Violin Tutor

Richard John Hirst

Student Number: 1514367

September 2016

Dissertation submitted in partial fulfilment for the degree of

Master of Science in Information Technology

Computing Science and Mathematics

University of Stirling

Abstract

Learning an instrument like the violin can be expensive due to the high cost of lessons. Even if a student cuts down on these lessons to save money, they can be hindered by a lack of knowledge to tune their instrument correctly, and a lack of knowledge surrounding finger placements for learning scales, something that is crucial to the development of a player.

Current technology does not provide a useful application which is mobile, free, easy to use, this project sought to develop an application which would act as a solution. It did so by taking advantage of smartphone technology, predominately through the widely used Android operating system.

Research into pitch detection methods found the Yin algorithm to be the most suitable in the creation of a fast and accurate tuner. This newly created tuning facility used in conjunction with a diagrammatic representation of scales, meant the creation of an application that provides users with two functions that were previously separate. The Violin Tutor application can provide players the ability to undertake independent study with verification on technique and tuning outside of lesson time. With further development the possibility of grading a user's playing and expansion to incorporate sheet music could be implemented in the future. With more scales added to the application, publication onto the Android play store will be pursued.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this dissertation reports original work by me during my University project except for the following:

- The code "FastYIN" and "PitchDetectionResult" discussed in Section 3.4.1 and implemented in Section 4.1 was obtained from the TarsosDSP library [42] and used in accordance with their license GNU general public license.
- The jTransforms 2.4 libraries discussed in Section 4.1 were obtained from [45] and used in accordance with their license covered under MPL/LGPL/GPL tri-license.
- Android developer site [46] was used in order to use correct documentation to develop the application in Section 4

Signature Richard Hirst

Date 08/09/2016

Acknowledgements

Without the following this project would not have been possible:

- To my mother and father for raising me to provide all they could in order for me to have the ability to learn and experience life.
- To Beccy for supporting me even with my varying moods for the projects duration.
- To my supervisor Leslie Smith for organising my equipment and room for the summer and guidance on the subject
- To fellow classmates Eileen, Michael and Mark and getting through it all
- To the office staff for supplying me with a fan to cope with the summer heat.
- The libraries of TarsosDSP and the yin algorithm which made my project possible
- To music or more specifically the evolution of ears on the human anatomy, in order to recognise waves and transform them into an output that is so pleasurable and makes life worth living.

Table of Contents

List of Figures

1 Introduction

1.1 Background and Context

I first played and subsequently began to learn the violin when I was 8 years old in 1997. My lessons were paid for and booked through the school/county council teaching scheme and took place in school time for the duration of 20-30 minutes. These lessons were charged for in advance for a semester, for which there were 3 semesters per academic year totalling 36 lessons a year.

Over the 12 years of attending these classes I learnt a lot, from the basics in how to play the instrument, to the fundamentals of musical theory and improving my talents as a player through graded examinations.

Much as I benefited from these classes, there were still problems with the format in how I was taught and had negative aspects attached. These included:

a) High cost of lessons: This made long term tuition out of reach for many wishing to undertake learning an instrument and even possibly short term tuition may be too much of a financial burden.

b) Long periods of time between lessons without assistance: lessons were once a week, within this gap between lessons there may be no other interaction to remedy any problems a student may have i.e. playing out of tune, or misinterpreting music/diagrams.

c) When first learning the instrument it was hard to find the correct tuning: without a teacher with a musical training ear to correct you, or a device to monitor playing a student could easily fall into bad playing habits, hindering progression.

Classes were rarely personal tuition, but were usually shared between 2 to 3 students of similar ability. Towards the end of secondary school, I was fortunate enough to be put into my own class and got the equivalent of having private lessons. Many others who were paying the same amount had to share their lesson time. Due to lessons having such a short time duration students may not be receiving the full focus they deserved.

An issue at the time was the lack of readily available or affordable tuning devices; a well-trained ear was used to mimic the note produced by a piano or tuning fork rather than an electronic device used to determine the correct pitch produced by the strings. This could achieve good results but was dependant on the ability of the one doing the tuning, however from personal experience I know students would often be left at a disadvantage over the summer holidays, having no access to tutors or tuning forks.

Another problem whilst I was learning the violin was more to do with the then current level of technology development. With internet use and availability of digital materials both being limited, the only access to useful learning materials and books came from specialist music shops and were often quite costly. Whilst the uses of the internet and resources available have expanded greatly since my childhood, reliable resources are often still expensive.

Technology has advanced significantly and the problems I had of obtaining resources and assistance with my learning could be reduced without the need of more tuition with involvement of human teaching.

1.2 Scope and Objectives

The purpose of this dissertation project is to develop a tool which would be able to straightforwardly assist violin players with their independent practise. The tool format has been chosen to be a smartphone or tablet application (app) and programmed to run on an Android device. This app could be useful in the following scenarios:

a) To allow a user to learn the basics of violin finger positions and practice simple scales independently.

b) Allow a user to tune a violin without the need to access an additional instrument (such as a piano or tuning fork) for tonal reference or having to rely on a teacher or a more skilled player for assistance.

c) To provide a way for a musician to verify whether the scales they are playing are correct, without the assistance of a teacher to correct finger positioning.

The application will not yet be published on the android market in order to further develop and implement findings from user testing. A potential release with these implementations will be occurring at a future date.

1.3 Achievements

1.3.1 Research and Learning New Technologies

New personal skills were learnt through research into different pitch detection methods, XML programming, and how to use the program Android Studio in order to produce an application. These new skills were used in conjunction with existing knowledge in Java programming and music.

1.3.2 Development of Application

An application was developed that works on a range of Android devices. It consists of two main functions. The first function is a tuner designed to determine the pitch of stringed instruments, particularly the violin. The tuner is accurate and faster in the detection of pitch than many current applications on the market.

The second piece of functionality provided is the application provides a diagrammatical picture of a chosen scale from a selection in a drop down menu. With the combination of both functions the user can determine whether or not they are playing the scale in tune, and remedy their finger positioning if necessary, something previously not available.

1.3.3 Further Development

Through personal testing and black box user testing, it was shown the existing functionality could be improved, while new future developments could be implemented such as the ability to grade how well a player performs.

1.4 Overview of Dissertation

In Chapter 1 personal background is explained into why this project is being undertaken. Objectives are set, before outlining personal achievements within the project.

Chapter 2 introduces concepts on music theory with a particular focus on pitch and details about the violin. An analysis on determining the best output and supporting software will be used to produce the Violin Tutor, before a critical analysis on existing technology and applications.

Chapter 3 discusses different methods of pitch detection and determining which one would be of the most use for the project.

Chapter 4 discusses how the implementation based upon the findings of which pitch detection result were used to develop the best possible application. Details are provided to show how specific areas of the application were developed.

Chapter 5 demonstrates the application running on three different mobile devices and rectifying any errors which occur. Black-box testing is undertaken to evaluate the success of the application and derive future improvements.

Chapter 6 gives a summary of the project, an evaluation of the application and how the application could be developed in the future.

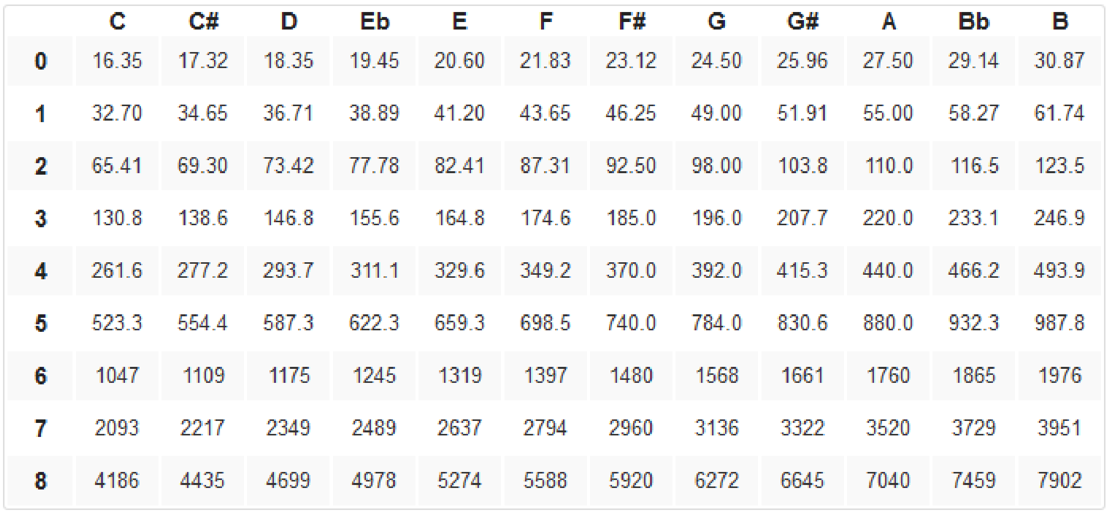
2 State-of-The-Art

2.1 Music Theory

Some music theory terminology is required to understand some basic concepts of how music works. Tonal music is made up of individual notes that each can be defined and distinguished by a range of characteristics, the most important one for this project concerns that of pitch.

