Digitizing Musical Instruments

Bassam Helal



CSP-301 Project Initial Document

October 2018

# Abstract

Digitally playing real musical instruments has become easier than ever possible, with sample based synthesis leading the forefront in synthesizing the sounds of real instruments. But when an instrument does not have any samples to speak of, synthesizing said instrument’s sound accurately can become a nearly impossible task with today’s synthesis technology. This project aims to solve that by analyzing the desired instrument’s solo musical recording sample and attempting to recreate its sound from scratch. Using the power of the Fourier Transform to decompose an instruments sounds and “crack the code” into how the instrument plays its notes, one can reverse engineer these instruments to bring them new lives into the digital medium. The instruments are thus translated from the analog world into the digital world, to be used by musicians across the world.

[**Abstract**](#_ccbykcb1oxad) **1**

[**1 Introduction**](#_robyw0saeepm) **3**

[**2 Background**](#_1x4zt4dtqul) **4**

[2.1 Fundamentals](#_6zn4iv7i4dqv) 4

[2.2 Digital Sound Synthesis](#_355xuun4v2g0) 6

[**3 Related Work**](#_7xa6pksl9e92) **9**

[**4 Implementation**](#_b5evc5rmhtu6) **11**

[4.1 Input Sample](#_cx95ramd9bze) 11

[4.2 Read Sample](#_oei9qw2dejs3) 12

[4.3 Split Regions](#_dsloxcudp8dy) 13

[4.4 Capture Data](#_8ugvazlppqyv) 14

[4.5 Synthesize Notes](#_682qlpgemx4p) 15

[4.6 Usable Instrument](#_csa5fml6e76y) 16

[**5 Initial Results**](#_r5td09o1xji6) **27**

[**6 Software Development**](#_bwczttt14c2l) **21**

[6.1 Software Development Methodology](#_3dxq341boovg) 21

[6.2 Risks](#_h2d5quo3056t) 21

[6.3 Schedule](#_9z08dblbvg3y) 24

[**7 Evaluation Metrics**](#_ky88le8xxpmb) **26**

[**8 Possible Challenges**](#_la64vnkej9n) **27**

[**9 Summary**](#_yg9kx2t0j14i) **28**

[**Appendix**](#_n4tqzq8469e1) **29**

[**References**](#_xut5sevxu2ta) **30**

# 1 Introduction

Music has existed since the beginning of time when man would blow into flutes made of bone as part of rituals of the tribes. The rapid advancement in technology has allowed for a very large and unprecedented volume of new ways of playing music, to the degree that one may argue that the last 50 years have done more for music than the last 200,000. With the computer came thousands of new digital instruments synthesized from mathematics and code, and with the advent of the internet, these technologies and sounds spread across the world. But what of the natural instruments that preceded these new instruments, have they spread across the world and become more accessible as their digital counterparts have?

As I was listening to new forms of music on YouTube, I stumbled upon the Egyptian Rebab, a spike fiddle used to accompany folk bands and is very recognisable with its distinct sound and playing style. I scoured around the internet to find a form of this instrument that I can use to play on my keyboard, but to no luck. How can so many clips and solos of this instrument exist yet no one has sampled an easily accessible high quality sample of it. I was more frustrated since the sound bytes were so many, enough to nearly sample the instrument myself manually using them. Why does this problem exist at a time where computers are learning to drive cars?

The existing solutions to this are sub-optimal requiring me to buy a rebab of my own, which is very difficult to get a hold of online, not to mention expensive. Synthesizing it never quite gave the same sound, it was close but it was easy to differentiate the real from the synthesized as certain behaviours and frequencies would be missing and often times it would either sound digital or too close to a violin and not have the unique sound that the egyptian rebab is known to have.

Hence I came up with an idea to write an application that can solve this issue. Provide it with a high quality solo of the instrument of choice and the application will return to you the instrument usable as if you sampled it but not containing any of the artifacts or noise from the clip. This resynthesis would effectively be digitizing the instrument, taking it from the analog world and translating it into the digital world.

