = yBuf;

Having visualized the gestures that I would use, and determining them to all be 2D, I decided against bringing the Z values in, as these values would not be a principle component in identifying gestures and would simply introduce unnecessary variance and increased complexity in calculations.

**4.3 Initial Gesture Selection**

Following on from the research and tests I had conducted, all of which pointed to simply thinking of the most natural looking and sounding gestures, I drew a diagram (referred to from now on as the ***Gesture Tree***!) that depicted the overall structure of the program, breaking it down into sections, along with 11 two dimensional gestures that I had chosen given what they were controlling or selecting and also a rough estimate of their “closeness” based on the expected acceleration values that might come out from those shapes, given the directions they move.



**4.4 Gesture Recognition Method and Data Calibration, Normalization and Comparison**

At this point I had 11 gestures that I had chosen as a basis upon which to build the gesture recognition system and a stream of X and Y values signifying the up/down and side to side movement of the handheld controller. To get the movement of the controller transferred into shapes recognized I needed to first calibrate the data, normalize and scale it into a set amount of readings and then compare the data with a pre-recorded set of data for each gesture that had itself been normalized and scaled.

The issue with calibration was that different people may hold the controller differently. As I mentioned earlier the Wiimote overcomes this by using a four sided case that can be held at one of four possible orientations given how the human hand naturally holds such a shape. The Wiimote also has a gyroscope which I assume it uses to calculate which way it is pointing.

Rather than achieving calibration and consistency in orientation between users by asking them to perform a gesture and them mathematically work out how the controller is currently orientated, I decided that controller shape and instructions as to how the user should hold the controller would achieve the same results if not better.

**4.4.1 Data Storing**

The C++ vector class offers variable size, dynamic arrays of any data type. After creating and recording sets of data in separate X and Y value arrays I found that I could push back one array with pairs of values. Using the keyPressed and keyReleased methods to trigger the start and stop of recording I was able to perform a gesture with the controller while holding down a key, during which each pair of X and Y acceleration values would be pushed back forming a vector of ”that shape”.

shape.push\_back(make\_pair(xAccel, yAccel));

**4.4.2 Normalization**

As the data rate of the Accelerometer is fixed, and the OF application runs at a defined number of times per second, holding down the record key for longer or shorter amounts of time will give different amounts of values in that vector. Given any recognition algorithm will require a standard number of values to compare, normalizing the amount of values that is produced during one record cycle is essential, whether the button be held down for 1 second or 10.

To achieve this, I created a function that will take a vector of any length and return, by linear interpolation, a vector of specified length.

vector< pair<float, float> > Gesture::vectorNormalize(vector< pair<float, float> > vectorIn, int amount) {

int vecInSize = vectorIn.size();

vector< pair<float, float> > vecLerp;

vector< pair<float, float> > finalVec;

for(int i = 0; i < (vecInSize-1); i++) {

float lerpXAmt = lerpDifference(vectorIn[i].first, vectorIn[i+1].first, amount);

float lerpYAmt = lerpDifference(vectorIn[i].second, vectorIn[i+1].second, amount);

for(int j = 0; j < amount; j++) {

vecLerp.push\_back(make\_pair(vectorIn[i].first + (j \* lerpXAmt), vectorIn[i].second + (j \* lerpYAmt)));

}

}

for(int i = 0; i < vecLerp.size(); i++) {

if(i % (vecInSize-1) == 0) {

finalVec.push\_back(make\_pair(vecLerp[i].first, vecLerp[i].second));

}

}

return finalVec;

}

For both X and Y values, the function takes the size of the vector that is passed into it, it then utakes the number of values you require the vector to be and using an interpolation function creates this amount of new points between each point of the original vector. That creates a vector that is the size of both the original and new vectors multiplied together, ie the highest common multiple (granted there will be cases where a lower common multiple is possible, but for complexity purposes and the actually impact on processing time given C++’s capabilities, this will suffice). Now that a vector with the highest common multiple of linearly interpolated pairs of X and Y values is created, the function then iterates through every nth pair, with n being the number of required pairs, to give a vector of the desire size.

I ran some tests where I would draw each of the different shapes while holding down the record key to see what size vectors were being produced, they ranged between 13 and 18. For slightly increased accuracy I chose 20 as the initial number of values I would convert each recording into for comparison.

**4.4.3 Gesture Recognition Method and Comparison**

The four options I considered for the method of gesture recognition were the two machine learning approaches of Hidden Markov Models using the ofxHMM addon and k-Nearest Neighbours using the ofxLearn addon, and the two purely mathematical approaches of calculating the Euclidean Distance between the recorded shapes and finally converting the acceleration values into direction values for all shapes and comparing them.

Given its simplicity, and that I only had two variables to compare, I decided that the Euclidean distance method would be the best place to start. For certain applications Machine Learning is much, much better than other mathematical methods, but like using Computer Vision as compared as an accelerometer over serial, may prove too complicated given the extent of the task.