Pitch is an audible percept of sound to humans, which may be quantified as a frequency [1]. In music the pitch of an individual note determines how high or low a note is, with the unit of measurement in hertz (Hz). For example, in relation to the violin, the note A4 (A in the 4th octave) produces a value of 440Hz, which is caused by sound waves that vibrate at 440 times a second [2].

In western music these numeric frequency values can be characterised into 12 different semitones, which when grouped together are called an octave and ordered by its hertz value on a frequency related scale(C,C#,D,Eb,E,F,F#,G,G#,A,Bb,B). Notes spanning from C through B can be classed as one octave, of which there is 8 possible musical octaves. A perfect octave is defined as the interval between one musical pitch and another with half or double its frequency [3]. Using the table seen in Figure 1, if A4 (440) is used as a base then an octave down to A3 is half this value (220.0) while A5 is double the base used of A4 for a value of 880.0.



	C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

Figure 1. **Note Frequencies (1)**

These notes are often described in the context of scales. For example - if 12 consecutively pitched notes are grouped together and played in succession; they are called a chromatic scale (Figure 2).

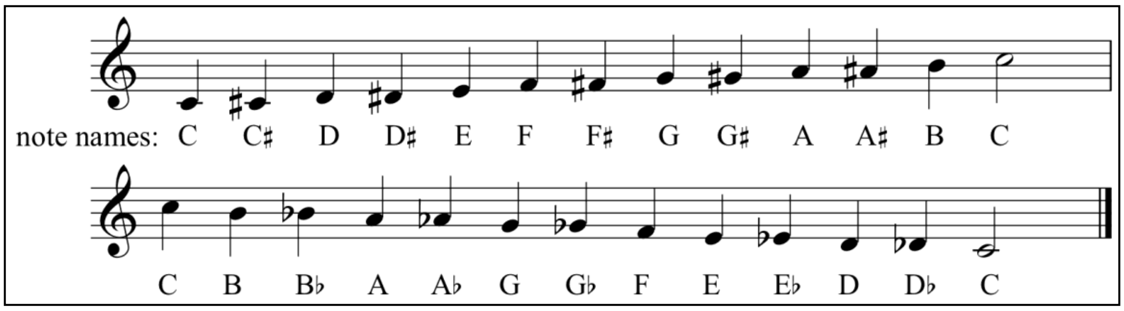

note names: C C♯ D D♯ E F F♯ G G♯ A A♯ B C
C B B♭ A A♭ G G♭ F E E♭ D D♭ C

Figure 2. **12 consecutively pitched notes ascending and descending to form a chromatic scale in C (2)**

Other scales can be made up of fewer notes than the chromatic scale, characterised by the intervals between each note. Major scales are said to sound “happy” while minor scales are considered to sound “darker” [4]. These are just a couple of examples of scales.

Arpeggios follow a different sequence than scales. An arpeggio is a series of increasing or decreasing notes played one after another, forming an outline of a scale at the first, third and fifth note. These notes form a simple chord also known as a triad [5].

Scales and arpeggios are useful for initially learning the instrument. They act as integral elements in the composition of musical pieces. Upon learning these fundamentals, the musician is given skills which can be used to compose new pieces of work.

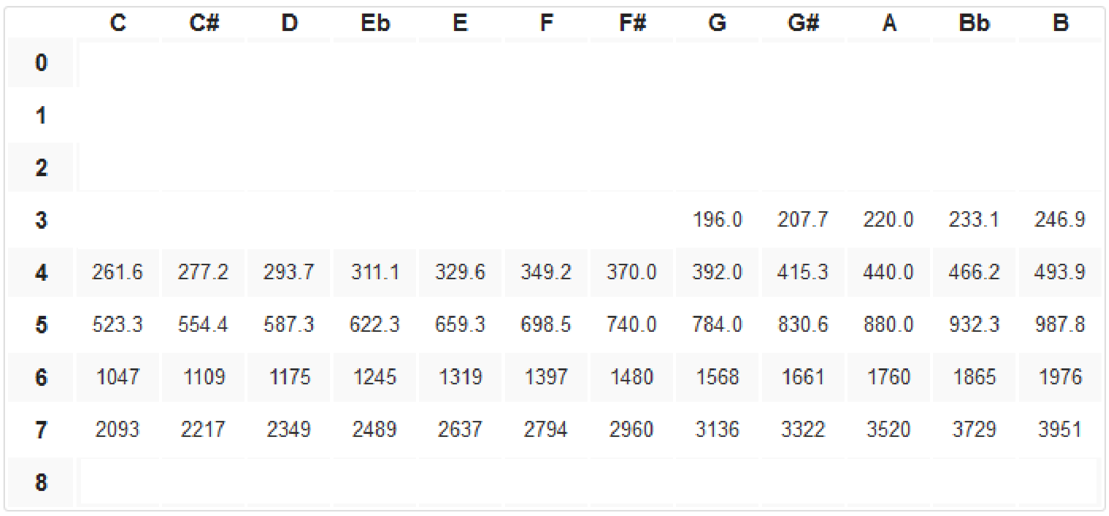
Musical exam boards such as the “Associated Board of the Royal Schools of Music” (ABRSM) have a total of 8 graded examinations, each with its own syllabus for students to practise for and achieve a qualification [6]. Obtaining these qualifications can give an indication in determining the level of skill a player has. In order to obtain said qualification a syllabus sets targets for players to aim for, in turn progressing their own abilities. One key component of this graded examination is the performance of a range of scales and arpeggios at request of the examiner. The higher the graded examination being undertaken, the greater complexity and number are required to be learnt from memory.

Up until grade 4 only major, minor, arpeggio and chromatic scales are needed to be performed for examination purposes [7]. There are many more scales such as dominant and diminished sevenths but they will not be covered in any great depth for the purpose of this dissertation, due to the initial target audience of the application concerning students who are at the early stages of adopting the violin.

Timbre is the character or quality of a musical sound as distinct from its pitch and intensity. Two instruments can be tuned the same, playing the same note yet produce two different sounds. Humans can differentiate the two sounds with relative ease, this however is tricky to do with computers.

2.2 Violin

This dissertation concentrates on the traditional violin. The violin is a stringed instrument with a wooden body, classed as the smallest and highest pitched instrument belonging to the violin family, which also includes instruments such as the viola, cello and double bass. The total range in pitch that can be produced from a violin spans from G3 to B7 (see Figure 3)



	C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B
0												
1												
2												
3								196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8												

Figure 3. **Table showing the total range of notes and corresponding frequencies that can be played on a violin (3)**

The violin (Figure 4) has four strings tuned in perfect fifths. The instruments strings are tuned to G3, D4, A4 and E5. The most common method to produce sound from the instrument is for a bow made up of horse hair and usually wooden material to be drawn across the strings which vibrates the strings producing soundwaves. The instrument can often also be played pizzicato, which is the act of plucking strings with a finger.

The construction of the traditional violin is very distinct. It comprises of a hollow body with "f holes" that amplify the sound produced from the strings. These strings rest on a small indented wooden block known as the bridge which allows the strings to be stretched across the neck of the instrument. The strings are held in place by the tailpiece and the tuning pegs at the top of the instrument. Unlike a guitar, the violin neck is not fretted and precise finger placement is required for correct execution of a note.