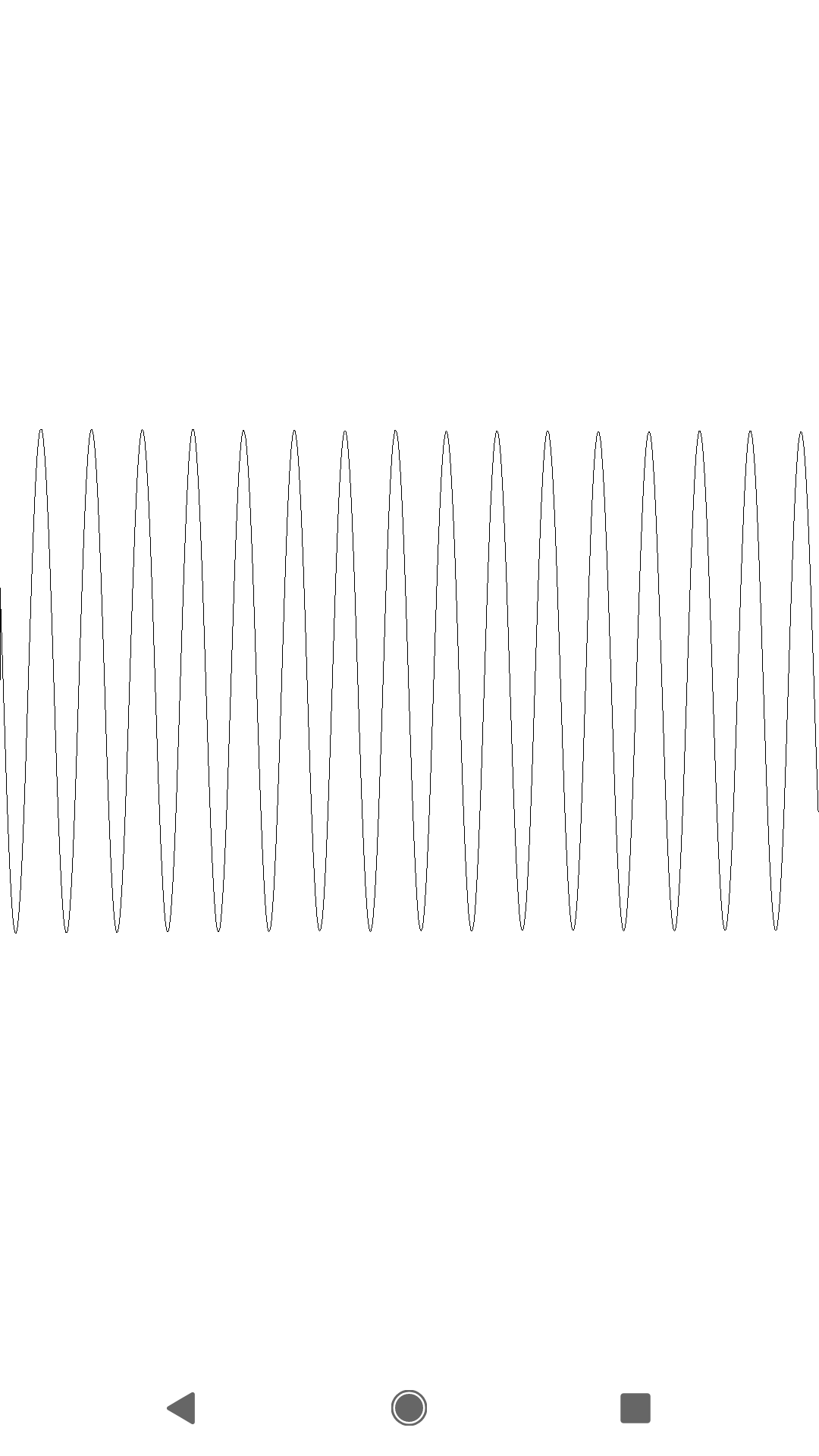
# 2 Background

Firstly we must introduce the reader to some background knowledge that must be acquired in order to understand concepts and ideas brought up later on. In 2.1 we briefly go over the fundamentals such as the basic properties of sound and what differentiates musical sound as well as the concept of timbre in instruments and the use of the Fourier Transform in synthesizing instrument timbre. In 2.2 we explain different sound synthesis methods and their relevance and usefulness to the issue followed by a brief rundown of the technologies that will be mentioned later that this project will use. This section is provided to bring the reader up to speed as we assume very little in the reader’s knowledge of any audio software development.

## 2.1 Fundamentals

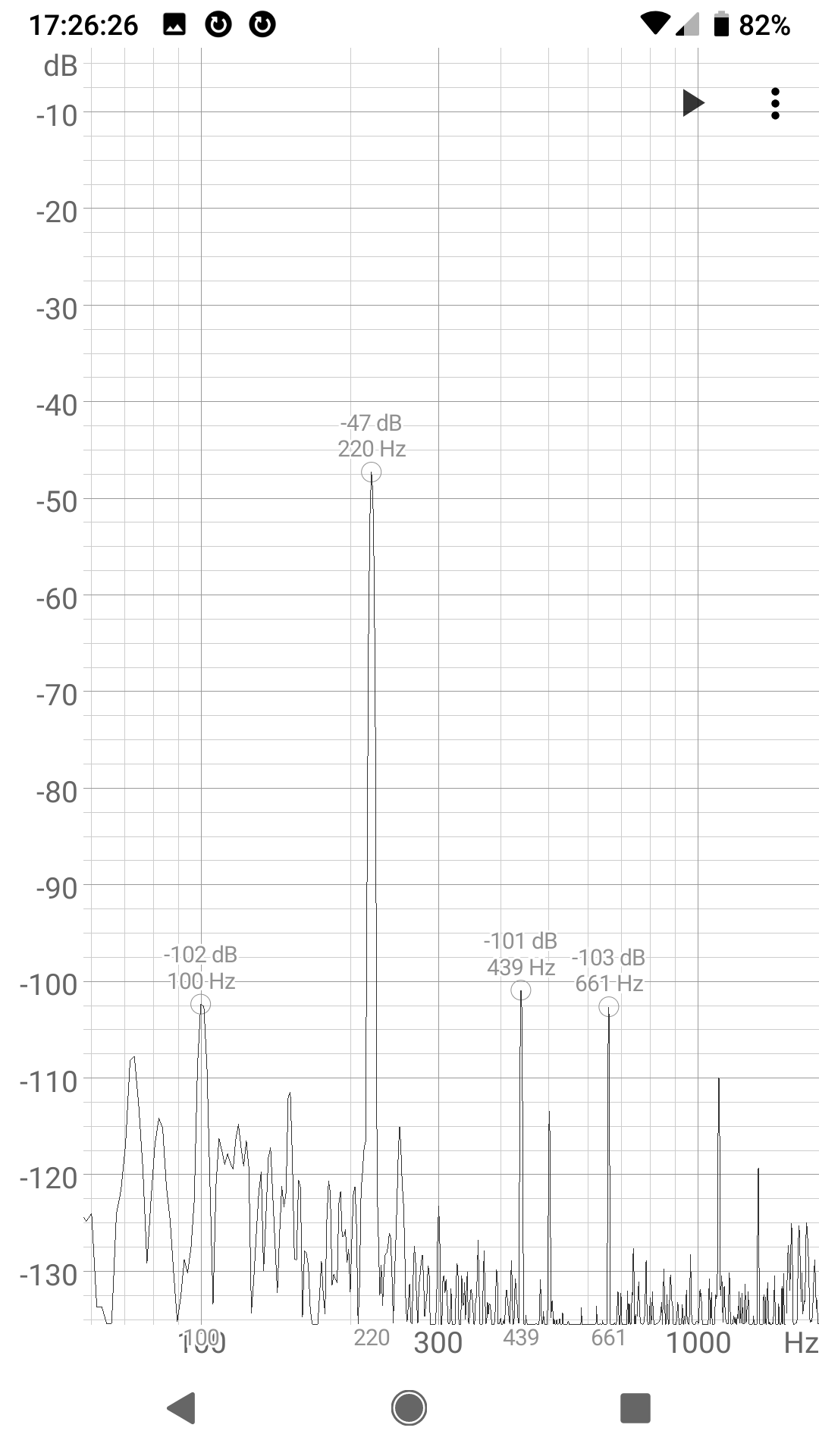
Before introducing any deeper technical knowledge, it is important to understand briefly from a high level view how sound works. Sound is defined by the American National Standards Institute (ANSI) as “Oscillation in pressure propagated in a medium” [1]. Since sound is an oscillation it has frequency, measured in Hertz (Hz) with 1 Hz corresponding to 1 cycle per second. The frequencies heard by humans ranges from around 20 Hz to 20,000 Hz (20 kHz) with the higher limit declining with age. The oscillating wave contains many properties, of which include frequency and amplitude. In the context of instruments, consider a string being plucked on a guitar, the string makes a sound for as long as it is visibly oscillating. The further away it moves when oscillating the louder it sounds, corresponding to amplitude. The faster it vibrates the higher its pitch will sound, corresponding to frequency. Understanding these fundamentals helps us understand musical sound more robustly.