**4.4.4 Gesture Recognition Algorithm**

After each of the shape vectors have been recorded and normalized, the following algorithm then calculates and totals the difference between the X values of the recorded shape and each of the reference shapes, and then calculates the same value for each Y value, adding that to the same total. This value represents the total difference between the recorded vector and each of the reference vectors, finding the minimum of these differences will then give the shape that the recorded shape is closest to.

int Gesture::select() {

for(int i = 0; i < record.size(); i++) {

sineDifference += abs(record[i].first - sine[i].first) + abs(record[i].second - sine[i].second);

triangleDifference += abs(record[i].first - triangle[i].first) + abs(record[i].second - triangle[i].second);

squareDifference += abs(record[i].first - square[i].first) + abs(record[i].second - square[i].second);

sawDifference += abs(record[i].first - saw[i].first) + abs(record[i].second - saw[i].second);

eightDifference += abs(record[i].first - eight[i].first) + abs(record[i].second - eight[i].second);

}

int returnValue = findMinimum(sineDiff, triangleDiff, squareDiff, sawDiff, eightDiff, 3000000000);

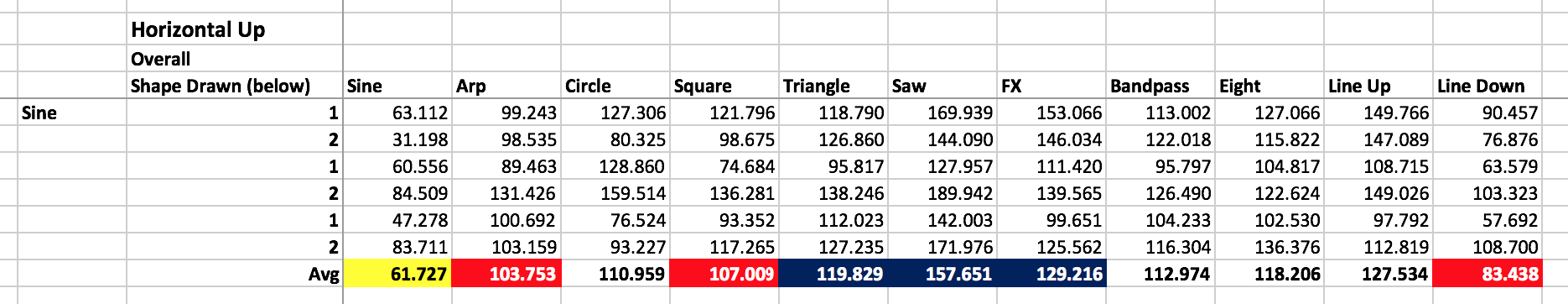
return returnValue;

returnValue = 0;

}

**4.5 Testing and Decision of Final Gestures**

Now that the algorithms are in place to record, normalize, store and compare gestures, I could now look into which shapes were accurately recognized, which shapes were identified incorrect and which shapes had the biggest difference. At this point I also needed to consider calibrations and the difference in how people would hold the controller and how they would interpret the shapes they were told to draw.

I created Gifs of each shape the indicated where each gesture would start, where to move and how fast to move the controller. Once this was done I recorded and stored vectors for all 11 gestures I had chosen to compare and wrote the code that would compare each of the 11 stored gesture vectors with an imcoming gesture, that I could then repeat for all 11 gestures to give difference readings between all 11 shapes and their own pre-recorded vector.

The above table shows the difference results achieved when I performed the sinewave gesture, each column giving the X/Y sum difference between the incoming gesture and each of the recorded gestures. The full table with results for each of the 11 gestures performed can be found in the attached documentation.

The six rows show two attempts for three different calibrations, in order to show how different users and dexterities can affect the difference values and therefore the accuracy of the algorithm. In each of these cases, the 11 pre-recorded shapes were recalibrated by that user. This step of recalibrating for each user is an element I will include in the final program. Having tested with my left hand performing the calibrations and my right hand performing the recordings, the results displayed such inaccuracy that person to person recalibration is the only way to use this method of gesture recognition and achieve any reasonable level of accuracy.

Once I’d averaged the differences for each of the six attempts over three calibrations I looked at the shapes that had both the three furthest distances and the three closest distances to the shape recorded. From this I could take the three closest gestures and establish that they could not be used together at any point in the program. The three furthest could be considered to be used together with the recorded gesture as they were shown to the most different.

Having made changes to my earlier Gesture Tree to incorporate these discoveries, I ran the tests again to see what difference had been made to the accuracy between gestures. The levels of accuracy were still fairly modest, the algorithm choosing the correct gesture around 60-65% of the time.

**4.6 Final Controller Design and Build**

**4.7 User Interface Implementation and Program Structure**

**4.7 Synth Engine Implementation**

**Bibliography**

1 "Wii Sales". *En.wikipedia.org*. N.p., 2017. Web. 7 Apr. 2017.

2 "Wii Remote". *En.wikipedia.org*. N.p., 2017. Web. 7 Apr. 2017.

3 “Gesture Recognition and Interaction with a Glove Controller”, Gerard Llorach Tó, University of Adelaide, Web, July 2013 (http://gerardllorach.weebly.com/uploads/2/6/7/1/26710587/gesturerecognitionandinteractionwithaglovecontroller\_gerardllorach.pdf);

4 "Design Principles - Overview - Ios Human Interface Guidelines". *Developer.apple.com*. N.p., 2017. Web. 7 Apr. 2017.

5 Power, Combining et al. "The Strange Agency Isn’T Afraid To Design A Synth With A Mind-Blowing User Interface". Synthtopia. N.p., 2017. Web. 7 Apr. 2017.

6 “Animation: From Cartoons to the User Interface”, Bay-Wei Chang, David Ungar, Web, 7 Arp. 2017.

7 https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/

8 http://www.samsung.com/ph/smarttv/common/guide\_book\_3p\_si/waving.html

9 http://theconversation.com/sublime-design-the-moog-synthesiser-26460

10 https://learn.adafruit.com/adxl345-digital-accelerometer/library-reference