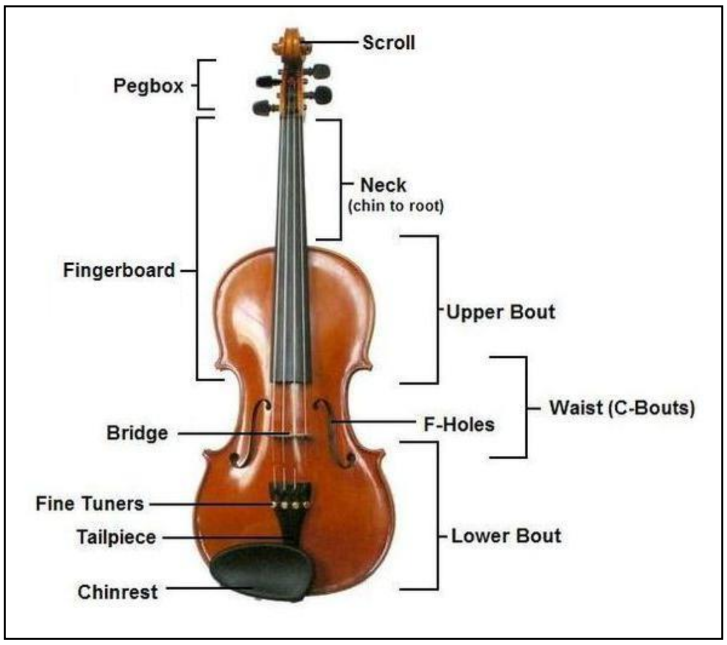


Figure 4. **Technical Description of a Violin (4)**

2.3 Choosing the Electronic Output

Since its inception in 1973 [8] and wider availability from the 80's onwards [9], the handheld mobile telephone has progressed from a large device designed with the sole purpose of making and receiving telephone calls, to becoming a much smaller all-encompassing device with improved processing power, battery life and networking capabilities.

Modern mobile phones, also known as "smartphones" have vastly improved processing power and graphical capabilities which match that of some current entry level PC and laptop counterparts [10]. With this, a range of functionality extending much further than its original purpose has become possible for portable devices.

Mobile devices have become an integral part of people's lives. In October 2014 it was reported that there were more active mobile devices in circulation than people on the planet with 7.2 billion active devices including tablets, smartphones and general mobile phones being used at the moment in time [11]. In the 4th quarter of 2015 alone the smartphone market grew 13.0% with 341.5 million shipments and has seen to be an ever increasing market with a constant demand for more powerful and technical devices [12].

The appeal of smart devices has increased to the point that the modern day user often has their mobile phone as the only computer that they own, rather than the once traditional desktop or laptop computer. A violin tutor could be made as a conventional computer program, but in order to use it many pre requisites would be required such as owning a desktop computer, a pre-amp and a microphone. This method also presents the problem of inflexibility and a lack of portability, due to this equipment having to be setup in a set location for use. Even with a laptop and its better portability the same problem of external equipment is evident, which not only has to be moved but has an initial setup.

The logical route would be to develop a violin tutor that could be used by the largest audience possible, on a format that could be able used anywhere a user would happen to be practising and not limited to one place. Developing a violin tutor for a mobile device would seem to fit these criteria due to how many devices are in circulation and the amount of users that could be targeted.

2.4 Which operating system to use?

Bearing in mind the points made so far in this study, smartphones can be seen as the best platform for which to develop a violin tutor. However there is the question in what operating system to focus on for this project. In recent years there have been two major operating systems that dominate the mobile market: Googles Android system and Apple's "iOS".

Both operating systems are built primarily from two different types of coding language. The official language for Android development is Java [13]. Large parts of Android are written in Java and its application programming interfaces (API's) s are designed to be called primarily from Java. The main programming language for iOS is Swift based on C and Objective-C [14]. Although possible to develop C and C++ apps on android it isn't something actively promoted or encouraged by Google.

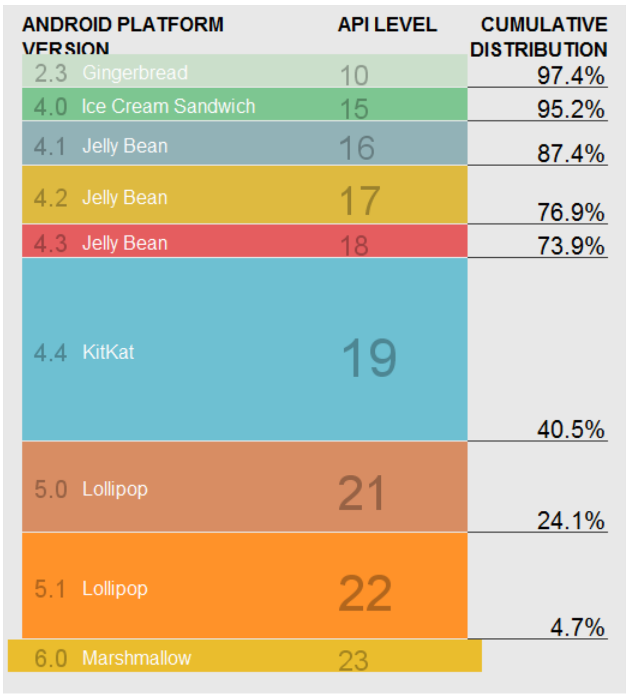
It was reported in 2015 that shipments of Android-based devices has surpassed 1 billion device shipments, with devices running Android accounting for 81% of the market. In comparison Apple accounted for 15% of the market with 193 million units shipped [15]. iOS only appears on Apple products whereas Android is used on a range of different manufacturer's devices, this is most likely the reason for the great difference in market share. This means by developing an app on android, there is a larger potential market for app users on Google Play (Android's app store) than those on the Apple counterpart.

Ideally the violin tutor app would be made for both operating systems to target the largest number of potential stakeholders/app users. Due to the time constraints of just a 3 month project to deliver an application, a choice must be made into which platform to develop for. From market share alone, in order to deliver a violin tutor app to the highest potential possible amount of users, it would be in the interest of the success of the app and project to develop the violin tutor app on Android.

2.5 Program to Develop Application

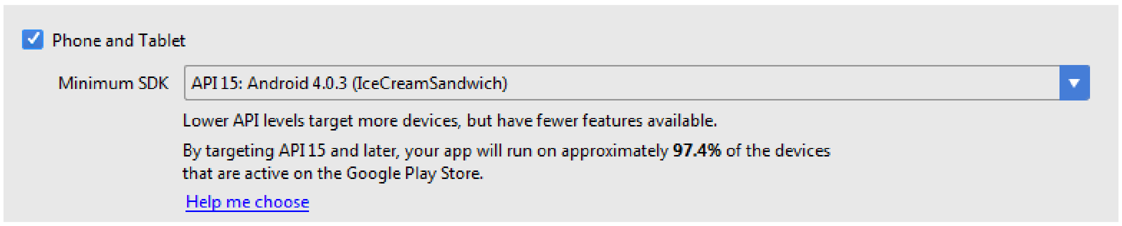
Research into what integrated development environment (IDE) to use to develop an app on for Android, found Android Studio to be the only IDE available, with no other alternatives. In order to develop native android apps, Android Studio simply must be used. Previously options such as using the program Eclipse with a Android development tools (ADT) plugin were possible, however this is no longer supported by Google meaning updates in APK and software revisions make this method incompatible with future alliterations of the operating system [16].

With the application being made for the Android platform, which Android version API needs to be decided. Figure 5 shows a breakdown of percentages which detail what devices are using which version of the android operating system. It can be seen from Figure 6 that by targeting API level 15 (Android 4.0.3) and upwards, a potential 97.3% devices currently in circulation have the ability to run the Violin Tutor application.



ANDROID PLATFORM VERSION		API LEVEL	CUMULATIVE DISTRIBUTION
2.3	Gingerbread	10	97.4%
4.0	Ice Cream Sandwich	15	95.2%
4.1	Jelly Bean	16	87.4%
4.2	Jelly Bean	17	76.9%
4.3	Jelly Bean	18	73.9%
4.4	KitKat	19	40.5%
5.0	Lollipop	21	24.1%
5.1	Lollipop	22	4.7%
6.0	Marshmallow	23	

Figure 5. **Percentage breakdown of devices and which version of android used. (5)**



✓ Phone and Tablet

Minimum SDK: API 15: Android 4.0.3 (IceCreamSandwich)

Lower API levels target more devices, but have fewer features available.