Given how sound works we can now understand what differentiates basic sound from musical sound. When the note A4 is played on a piano, there is not only one frequency being played, rather many frequencies. The frequency which determines the pitch is the lowest one, known as the Fundamental Frequency, it is the most prominent frequency your ear detects and is always the lowest one, in the case of A4 is 440 Hz. The other frequencies are known as overtones, which are always higher than the fundamental [2]. Without these, the sound would be a sine wave at the frequency of the fundamental, which we consider as being a basic sound. It is these additional frequencies that give instruments timbre, a term commonly used among musicians to describe the color of an instrument [3]. It is timbre that allows us to differentiate two or more different instruments despite them playing the same note. Harmonics are overtones that occur at an integer multiple of the fundamental, these are important since it is these that make musical instruments sound more pleasant and musical. The closer the overtones are to being harmonics, the more musical the sound will be, the less harmonic the overtones are the more like noise the sound will resemble. The piano and many other classical instruments have frequencies that are almost entirely harmonics. At A4, 440 Hz (Fundamental) 880 Hz, 1320 Hz, 1760 Hz, 2200 Hz etc. These musical properties form the foundation of any musical sound synthesis and thus will be used and referenced throughout the project.



**Figure 2.1** *Time Domain representation of a 220 Hz sine wave tone*

It is possible to decompose a complex sound, such as that of an instrument like the violin, using what is known as the Fourier Transform into many simple sine waves representing which frequencies are present [4]. Therefore we can reduce a violin’s A3 note (220 Hz) into multiple sine waves with a frequency, each corresponding to a single overtone. This correlates with our understanding of overtones, since the first sine wave’s frequency is the fundamental, the next is the first overtone and so on. An instrument’s sound can have a very large number of overtones, possibly thousands if a high level of accuracy is desired. Audio signals can be represented in 2 domains, the time domain and the frequency domain. The time domain has time on the x axis and amplitude on the y axis, so it represents the change of amplitude over time. The frequency domain has frequency on the x axis and amplitude on the y axis, so it represents the amplitude of each frequency across a range or spectrum. Using the Fourier Transform and its inverse, we can convert an audio signal from one domain to the other and vice versa. The Fourier Transform enables us to identify the timbre of an instrument by the frequencies of its overtones thus allowing us to replicate the timbre.



**Figure 2.2** *Frequency Domain representation of a 220 Hz sine wave tone*

## 2.2 Digital Sound Synthesis

There exist various forms and methods of sound synthesis, which is the term given to define the creation (or synthesis) of sound, often digitally and often for musical purposes, both of which will apply here. However, it is important to note that our interests in this project lie within replicating real instruments’ sounds, by which we mean musical acoustical physical instruments such as the piano, the violin, the flute, the guitar etc. Although they are of use, we do not concern ourselves with synthesized digital instruments that do not occur in the natural world such as the moog, the supersaw, the wobble etc. These sounds have already been perfected and were created digitally already, unlike those which occur in the natural world which have not been directly translated digitally in our belief. This distinction is of importance going forward as one may argue that there is not much to improve with digital instrument synthesis, however, the same cannot be said about real instrument synthesis.

Arguably the simplest and most commonly used form of sound synthesis used to replicate real instruments is sample based synthesis. In this type of synthesis, the synthesizer replays a pre-recorded sample when a note is struck, and performs any further modifications to that sample. Further modifications include any effects such as distortion or reverb as well as when adding notes to play chords since the chords themselves are not pre-recorded. The pre-recorded samples are often done so in a studio with professionals playing every note of the instrument. This is the most widely used form of synthesis to date for replicating real musical instruments for a variety of reasons. Not only is it unmatched in accuracy and quality, it is also extremely computationally efficient, both aspects benefiting greatly from the increase in computational power within the last 4 decades. The main issue with this form however, is its full dependence on many pre-recorded samples done in high end studios requiring a professional and the real instrument itself, something that can be very costly. Nevertheless, sample based synthesis remains the leading form of synthesizing sounds of real musical instruments.

Another few forms of synthesis are used mostly in synthesizers that are not intended to accurately imitate real instruments, and have been arguably more successful at creating new digital sounds as mentioned previously. Additive synthesis is used to create timbre by adding many sine waves, considering that any sound can be decomposed into multiple sine waves as Fourier analysis shows [5]. Subtractive synthesis can complement this by providing a harmonic rich sound (often synthesized with additive synthesis or some other form) then passing it through a series of filters to attenuate certain frequencies. Frequency modulation synthesis was once the industry standard of producing sounds in software and games. It uses a main frequency, called the carrier, to play the sound and another frequency, appropriately called the modulator, to modulate the carrier to create more complex tones and thus timbre [6]. While they may have not been very successful in replicating real instruments, understanding these forms of synthesis will help us realize the many ways sound can be synthesized without a pre-recorded sample.

A more interesting form of sound synthesis that is used more commonly to replicate real instruments is physical modelling synthesis. In a way physical modelling can be thought of as the ray-tracing of the audio world, using mathematics and physics, the properties of the physical instrument are modelled in a mathematical formula to simulate the sound that instrument makes with those properties. The properties can include a very large number of parameters, in a violin for example can include the width of the bow, the speed of movement of the bow, the sound holes (f holes), the location of the bridge and many many others [7]. While physical modelling synthesis can be surprisingly accurate, it still contains its own share of problems, mainly to do with computational efficiency as well as difficulty in providing new sounds accurately. If one were to desire to have an Erhu sound (chinese spike fiddle) they would have to manually manipulate the algorithm with the provided interface until the sound resembles that of the one desired. That is assuming the algorithm allows the changes needed to go from common violin to one with almost completely different properties, and assuming that the common violin synthesis is highly accurate which is debatable. Nevertheless, physical modelling synthesis is a form of synthesis that has the potential to become the synthesis form of choice given further research and development.

Finally we explain VSTs which will be the environment that allows for the application to run, as well as JUCE, the framework that this application will use. VSTs are audio plugins that provide either synthesizers or effects or both to a Digital Audio Workstation (DAW) such as Pro Tools or FL Studio and our application will essentially be a VST synthesizer. Developing a VST instrument from the ground up would be very difficult to do, and for that same code to be able to work on any platform would be impossible for a single person to do, luckily the JUCE framework provides most of that functionality for us. The JUCE framework is a framework used primarily to write audio applications in C++ including VST plugins and synthesizers, it provides a lot of common tools and libraries for audio development and digital signal processing as well as any graphical elements to such an application. More importantly however, is it provides cross platform code, the finished application will be able to compile and run on any environment or operating system. Overall, these technologies are some of the industry standards in developing audio software which we will also be making use of in our project.