By targeting API 15 and later, your app will run on approximately **97.4%** of the devices that are active on the Google Play Store.

Help me choose

Figure 6. **API level 15 will target 97.4% of devices. (6)**

2.6 Other Programs and Equipment to be used

Audio recording software for a PC will be required in order to record and analyse sounds produced from the violin. Many programs are available to perform such a task, a common audio editing and production programme is Adobe Creation [17]. This programme and others that are similar are very expensive and have lots of features and functionality that will not be used to undertake this dissertation.

Audacity is an open-source free piece of software which allows the ability to record, playback and edit recordings from a microphone, as well as allowing spectrum analysis for signal processing. Due to its functionality and availability it is the perfect choice for this project.

While selecting the correct audio manipulation software for the application's development is important, the correct hardware must also be used in order to achieve the most reliable results in the analysis of sound data. For this a pre-amplifier and a microphone need to be used in order to capture audio recordings.

The pre-amp used is a Steinberg UR22 USB Audio Interface which allows the connection of microphones and uses a sample rate ranging from 44.1 kHz to 192 kHz which is more than enough for the capabilities of an Android application [18].

The microphone to be used is a Rode M3 Cardioid Condenser Microphone. Condenser microphones require power from a battery or external source. The resulting audio signal is stronger signal than that from a dynamic microphone. Condensers also tend to be more sensitive and responsive than dynamics, making them well-suited to capturing subtle nuances in a sound, ideal for accurate analysis.

Similarly, the type of microphone found in most mobile devices is an electret condenser microphone. While the way electret condenser microphones work is slightly different from normal condensers (a condenser requires a battery source where as a electret has a special capacitor which provides a permanent voltage meaning no need for external power) the differences are merely down to the integral of the technology into smart devices and can be considering as working in the same way for the parameters of this application development project [19].

The microphone chosen is referred to as being unidirectional in cardioid with response. Figure 7 illustrates that the microphone picks up sound from one direction, with sound being picked up in front of the microphone. This is known as the cardioid function which prevents rather unwanted sounds from the surrounding area being picked up.

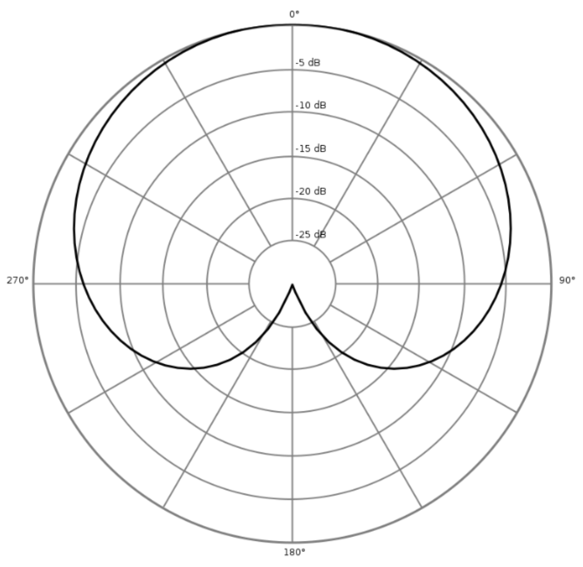


Figure 7. **Illustration showing area that a cardioid microphone picks up (7)**

2.7 Current Technologies and Applications

Before the widespread availability of digital technology, there were limited cost effective options available to a musician to tune their instrument and to learn scales. One of the frequently used options was to tune by ear by comparing pitch with another instrument which was already in tune e.g. a piano. The problem with this method is that not all players have access to an additional instrument. It also works on the assumption that the additional instrument, such as a piano, is itself in tune.

An alternative method could be seen to be a tuning fork; a two pronged steel device which when struck vibrates to the note of a given pitch. However to tune each open string on the violin, a total of four forks would be needed which could be costly. Like the method of tuning against a piano, the precision of the tuning is down to the ability of the player to determine the differences in pitch, and match them so they are the same.

Figure 8 shows two electronic tuners. The one on the left requires the dial to be set to the note a user is wishing to identify, if the string resonates at the desired frequency a light will turn green in order to notify the user that it is tune. Although useful, this device only indicates whether or not a tuner is in or out of tune and not by how much. The tuner on the left is a modern tuner obtainable today at many music shops. Unlike the previous tuner this actively “listens” for a note to be played i.e. it does not need to be set to listen out for a frequency/note to analyse. The display will show not only the note being played but how close to it they are. The problem with both of these tuners is that, while they function well as tuners, they only have one purpose and also requires a musician to carry around an additional item. This can be annoying for a musician who is for example, learning at school and needs to carry school equipment, their musical instrument, as well as additional items.

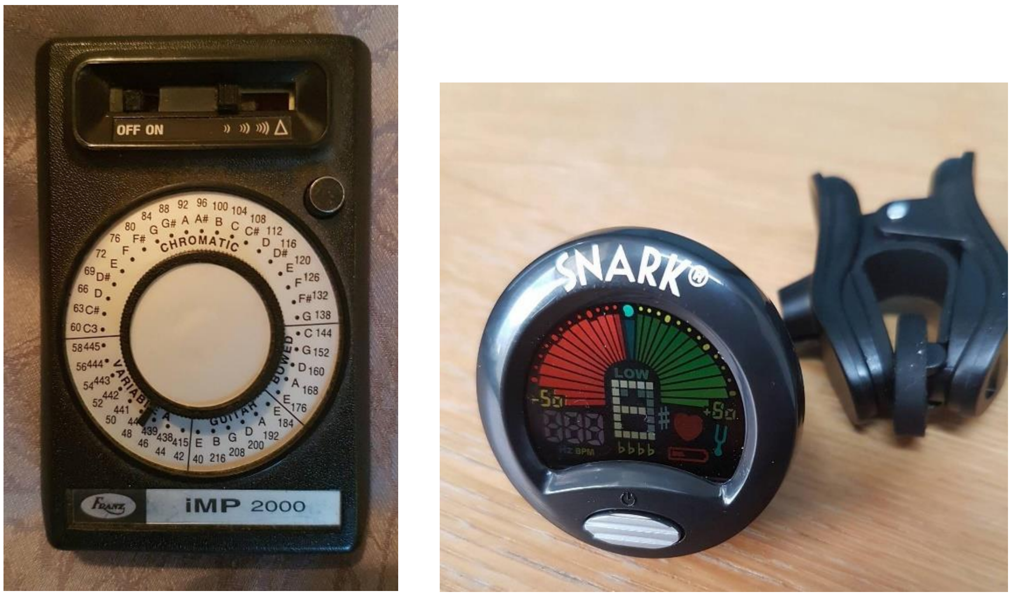


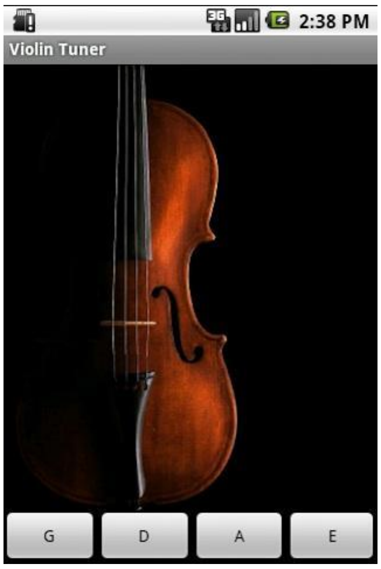
Figure 8. **Two different electronic tuners (8)**

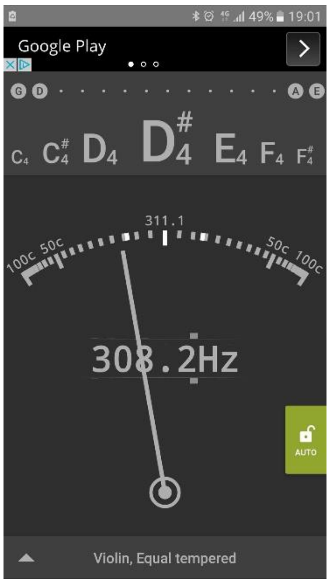
In terms of scales the only obtainable information currently is either what was taught in a violin lesson, or within a scale book which must be purchased or borrowed. The problem of the scale book is that they are usually written in sheet music notation, which for a beginner may not be of much use when the placement of fingers on the fingerboard is unknown.