# 

# 3 Related Work

Sample based synthesis will be covered more thoroughly in this section as we provide examples of popular uses of it in industry. We then detail why exactly this project’s application is different and solves a specific problem that sample based synthesis in its current form causes.

As stated previously, sample based synthesis is the most popular form of synthesis used to replicate real instruments, and many examples exist to back this up. Professional keyboards that have built in sounds of real instruments such as the KORG keyboards in the PA series and the KRONOS series, all synthesize their sounds using sample based synthesis, then perform any additional modifications to the pre-recorded samples [8]. The same can also be said about their VST synthesizer/ sampler alternatives, such as the leading industry standard Native Instruments KONTAKT which also, when desiring real instruments, will playback pre-recorded samples and perform additional modifications to them when necessary [9]. In both cases the idea is the same, the synthesizer uses some pre-recorded samples stored in the machine’s storage drive or memory and then replays them back when a key is struck, performing any additional modifications to the sample when required (such as reverb or distortion).

The issues that this form of synthesis has mainly arise from a scenario where one would like to have an instrument that hasn’t been sampled before in their digital format of choice. This becomes especially obvious with more “ethnic” or world instruments. In the following examples we will consider the Erhu mentioned previously, a chinese spike fiddle that somewhat resembles the violin but is, for the most part, largely different in many regards. This specific instrument, along with its siblings in the chinese huqin family of instruments and less so the more broader family of spike fiddles such as the rebab, do not have many widely available samples to be used for playing back. When they do exist they are either incomplete (some notes unsampled) or, more commonly, recorded in a lower quality, either because of a lower sampling rate or just a large amount of noise. The current solution to this issue that we have used on our keyboard is to take the closest sound that is already sampled, often a violin, and add synthetic effects to more closely resemble the desired instrument, in this case a voice modulator. While the results are incredible, they are however of an entirely new sound, one that only somewhat resembles the Erhu but does not mimic it directly or accurately enough. Even when the modifications yield a desirable sound, the process can become very cumbersome for only one instrument. This is just one direct consequence of the major issues with sample based synthesis which will be explained further.

The issues with this implementation create a gap in the available technologies that a musician may choose to use to synthesize real musical instruments. The more obvious issue is the heavy dependence on studio quality recordings of most, if not every note of an instrument. This would require us to rent a studio or at the very least have high quality recording equipment to use which is very expensive. The next issue is the instrument itself which would be required to play accurately to record. For many of the instruments in question (world instruments) they are ones less commonly found in the market, they are either expensive or extremely difficult to acquire and even if acquired, would require a specialist in order to play the notes well enough for a recording. All of this creates a huge barrier for entry for someone who would like to play the instrument digitally and essentially creates a large void for such instruments in this form until they are sampled by someone willing to do so and provide the samples to the community, assuming their pricing will be fair and the samples are of a good quality. While sample based synthesis has its many advantages, hence its continued popularity and success, it falls very short in some areas causing a major gap in in the technologies that a musician may use to replicate real instruments.

Our application aims to solve the scenario mentioned above by addressing it directly, it is the main problem to be solved, that so far we believe has not been solved yet. Our application will provide the user with the ability to input a sample of the instrument playing a solo piece and using that, create the entire instrument to be used by the user, nothing is replayed back like in sample based synthesis, rather, all sounds are synthesized. This makes our form of synthesis something in between sample based synthesis and the other forms of synthesis such as additive synthesis. A sample is required but it can be any audio clip, perhaps even one from YouTube, where the instrument is playing a musical piece on its own, of course some constraints will apply which we detail further in the implementation section. However this application will in theory allow for a user to hear a new instrument and not before long be able to play that same instrument without ever having to hold it in their hand, in fact without ever having to leave their computer. This is the reason why this application, regarding this specific problem, is ahead of the competition and provides a novel feature that will be useful to many musicians.

# 

# 4 Implementation

Here we explain, in depth, the planned implementation pipeline of the application, a series depicting each step that occurs within the application. It is important to note that this is subject to change of course and that some details are left behind intentionally mostly since they are very flexible to change. However, the basic structure should still apply and the final application’s pipeline and general workflow should closely resemble this one.

Figure 4.0.1 below shows the basic pipeline of the application, each aspect of this pipeline will contain a dedicated section to further explain in detail the general idea of what is required from each stage. The input sample section describes the input that the application requires from the user, a sample of the instrument playing some musical piece which will be used to extract the data from. The next section details the need for reading the data of the sample which will be important to use when analyzing. The split regions and capture data sections describe the main body of the algorithm, which will, using computational methods such as the Fourier Transform, analyze and split into regions, cyclically, the data provided in order to decompose the instrument’s various provided notes. The following section briefly explains how the notes will be synthesized or “re-synthesized” by using additive synthesis techniques. Finally we explain the output that the user is expected to have after the processing of the algorithm, a usable instrument that resembles that of the sampled one and plays every note, even those not sampled.

**Figure 4.0.1** *Application pipeline*

## 4.1 Input Sample

The application will first be provided an audio sample as an input from the user which will be used as the source of knowledge for the algorithm. The input sample will contain audio of only the desired instrument playing any musical piece, initially these will be very basic but as development furthers on the algorithm should be able to make use of and understand more complex musical pieces.

The sample will be quite constrained, with one of the main constraints being, in addition to the instrument being solo, that the instrument must be played monophonically, meaning only one note can be played at any given moment. Though it may seem that this is a very limiting constraint, the reasoning behind it is very valid, and, moreover, a large majority of solos by many world instruments as well as popular instruments, are monophonic already, as this tends to resemble the human voice fairly well. Think of solos performed using the saxaphone, guitar, violin, flute and even the keytar, all are most often played monophonically, not to mention the vast majority of world instruments, particularly those of the east, do this especially more often as eastern music is more modal than that of the west which is more harmonic. The sample will be a WAV format recording, ideally recorded under good conditions with little noise and a high sampling rate such as 44.1 kHz or higher, however these are not necessary but desired to create a more authentic output. The WAV format is chosen as it very popular and represents the pulse code modulations after each sampled interval, WAV recordings are also used frequently with professional samplers as well. In an ideal scenario, the sample contains a large dynamic range of notes, each of which is played for long enough to identify pitch and be able to discriminate between other notes, and the entire piece is played in one single style (straight violin). However, most likely this will not be the case and there will exist a variety of playing techniques with notes of varying duration, this is something that the algorithm will solve in the split regions and capture data steps among the many other things the algorithm will perform.