With a massive development in digital technology in the past 15 years there are now a range of applications that are currently available on the android app market. There are many tuners that are available that have the same functionality as having a digital tuning fork such as an app called “Violin Tuner” shown in Figure 9 [20], where a sound is produced and the user has to match the sound by ear. Although this reduces the cost of purchasing forks and space required to store them, the problem of tuning by ear remains.

There are applications that utilise the microphone in order to perform a tuning functionality, which in turn remove the need a well-trained musical ear. One of the leading free applications with over 200,000 ratings is called “gStrings” (Figure 10) [21]. Using the microphone as an input, the app displays the current note being performed, the frequency of the produced note, and a gauge to determine precisely where the produced note is lying. The results produced by the application proved to be accurate, which was tested by playing with another store bought tuner as a reference. The method of determining the produced note without further user interaction is advantageous over other applications which require further button presses, as it allows the user to focus on playing the violin. However in using this application a substantial delay was apparent for detection of a frequency. This suggests that perhaps the app works by recording a short sample of sound and calculating where the recording lies.

An application specific to use of a violin is called “Easy Violin – Violin Tuner” [22]. The app works on a similar principle, but has limited functionality with a string selection having to be made prior to tuning (similar to “Violin Tuner”), and only having four possible notes to tune against (open tuned strings). This is not flexible when apps such as “gStrings” will determine any note being played.





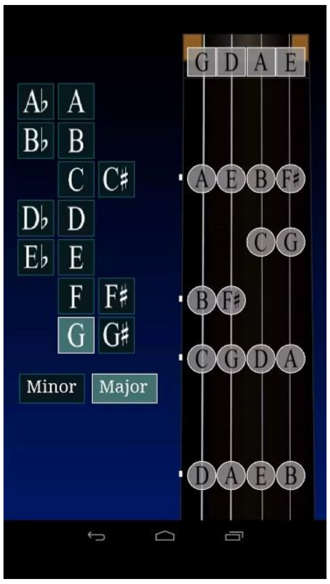


Figure 9. **(Left) – “Violin Tuner” Application Screenshot (9)**

Figure 10. **(Center) – “gStrings” Application Screenshot (10)**

Figure 11. **(Right) – “Violin Scales” Application Screenshot (11)**

Figure 11 demonstrates an application called “Violin Scales” [23] which has a database detailing the finger positioning for various major and minor scales. Touching a note on the screen plays back what the note should sound like, again similar to a tuning fork. The diagrammatical view of displaying a scale is a great representation for learning opposed to sheet music; this method allows a user to see clearly where placement of fingers should occur in order to correctly produce a scale.

An application which tries to incorporate an active “listening” tuner element is called “Scales Practice” [24], which has received around 2,000 reviews and a 4 star rating from users. This app shows scales in sheet music notation form and has the feature of being able to check the pitch. However the issue from personal use and reviews arose with determining pitch produced from a violin, the application had issues with the analysis of the pitch and it was not always accurate. This could be detrimental to a musician at the start of their studies.

Expanding outside the app market there is one prominent piece of software which is a successful instrument tutor. “Rocksmith” is a video game available on PC/Xbox 360 and PS3, it uses a USB to input jack cable, as a means to connect to either a bass or electric guitar [25]. Through the cable the video game provides the functionality of initially tuning the instrument. The main feature gives players the ability to play along to pre-recorded songs with on screen notation. When the user plays along the software notifies the user if they have correctly played the note, this is done by determining whether the input from the user was placed at the correct place of the fretboard and played at the correct time interval. At the end of the song a percentage is delivered in how many successful notes have been played and what areas need improvement. Although the desire to develop an application of this magnitude would be desirable, given a short time period and being a sole development means this would be not realistic or feasible.

Having seen what is available on the Android market, there appears to be a lack of applications that possess the functionality of being both a tuner and a resource for users to practice scales. Currently applications only exist for one feature exclusively. In order to offer a new product to the market, the combination of these two functions is required to be useful to a user in independent study. Doing so would provide a player the ability to use the tuner as a reference to determine whether they are playing the scale correctly with correct finger positioning.

3 Teaching computers to listen

3.1 Determining the Pitch

In order to implement the proposed Violin Tutor application, a method must be established to allow a program to listen to an input, analyse the input and assign it a classification, and then output what the system has determined that input to be.

To achieve this, a pitch detection algorithm (PDA) will need to be implemented. This is an algorithm that is designed to estimate the pitch of a signal, which as previously stated in music, can be quantified as a frequency.

There are two distinct properties that a signal can have: quasi-periodicity or oscillation. Quasi-periodicity is the property of a system that displays irregular periodicity. For something to have quasiperiodic behaviour it must have a pattern of recurrence with a component of unpredictability that does not lend itself to precise measurement. It is almost but not quite periodic, on a small scale it can appear to be periodic but unpredictable at some larger scale. Such periodic behaviour is defined as recurring at regular intervals, such as "every 60 minutes" [26].

The characteristic of oscillation is to have the quality of a repetitive variation, typically in time, surrounding a central value or between two or more different states. The advantage in analysing music is that it can be determined by its oscillating signal [27].

In determining what method to use it should be noted that although sounds from different sources can have the same frequency, the pressure wave of these sound waves are can vary massively. Figure 12 illustrates the difference of a pure sine wave (a waveform representing periodic oscillations of constant amplitude as given by a sine function, represented as a curve), a violin and a piano. It can be seen that although all three are set at the same frequency their sounds can be different. The violin produces a jagged waveform and therefore a sharper sound, whereas a piano produces a smoother waveform similar to a sine wave and therefore a purer sound. This means that one method which correctly determines the pitch of a piano may not have the same result on a stringed instrument and this should be accounted for.

When a string is under tension (such as on the violin) it will have resonant frequencies directly related to the mass, length, and tension of the string. Once struck by either the bow or plucked, the string will vibrate producing mechanical waves which are transmitted to the body of the instrument, which also vibrates along with the air inside it [28]. This vibrating string produces a sound with a constant frequency i.e. constant pitch which when the length or tension is correctly adjusted produces a musical note.

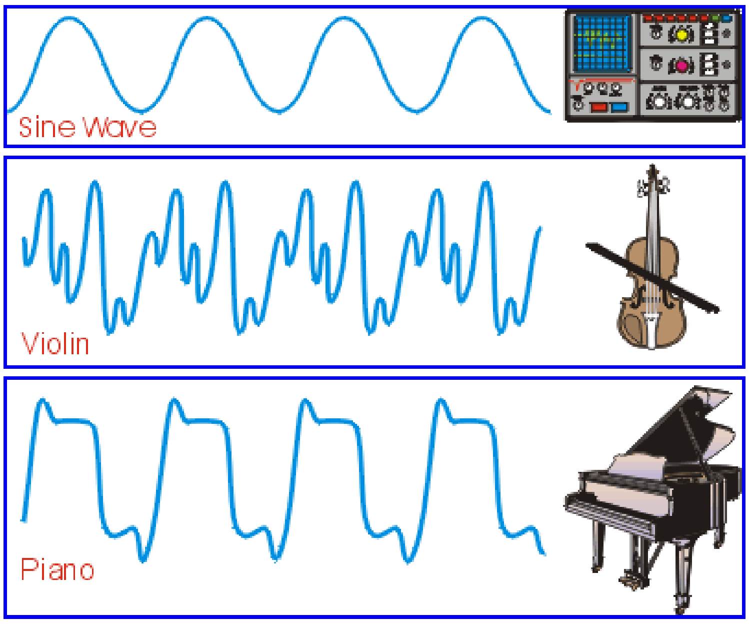


Figure 12. **Illustration of difference in overtones (12)**

3.1.1 The Tale of the Two Domains

There are two main terms that are required to be understood in order to determine the best approach to determine pitch, which is that of the time domain, and the frequency domain. The use of these contrasting terms was developed in U.S. communication engineering in the late 1940's [29].

Time domain is the analysis of mathematical functions, physical signals or time series of economic or environmental data, with respect to time. Conversely frequency domain is the same analysis with respect to frequency rather than time [30].

A time domain graph (Figure 13) shows how a signal changes with time, whereas a frequency domain graph shows how much of the signal lies within each given frequency band over a range of frequencies. This is demonstrated with the red line representing time and the blue line being frequency. In the 3D cross section you can see exactly what the natures of these frequencies actually are.

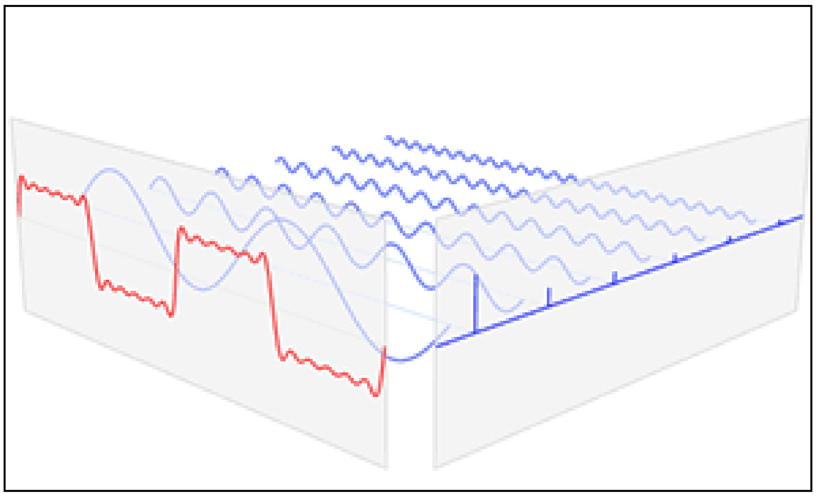


Figure 13. **Visualisation of the relationship between the time domain and the frequency domain (13)**

3.2 Time-domain approaches

In the time domain, a pitch detection algorithm (PDA) estimates the period of a quasiperiodic signal, then inverts that value to give the frequency. A very simple approach would be to measure the distance between zero crossing points of the signal (Figure 14). However this would not work well with complicated waveforms which are composed of multiple sine waves such as that of a violin.

There are more sophisticated approaches which compare segments of the signal with other segments offset by a trial period to find a match. This is the premise of the autocorrelation algorithm.

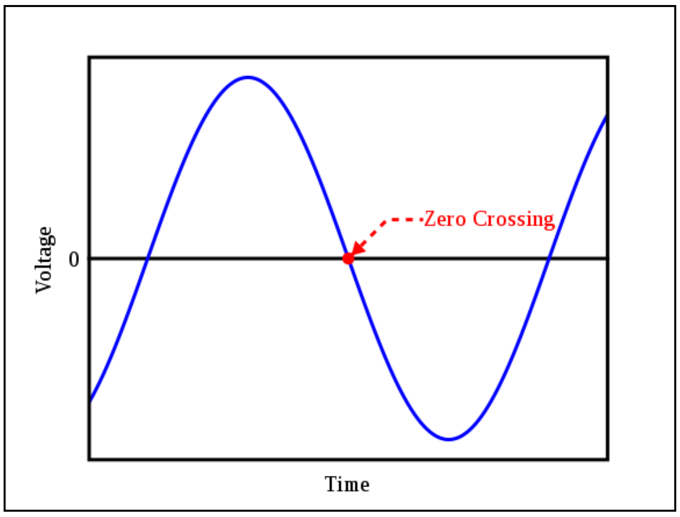


Figure 14. **Illustration showing the point of a zero crossing (14)**

3.2.1 Autocorrelation

Autocorrelation is used to compare the correlation of a signal with itself at different points in time. If the signal is periodic (oscillation), then the signal will be perfectly correlated with a version of itself, with the time-delay being an integer number of periods. This mathematical tool can be used for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies. It is often used in signal processing for analysing functions or series of values particularly time domain signals [31].

Mathematically, the autocorrelation corresponding to a delay time (τ) is calculated by:

1. First finding what the value of the signal is at a time i.e. - t,
2. Finding the value of the signal at a time (t) plus the delay (τ),
3. These two values are then multiplied together
4. Process is repeated for all possible times
5. The average is computed across of all those products.

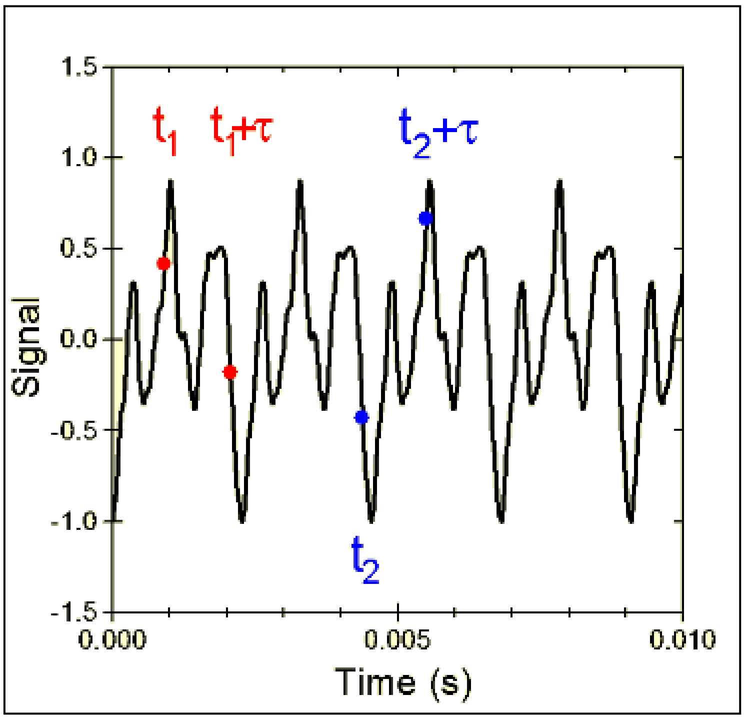


Figure 15. **Image of autocorrelation process (15)**

This process can be repeated for other values of τ, resulting in an autocorrelation which is a function of the delay time τ.

3.3 Frequency Domain Approaches

Time domain approaches can give quite accurate results for highly periodic signals, but can have false detection problems and can sometimes cope badly with noisy signals, and do not work well with polyphonic sounds. Pure autocorrelation may work well with a flute or a piano which produce a pure sound, but perhaps not so well with a stringed instrument which requires more precise accuracy.

To derive a higher accuracy more processing power is required, however efficiency in such a process can be obtained from the use of the Fast Fourier Transform (FFT).

3.3.1 Fast Fourier Transform

The Fourier Transform (FT) is a function, developed by Joseph Fourier [32], which converts a signal from its original domain (time) to a representation in the frequency domain and vice-versa [33]. The Fast Fourier Transform (FFT) is a computationally efficient method of generating an FT which was developed by James W. Cooley and John W. Tukey [34]. As previously mentioned, the autocorrelation algorithm tracks a signal over time. Fourier Transform differs as it can show what signal is present at each frequency.

With a time-based pattern, the FT measures every possible cycle, and returns the overall "cycle recipe" which is made up of the amplitude, offset, and rotation speed for every cycle that was found. An analogy for this would be if an FT was given a smoothie drink to process, the FT can find the recipe by running the smoothie through filters to extract each ingredient. The rationale for this is due to recipes being easier to analyse, compare and modify than using the original smoothie itself. With the information gained the original smoothie can be reconstructed.

This example of filtering a smoothie can be applied to a signal. Sound waves can be separated into ingredients (such as the bass and treble frequencies), allowing sections that are trying to be obtaining to be boosted, with unwanted aspects like crackling or other noises hidden. Once each individual frequency has been obtained, then the original signal can be reconstructed.