## 4.2 Read Sample

Next the sample must be read and understood by the application, as when an audio file is loaded to any audio editing software and after a short processing wait, all the information of the file is visible. This is done by reading the entire audio file including its metadata and providing the information to the user as well as often a visualization of the audio file. The reading of the data, as with most but not all things in this application, will be using the help of the JUCE framework as well as the C++ standard library to read and process files, regardless of file type. This may seem like a trivial insignificant step, but it is of vital importance for the later steps, as they will require constant and rapid access to information from the audio file to ensure the algorithm is both efficient and performant. An efficient reading can make the difference between a fast, correct synthesis and a slow, incorrect synthesis. The reading can also aid in simplifying complex tasks for later steps, such as noise reduction or negative gain. The reading could detect that the quietest sound is actually 0.3 in amplitude, signifying a poor mix (highly compressed dynamic range) or excessive noise, and can quickly reduce the gain or run a quick noise reduction, all before any complex computation is done. This is just one of many possibilities of how the reading sample step is a very important aid for the following steps.

## 4.3 Split Regions

This step and the capture data step will cooperate with one another to efficiently capture all the acoustic information that the sample provides, specifically in this step, the algorithm attempts to split the musical piece into separate regions based on certain properties. The main and most important property being pitch. The piece is divided, if possible, into slices such that each slice corresponds to a single note. The pitch detection will be done by the capture data step by means of Fourier Transform and other more technical data capturing methods which will be used to split the sample into as many useful regions as possible.

Pitch is the most important property as it is quite quick to discriminate between two notes of different pitch, but more importantly because it helps give assumptions about the overall behaviour of the instrument, particularly with overtones. For example, given instrument 𝜷 and its note A3(220 Hz) playing, using Fourier Transform we can quickly find all other frequencies playing at that exact point in time. Knowing that this sample is monophonic and that no other instrument is playing we can assume that only instrument 𝜷’s A3 note is playing. We use the other frequencies currently present with the fundamental to find all overtones louder than some pre-defined amplitude level, lower than which we will consider noise. Assume we have the first 50 overtones of instrument 𝜷’s A3 note, we now have the frequency-amplitude pairs of this note’s overtones, which we can use to deduce the ratios that govern which overtones exist and how loud are they. Hence that when we are required to play B3 we can use the properties taken previously from A3 to synthesize this note, despite it not existing in the sample.

Of course there exist a large number of caveats and exceptions to this method, but the method is a stepping stone towards more advanced techniques. There exist many more possible properties, this only considers the frequency-amplitude pairs of the overtones of a note and does not take into consideration other factors commonly found in instruments such as vibrato, where, even the frequencies and their amplitudes themselves oscillate. Such properties are many and can have behaviour that is non-trivial to record and analyze, but such a thing is not impossible. The splitting regions step works with the data capturing step to split the entire recording into usable regions upon which the algorithm can analyze independently, just as a human would.



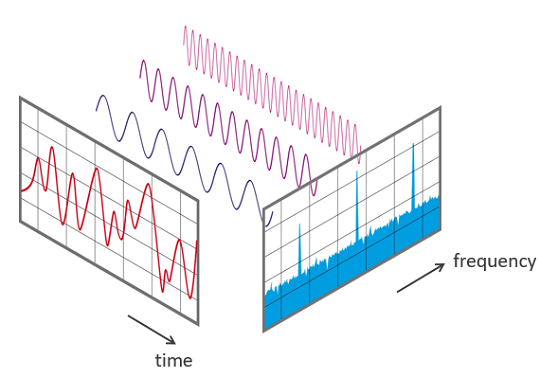
**Figure 4.3.1** *Image-Line Edison detecting pitch regions*

## 4.4 Capture Data

The capturing data step will by far be one of the more complex steps of this entire application and is concerned with capturing and analyzing the data given by the recorded sample. Initially we had intended for this to be simply a Fourier Transform but as initial results have shown (see section 5), there are many more factors that need to be accounted for and thus somehow measured and recorded.

In light of the discovery, we have proposed and planned a new technique to capture and analyze the recorded sample. A continuously recording Fourier Transform which itself will be continuously monitored and analyzed, this is to address issues to do with frequencies and amplitudes of a note oscillating themselves as a sort of vibrato, as well as the issue that some frequencies lose their amplitude earlier or faster than others. This all in all leads to even the frequency-amplitude pairs (which represent the overtones) requiring to be themselves analyzed and capture some data of their own. This is analogous to a second order derivative in mathematics, where even the rate of change has a rate of change. This certainly adds a significant level of complexity to the algorithm, but makes it all the more fun to write, as well as much more efficient in considering more subtle nuances within instruments which makes the algorithm more versatile and capable of dealing with a larger number of possibilities.

An issue that arises from this however, that we have not very heavily considered, is persistence, rather, how this information will be persistently stored on the machine. In most cases one would like to avoid the huge cost and overhead in development of using database systems, particularly those written in the nearly archaic SQL. Thus we think it would be in the best interest to rather stick to some text based system such as JSON or XML, this is an issue that will be of greater concern and understanding later but it is useful to plan ahead all scenarios regardless.



**Figure 4.4.1** *The Fourier Transform is used to translate an audio signal between the time domain and the frequency domain (image courtesy of Wikimedia)*

## 4.5 Synthesize Notes

The final step of the algorithm is, in theory, the simplest of them all, the synthesis of the sounds for each note. We assume it is the simplest of all since it will behave in the same was as existing synthesis methods that are successful at synthesizing non-real instruments, methods such as additive and FM synthesis mentioned in the background section. In this step, we will have all the parameters needed to synthesize every note stored somewhere, including the frequency-amplitude pairs of each overtone as well as the behaviour of each overtone over time among many other parameters. Using these parameters we will simply use additive synthesis to add all overtones together while affecting the behaviour of each one as time passes. Our additive synthesis will also have many Low Frequency Oscillators (LFOs) that will oscillate at very low unhearable frequencies on parameters such as pitch and amplitude to give effects such as vibrato. This is a very common phenomenon in sound synthesis so we doubt there will be many hurdles with this.

The dilemma that we currently face is whether to let all this computation to be done every time a key is struck (a Just In Time approach) or to precompute it thus making it somewhat (but not entirely) similar to sample based synthesis. This decision will need to be made when we have a clearer understanding of the computational efficiency of our additive synthesis and thus whether it is reasonable to compute it just in time for every key press, however such a decision should seem obvious when the knowledge is acquired but is again, useful to consider early on regardless.

## 4.6 Usable Instrument

The end result is a completely usable instrument within the application which responds to MIDI events and plays all notes as if it were an entirely synthesized instrument not using any samples but would have the resemblance of the real instrument as if it were sampled. Doing this would make this synthesis somewhere between sample based and additive synthesis, as a sample is required to understand how to synthesize, but the synthesis is additive and not playing back the pre-recorded sound. This would allow the instrument to play any note, regardless of whether or not it was in the sample, which also effectively extends the range of the instrument. Although such a statement must be made with caution, the rebab was not intended to be played over many octaves as its range is very small, meaning playing it at notes much further from its own could result in an unpleasant sound, but only in theory. In the end, the musician now has a new instrument in their fingertips, one that would have been costly and time-staking to sample is now directly usable in their compositions, all they needed to do was hear it and plug it into our algorithm.

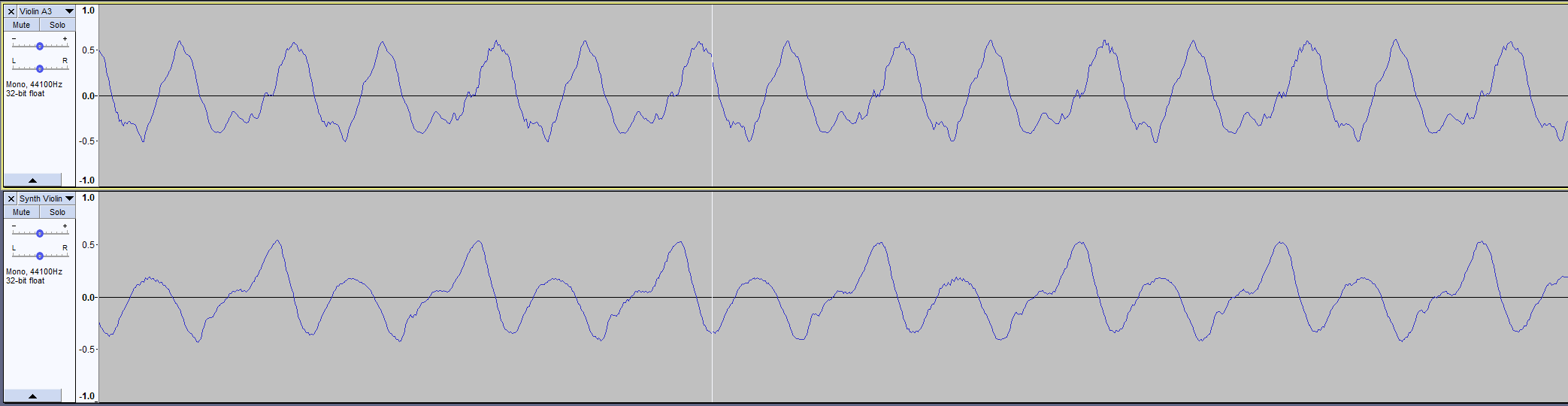
# 

# 5 Initial Results

In the following section we show some initial results and code from testing and playing around with ideas and concepts relating to the application. This was very helpful as it showed where there were aspects that were previously disregarded or unknown. However, it is important to note the crudeness of this process, this was done manually to “emulate” what the final application may do, suffice to say the results were promising but somewhat distant from accuracy.

First we recorded a violin’s A3 (220 Hz) note using our personal smartphone, this violin is a sample played from our personal keyboard (for a list of all equipment used see Appendix) that is a sample from a real violin, note that during the project development a real violin will be used. Using the smartphone we used an application that applies a real time Fast Fourier Transform on the currently playing sound. Then we manually took note of the first 29 overtones of the note at a single instant, with emphasis on single instant as this will explain some anomalies later. Then we quickly coded up a synthesizer that played all 29 overtones with their amplitudes and frequencies. The technology used was not the one proposed to use (JUCE and C++) rather we used JSyn and Kotlin, essentially quickly synthesizing sound in the Java platform.

The results were somewhat positive, the sound is quite close, however upon further inspection using Audacity (see figure 5.1 below) it became clear that there were a few parameters that were not considered. The first being any kind of modulation, the Fourier Transform captured a single instant’s frequencies, whereas when replaying the violin’s sound it is clear there is some mild fluctuation in pitch happening, this may be solved using an LFO (Low Frequency Oscillator) to lightly modulate pitch. The other problem is phase, while the Fourier Transform showed which frequencies were present, it did not show any information about their synchronization with one another, the synthesized sound played all of them in phase, whereas upon further inspection of the violin’s waveform it seemed that there possibly were some frequencies that had different phases which can result in a very different sound.



**Figure 5.1** *Detailed comparison between recorded sound (top) and synthesized sound (bottom) of violin’s A3 note in Audacity*

We personally think that these factors alone are large culprits for the not so accurate sound, however there also exist a couple more possibilities that should be considered. One is noise, from all sources. Noise could come from the microphone as smartphone microphones are not the highest quality, but noise could also come from the keyboard’s speakers, a significant amount of noise can be heard from them when the volume was increased for recording. Another possibility is the Fourier Transform application not being very accurate or having misreadings, it is less likely but not out of the picture. The overtones towards the higher end of the spectrum were much further away negatively from their closest harmonic. While this is an expected feature in real musical instruments, and such differences (10 - 30 Hz) are less noticeable in the mid to high frequency range (3 - 6 kHz) it is still a possibility to consider that the application may be misreading.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Overtone #** | **Frequency (Hz)** | **Closest Harmonic Multiple** | **Closest Harmonic Frequency** | **Difference** |
| 0 | 220 | 1 | 220 | **0** |
| 1 | 440 | 2 | 440 | **0** |
| 2 | 660 | 3 | 660 | **0** |
| 3 | 882 | 4 | 880 | 2 |
| 4 | 1102 | 5 | 1100 | 2 |
| 5 | 1324 | 6 | 1320 | 4 |
| 6 | 1541 | 7 | 1540 | 1 |
| 7 | 1758 | 8 | 1760 | -2 |
| 8 | 1980 | 9 | 1980 | **0** |
| 9 | 2414 | 11 | 2420 | -6 |
| 10 | 2643 | 12 | 2640 | 3 |
| 11 | 2854 | 13 | 2860 | -6 |
| 12 | 3070 | 14 | 3080 | -10 |
| 13 | 3281 | 15 | 3300 | -19 |
| 14 | 3504 | 16 | 3520 | -16 |
| 15 | 3727 | 17 | 3740 | -13 |
| 16 | 3938 | 18 | 3960 | -22 |
| 17 | 4160 | 19 | 4180 | -20 |
| 18 | 4594 | 21 | 4620 | -26 |
| 19 | 4816 | 22 | 4840 | -24 |
| 20 | 5039 | 23 | 5060 | -21 |
| 21 | 5250 | 24 | 5280 | -30 |
| 22 | 5473 | 25 | 5500 | -27 |
| 23 | 5695 | 26 | 5720 | -25 |
| 24 | 5906 | 27 | 5940 | -34 |
| 25 | 6141 | 28 | 6160 | -19 |
| 26 | 6352 | 29 | 6380 | -28 |
| 27 | 6586 | 30 | 6600 | -14 |
| 28 | 6797 | 31 | 6820 | -23 |