Given a trajectory the FFT can break this down into a set of related cycles that describes it. Each cycle has strength, a delay and a speed. The trajectory is processed through a set of filters:

- Each filter gives a cycle and the remainder of the trajectory. The filters available are low pass, high pass, band pass, band block and threshold.
- Low-pass filters block all frequency components above the cut-off frequency, allowing only the low frequency components to pass while high pass is the opposite [35].
- Band-pass filters only allow frequencies within a specific range determined by the lower and upper cut-off frequencies to pass the filters, while band-block filters remove all frequencies within the chosen range [35].
- Threshold filters remove frequencies whose amplitudes are below a specific threshold value [35].
- Filters are independent, each one catches a different part of the trajectory
- There are enough filters to catch all of the trajectory, i.e., the last filter leaves no trajectory remainder

What is exceptionally useful to know is that every periodic signal can be made up of the sum of sine waves, giving the same results no matter on the order of how they are mixed. Figure 16 shows that starting with a standard sine wave, that with every subsequent addition of a sine wave (each one being of a higher frequency than the previous one) the sum of these waves can affect the shape resulting in the appearance of a square wave. The same figure also shows how frequencies spikes would be visualized; each frequency is plotted with the Hz value against the content within the summed total of a square wave.

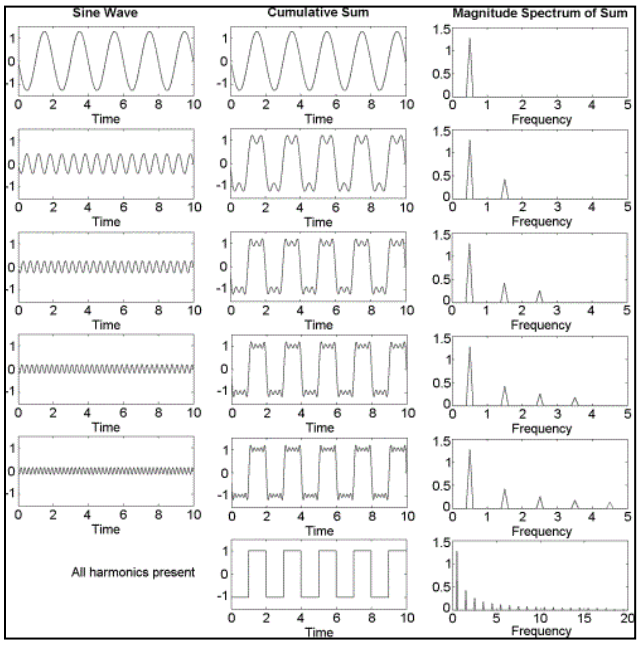


Figure 16. **Illustration showing how cumulative sum of sine waves form a square wave (16)**

3.4 Best of Both Worlds?

Using autocorrelation or FFT independently can obtain results in determining a pitch, however either method independently can lead to inaccurate classification. An unmodified autocorrelation method is better at finding the true fundamental for a wave but does not deal well with inharmonicity: the degree to which the frequencies of overtones depart from whole multiples of the fundamental frequency. This is not useful when determining a violin due to the instrument being inharmonic in nature [36].
Using solely an FFT can cause incorrect classifications of octaves due to an FFT not being useful for picking peaks in amplitude, autocorrelation would be better for that purpose.

Rather than focus on a singular domain, some algorithms expand upon one set implementation to improve efficiency and reduce error and latency. The one algorithm which is of particular interest for implementation of pitch detection in the violin tuner is the yin algorithm.

3.4.1 Yin Algorithm (YinFFT)

The yin algorithm is a fundamental frequency estimator for speech and musical sounds. Its basis is on the previously discussed autocorrelation method with a number of modifications that are used in order to prevent errors. With the development of this algorithm it was found in accordance with the Yin paper published in 2002 [37] that error rates were about three times lower than the best competing methods with no upper limit on the frequency search range. This makes the yin algorithm optimal for high pitched voices and music and the best possible algorithm to use within this proposed Violin Tutor application.

The fundamental frequency (F0) is defined as the lowest frequency of a periodic waveform, in terms of music the fundamental is the musical pitch of a note that is perceived as the lowest partial present. The fundamental frequency of a periodic signal is the inverse of its period, defined as the smallest positive member of the infinite set of shifts that leave the signal invariant. This definition applies strictly only to a perfectly periodic signal, where such a signal cannot be altered in any way without losing its perfect periodicity.

However music can depart from this definition in several ways, sounds may be periodic yet “outside the existence region” of pitch [38] or a sound may not be periodic yet evoke a pitch [39]. The word pitch is often used in place of the term fundamental frequency with estimation methods of frequency referred to as the previously discussed Pitch Detection Algorithm (PDA) [40]. The assumption is that pitch is derived from either the periodicity of neural patterns in the time domain or from the harmonic pattern of partials resolved by the cochlea (spiral cavity of the inner ear) in the frequency domain [41]. However both processes yield the fundamental frequency or its inverse, the period.

YinFFT algorithm is an optimised version of the Yin algorithm for computation of the frequency domain. It estimates the fundamental frequency corresponding to the melody of a monophonic music signal such as that made by a violin or by a person singing. Computations of autocorrelation can be reduced significantly using the FFT method.

3.4.2 YinFFT How it Works

What will follow now is step by step process of how the library class “FastYIN” developed by Tarsos [42] works in accordance to the paper presented by Aubio [37]. This library is an implementation of the YIN pitch tracking algorithm which uses an FFT to calculate the difference function, making the calculation of the difference more performant.

Step 1:

The library creates a new pitch detector for a stream with a defined sample rate and processes the audio in block of a defined size:

```
public FastYin(float audioSampleRate,
               int bufferSize)

audioSampleRate - 44100
bufferSize - 2048
```

The sample rate is 44.1 kHz which is the common sampling frequency of compact discs. The justification for this is the entire range of human hearing is 20-20kHz and to fulfil the Nyquist criterion which determines the stability or instability of a feedback system, the sampling rate must beat least twice the maximum analog frequency [43]. It was also noted that 44,100 can be factored as the product of the squares of the first first four prime numbers [43].

The bufferSize is the size of an audio buffer in samples, the smaller the buffer, the less time it will take for audio data to pass through it, therefore lower latency. However with a smaller buffer there is less overhead for delays in processing so the CPU would need to work harder to ensure delays are kept within the time allowed by the buffer [44].

Step 2:

The next step is that of the autocorrelation algorithm, which as detailed earlier in this paper is a cross correlation of a discrete signal with itself at different points in time is performed. The parameter is the audioBuffer which contains the audio information. It is not modified so it can be reused for FFT analysis. What it returns is an estimation of the pitch in Hz or -1 if there is no pitch detected or present in the buffer:

```
public PitchDetectionResult getPitch(float[] audioBuffer)
```

It derives from the supporting class PitchDetectionResult where not only is the pitch in Hz is derived but also a probability for said detected pitch existing. This probability can be used to measure levels of noise, periodicity, salience and clarity of the detected pitch – these elements can be calculated together:

```
public float getProbability()
public void setProbability(float probability)
public boolean isPitched()
```

The problem seen in Figure 17 from doing just autocorrelation is the sensitivity to peak amplitude changes; an increase could mean the chosen peak may not be the highest.

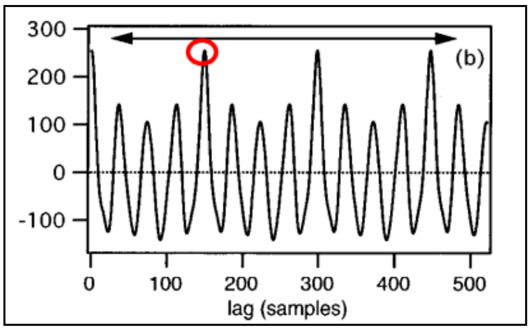
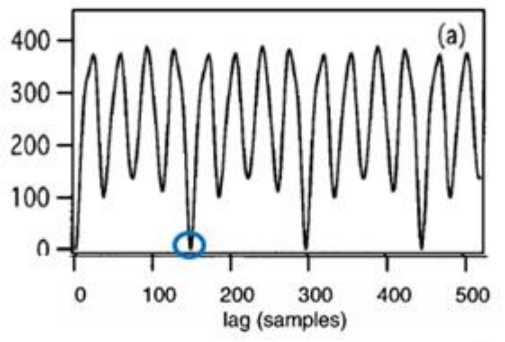


Figure 17. **Yin Step 2: Peak Amplitude (17)**

Step 3:

The difference is a measure of the average rate of change of a function over an interval. This difference function replaces the initial autocorrelation function in order to look for dips rather than peaks. This instance of the algorithm is implemented with an FFT, with the purpose to reduce the number of operations performed. It splits dt (T) into two components.