**Figure 5.2** *The first 29 harmonics of the violin’s A3 note, note that some harmonics such as 10 are 20 absent*

The main part of the code is shown below. Kotlin is a Java based language with syntax elements from languages such as Java, Haskell, C# and Python, the below code should read quite easily. Note again that this will not be used in the development of the application, this is just to have some sound synthesized quickly in a language we are already familiar with.

fun main(args: Array<String>) {

val synth = JSyn.createSynthesizer()

val lineOut = LineOut() *//used to output sound*

*// one oscillator for each overtone*

val oscillators = getOscillators(overtones.size)

synth.start()

synth.addAll(oscillators)

synth.add(lineOut)

connectAll(lineOut, oscillators)

*/\* plays each overtone in a separate oscillator and sets the frequency and amplitude of each \*/*

oscillators.*forEachIndexed* { index, oscillator ->

oscillator.play(

frequency =

overtones.keys.toList()[index],

amplitude =

*dBToAmplitude*(overtones.values.toList()[index])

)

}

lineOut.start()

*// continues playing sound*

while (lineOut.isEnabled) continue

}

# 6 Software Development

## 6.1 Software Development Methodology

The software development methodology that has been chosen for this project can be loosely defined as Scrum with heavy emphasis on Test Driven Development. These have been chosen as they fit the style of the project as well the abilities and experience of the team. With this project it is important to be agile as new discoveries and changes may need to be made in short notice.

Our methodology takes from Test Driven Development the emphasis on having tests and using them as a way make sure the project is on track and meets requirements as well as adjusting the implementation among other things in order for the tests to pass. We have always been strong proponents of Test Driven Development and this project is no exception to that, through tests only can we ensure everything works as we want it to. In terms of testing frameworks that is still a question, but be sure that we will have a large suite of unit tests, a smaller suite of integration tests and many runs of manual acceptance tests. Not to concede that many manual comparisons between the original instrument and synthesized instrument will be done, which would technically count as testing as its consequences changes the application.

In terms of structure and organization our methodology takes from Scrum. The entire project can be broken down into phases, each of which can be broken down into sprints, each being 2-3 weeks in length with regular progress meetings and updates with both the client and the team. The client will be my supervisor Dr. Tam and myself (I am a client as I myself would like to use this application), and the team will consist solely of myself. The meetings are those that involve myself and Dr. Tam where I shall update him with progress as well as ask for feedback and any advice. The progress updates will be in the form of journal entries, where I plan to write the project’s progress every sprint, this is to help keep on track as well to help when writing the final dissertation as a reflection. From Scrum it also takes the sprint backlog, or the kanban board, which will be used to track tasks per sprint using Trello, a system I am very familiar with and use in my personal life.

## 6.2 Risks

There exist a large number of possible risks, in order to structure them in an appropriate way, we break them down into 2 types, personal and technical. Each risk has a number from 1-5 depicting both the possibility of the risk occurring and its consequentiality on the project, with 1 being the lowest and 5 being the highest. The mitigation strategy is either a prevention tactic, a strategy to take when the risk occurs or both. In the below tables, P represents Possibility and C represents Consequentiality.

Technical Risks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Description** | **P** | **C** | **Mitigation Strategy** |
| Accuracy | Accuracy of the synthesized sound is sub-optimal | 4 | 2 | Consistently test and tweak algorithm and codebase to optimize accuracy throughout project lifecycle |
| Technology | Required technology is more difficult than previously perceived | 2 | 3 | Spend more time learning it by starting early, spend time learning only the essentials and avoid learning any unnecessary parts |
| Dissertation | Dissertation is is too difficult or time consuming | 4 | 5 | Start writing extra early or even during application development to ensure highest quality at lowest effort with plenty of time |
| Data Loss | Loss of data or project progress, whether it be code or any dependent data such as audio files | 1 | 5 | Back up all project data in the cloud, preferably using GitHub but also using Google Drive and even on a separate physical drive |
| Change of Requirements | During development changes to requirements seem more reasonable or desireable | 3 | 2 | Structure sprints and workload in a very agile way such that this has little effect on how the work will be structured |
| Client communication | Communication between client is poor | 1 | 4 | Be extra persistent in continuous communication with client in a polite and professional manner, document all attempts of communication |

Personal Risks

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Description** | **P** | **C** | **Mitigation Strategy** |
| Time | Not finishing the project on time | 3 | 4 | Manage time correctly and make preparations in advance when delays may occur, give buffer time for each sprint |
| Time | Spending too much time on single piece of work, delaying other parts, possibly leading to incomplete project | 2 | 4 | Divide workloads into sprints and follow milestone goals strictly, divide project into modular pieces, if some functionality takes much longer than anticipated including buffer time, skip it and work on another functionality assuming it is working then come back to it later |
| Travel | May need to travel for Visa or to visit family | 3 | 2 | Bring laptop with me and work away from Swansea |
| Stress and Burnout | Work leads to stressing and lack of motivation | 2 | 3 | Take breaks often and not over work oneself, come to supervisor for advice |
| Natural Disaster | Our alien overlords finally decide to invade us and aim directly and only at my laptop | 1 | 5 | Backup all project data and use University computer systems to continue development |

## 

## 6.3 Schedule

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Task** | Sep-18 | | | | Oct-18 | | | | Nov-18 | | | | Dec-18 | | | | Jan-19 | | | | Feb-19 | | | | Mar-19 | | | | Apr-19 | | | | May-19 | | | |
| **Research** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Background research |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Define project |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Learn code libraries and language |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Initial document |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Synthesizer** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Code basic interface sdad |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Produce basic sound |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Responds to MIDI actions |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Polyphonic sound |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Test and tweak |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Algorithm (Single)** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Read recorded sample |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fourier transform |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Synthesize note |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Test and tweak |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Algorithm (Multiple)** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Split regions of recorded sample |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Synthesize given notes |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Synthesize missing notes |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Test and tweak |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **Combination** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Modify UI to accommodate algorithm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Responds to MIDI actions |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Test and tweak |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **University Obligations** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gregynog (end of Jan '19) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Dissertation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Project Fair May '19 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

The above chart shows the proposed schedule of development. There exist 4 milestones each being 2 months apart. Each month is split into 4 equal parts this corresponds roughly to a week. A sprint is approximately 2-3 weeks, notice that most tasks are this length with a few somewhat longer.

The milestones are as follows:

1. (November ‘18): Initial document submitted, background research and learning done and application development begins
2. (January ‘19): Basic synthesizer is completed which reacts to MIDI events and has a graphical user interface, algorithm still missing
3. (March ‘19): Algorithm can produce results with a single note, provided a single note it will completely resynthesize that note accurately
4. (May ‘19): Algorithm can produce results with multiple notes provided multiple notes and algorithm is combined with synthesizer to create final product

For each task a buffer time is given in addition to the time provided here, as some tasks are likely to take longer than expected and other shorter than expected.

# 7 Evaluation Metrics

Evaluation metrics include qualitative and quantitative methods in order to fully evaluate the application’s performance and quality, especially since music is both a scientific and an artistic domain.

Quantitatively will be the technique most commonly used throughout project development in order to improve the accuracy of the synthesized sounds to those of the real ones. Quantitative methods include but are not limited to the following:

1. Direct comparison of synthesized note with given note using inverted polarity in audio editing software
2. Compare both notes using frequency analysis using Parametric Visual EQs to detect where gaps may be, this is similar to Fourier Transform
3. Fourier compare both notes to find differences in frequencies
4. Compare synthesized missing note to the real note that is missing
5. Compare by ADSR region, or compare ignoring ADSR (ADSR is the amplitude envelope)
6. Compare with and without using noise reduction

Qualitative methods will also be considered but will be somewhat less used than their quantitative counterparts, particularly in the initial stages of development as they will mostly be concerning perception. We will ask participants do distinguish which sound is the real one and which is the synthesized one throughout the lifecycle of the development but more so towards the end, this applies to myself as well. Some techniques include:

1. Random guess which sound is real, which is synthesized
2. Play and record phrases using synthesized sound and ask participants which instrument they hear
3. Ask musical experts to notice which is real and which is synthesized
4. Ask participants to give long response on what they hear, what the sound makes them feel, how do they picture it etc
5. Record a piece made entirely from synthesized sounds and ask for responses

These evaluation methods will be performed constantly throughout the project development with the quantitative methods always being performed during development and when the synthesized sound is believed to be of a high enough caliber of accuracy then the qualitative methods will be used more frequently.

# 

# 8 Possible Challenges

There exist a large number of possible challenges that we may face during the development of the project, many of which have been previously mentioned briefly. We will reiterate and introduce some new possible challenges.

The first main set of challenges are those previously mentioned, specifically noise, phase shifts and frequency-amplitude pairs in overtones themselves oscillating. Noise is an issue that is very highly anticipated and expected and will most likely be dealt with noise reduction algorithms, either pre-existing or a little more creative. The issue with noise is that it adds impurities to the sound heard both by the algorithm when analyzing and by the human after the algorithm has synthesized an instrument that has excessive noise, it removes any musical qualities in the instrument and deforms it. Another challenge but one which was only recently even acknowledged is phase shift, which means that as each overtone is playing it may have started oscillating at a different time, yielding some very vastly different results, this issue should not be game-breaking but a solution has not yet been found. The most difficult issue is the oscillating frequency-amplitude pairs which adds a powerful expressive quality to the instrument at the cost of making the algorithm much more complex, this has been explained in detail in section 4.4 but essentially end up requiring much more detailed analysis of events in a single note, more than that which was previously believed which was a simple Fourier Transform.

The next set of possible challenges are ones that have not been mentioned, which include the ADSR envelope and in-recording effects. The ADSR envelope is essentially the envelope that describes a sound’s amplitude change, consider a guitar, when plucked it is at its loudest but continuously fades away. The ADSR envelope causes a problem in terms of analyzing, as different envelopes will happen depending on playing style, what makes things worse is that the envelope is one of the key aspects of an instrument and so cannot be ignored easily. The in-recording effects issue may be a little contrived, essentially, it is somewhat possible that the recorded sample has some effects on the instrument such as the reverb of the room or a delay done in post recording. Such a thing may have drastic consequences on the final result. In the best case the instrument is replayed with its effects from the recording, but in the worst case this potentially confuses the algorithm and creates very strange results.

# 9 Summary

To summarize, we think this application has the potential to fill a gap that current synthesis techniques have failed to address and can be used as a means for us to even more deeply understand the properties that make real instruments so complex and beautiful. A musician will now be able to hear a clip of an instrument they would like to use, and through the power of our complex algorithm in this application, be able to play that same instrument without having to leave their seat without paying a penny. Perhaps best of all is that this project can expose musicians to new instruments they otherwise would never think of playing while at the same time making these instruments live an even longer life and travel into the hands of people in places that never heard anything like that instrument. Audio and music are powerful mediums that we have not been able to use or understand the full potential of, this application may bring us closer to that, even if by an inch, it’s an inch in the right direction.

# Appendix

Applications used:

1. Recording application: <https://play.google.com/store/apps/details?id=com.andrwq.recorder>
2. Time Domain representation application: <https://play.google.com/store/apps/details?id=com.synaptik.soundwave>
3. Frequency Domain representation application: <https://play.google.com/store/apps/details?id=org.intoorbit.spectrum>

Equipment used:

* All recordings were done using Google Pixel (2016) microphone
* Keyboard used is the KORG PA700 (2018) which uses sample based synthesis (PCM samples) to play back real instruments
* Audacity audio editing software used to view and compare sounds: <https://www.audacityteam.org/>

# 

# References

[1] ANSI/ASA S1.1-2013

[2] "Overtone | acoustics", Encyclopedia Britannica, 2018. [Online]. Available: https://www.britannica.com/science/overtone. [Accessed: 21- Oct- 2018].

[3] "Timbre | sound", Encyclopedia Britannica, 2018. [Online]. Available: https://www.britannica.com/science/timbre. [Accessed: 21- Oct- 2018].

[4] R. Bracewell, The Fourier transform and its applications. New York: McGraw-Hill New York, 1986.

[5] G. Reid, "An Introduction To Additive Synthesis", 2000. [Online]. Available: https://www.soundonsound.com/techniques/introduction-additive-synthesis. [Accessed: 21- Oct- 2018].

[6] J. Chowning, The Synthesis of Complex Audio Spectra by Means of

Frequency Modulation, 1973

[7] R. Rabenstein and L. Trautmann, "Digital sound synthesis by physical modelling", in Image and Signal Processing and Analysis, 2001, pp. 12-23.

[8] KORG Pa4x User Manual English, 17th ed. 2018.

[9] KONTAKT 5 Manual English. Native Instruments, 2016.