Looking for these dips is to overcome the problem of potential peak amplitude changes presented by autocorrelation, as it is immune to them. However a new problem is presented with dips close to 0 lag might be deeper than illustrated in Figure 18 due to imperfect periodicity.



$$d_t(\tau)=\sum_{j=1}^{W}(x_j-x_{j+\tau})^2$$

Figure 18. **Yin Step 3: Lowest Dip (18)**

Step 4:

In order to overcome the previous problem the cumulative mean normalized difference function replaces the difference function. This works out what then takes the deepest dip.

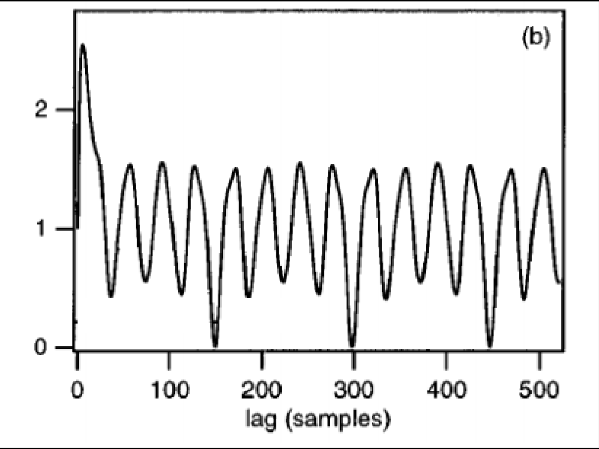


Figure 19. **Yin Step 4: Greatest Dip (19)**

The problem with this is that it may choose higher order dips producing a lower octave error, as seen in Figure 19. The first dip would be the deepest yet would not be the lowest.

Step 5:

An absolute threshold is implemented which is typically around 0.10-0.15. The purpose of this is once set, the first dip which exceeds this threshold is chosen rather than just the dip which is deepest.

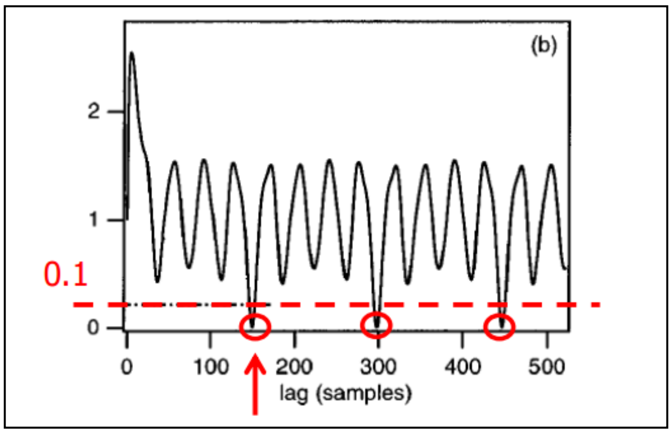


Figure 20. **Yin Step 5: Absolute Threshold (20)**

Step 6:

Parabolic interpolation is used to find the exact dip location; this is achieved by finding the highest local values and its two neighbours. The final step would be to use the best local estimate where the algorithm finishes and can provide its result.

4 Implementation

In a typical waterfall or agile method after completion of background research, the design phase would be undertaken. Due to the nature of this project this approach would not be very beneficial and iterative development is a better solution. This process is different where a small portion is developed and tested in repeated cycles. In order to fulfil the objectives the application requires:

- A tuning facility
- A scales page with the tuning facility
- A page to traverse through the two functions.

This chapter details the process undertaken with sample code to produce a functional application that meets the objectives set in section 1.3 to be useful to a player, while offering something that is not currently available on the Android market.

4.1 Implementation of the Yin algorithm into android

Having detailed in the previous chapter the different domains which can be measured and the method which would produce the least amount of error while delivering the best performance, the Yin algorithm (more specifically the library containing its calculations) needs to be implemented into producing a working Android application.

The main activity of the application is the “tuner” element, which can also be seen as the foundation of the app. It is crucial that this element, above all others, functions within the programme. The following details the creation of numerous Java classes in order to process audio and to determine its pitch. A detailed analysis is required to show the construction of the following classes and how they are used to fulfil the analysis of pitch.

The main functionality is detailed within the classes Tuner, Util and Horizontal Bar – with FastYin and Pitch Detection Java classes acting as libraries. These library classes are obtained from TarsosDSP [42], which is a collection of classes to perform simple audio processing, in line with the discussion of the Yin algorithm in chapter 3. These classes utilise jTransforms 2.4 which is used in the form of a jar file contained within the project [45]. It is an open source, multithreaded FFT library written in java. The Android developer site [46] was used in order to use correct documentation to develop the application. Code stated within this section is obtained from the newly created “Violin Tutor” app.

4.1.1 Setup for a recording

The constructor used to initialise audio recording is the AudioRecord class. This manages the audio resources for Java applications giving the ability to record audio from the input hardware of the platform. This class requires a sample rate; the number of channels; its recorder encoding format and the audio source as parameters:

```
private static final int SAMPLERATE       = 44100;
private static final int NUM_CHANNELS     = AudioFormat.CHANNEL_IN_MONO;
private static final int RECORDER_ENCODING = AudioFormat.ENCODING_PCM_16BIT;
```

The sample rate is to be 44.1 kHz which is the same level used within compact discs and is over double the maximum human hearing frequency of 20 kHz. This is also the format that all Android devices support [18]. "Channel_In_Mono" specifies that the source is to be used as an input and to be set as monophonic sound reproduction as the sound will be coming from one position rather than stereo which would deal with audio coming from different positions [47].

It is stated that encoding should be done at 16bit, meaning the audio bit sample is a 16 bit signed integer stored in a ByteBuffer with the short having a full range from -32768 to 32767. This is once again CD quality with a range suitable for the purpose of detecting frequency. This level is more than suitable for reading the frequency of a violin, when the highest note possible, B7, registers as 3951Hz (Figure 1).

With these settings the minimum buffer size can be established. This returns the minimum buffer size required for creation of an AudioRecord object, in byte unit:

```
final int minBufferSize = AudioRecord.getMinBufferSize(SAMPLERATE, NUM_CHANNELS, RECORD-
ER_ENCODING);
```

A final buffer size is established by calculating the obtained minimum buffer size and comparing it to a previously declared buffer size (int) by using the "Math.max" function, comparing both figures and using the higher value. Typical window sizes used for a 44.1 kHz signal can be 512, 1024, 2048 or 4096 samples, along with a 75% overlap in time [48].

With these parameters established a recording can be made, with the audio source declared as "MIC":

```
private void startRecording() {

private static final int BUFFER_SIZE    = 4096;
final int bufferSize = Math.max(minBufferSize, BUFFER_SIZE);


recorder = new AudioRecord(MediaRecorder.AudioSource.MIC,
        SAMPLERATE,
        NUM_CHANNELS,
        RECORDER_ENCODING,
        bufferSize);
recorder.startRecording();
isRecording = true;
```

4.1.2 Threading

One important implementation is to incorporate an additional thread which will be handling the task of processing the audio data. By default, Android is set so that all the individual components contained within the application, will run on the same process known as the main thread or sometimes referred to as the user interface (UI) thread.

There is a 5 stage hierarchy detailing the importance of processes, with the top process called the foreground process, which is also the most important. These processes usually occur when an interaction with the user is taking place or a service is active i.e. onResume(), onPause() functions. Processes continue to follow the hierachy in order of importance:

- Visible processes
- Service processes
- Background processes
- Empty processes.

The creation of a new thread alleviates poor performance and reduces the risk of blocking access to the UI with long operations. The effect of blocking access can result in error messages of “application not responding” which can be frustrating for the intended end user.

To incorporate this process hierarchy, the tuner class declares two foreground processes which call from two methods. When the Tuner class is used, onResume() is called and uses the method “startRecording”, consisting of establishing a new AudioRecord which starts recording on a newly declared thread. Conversely, when the user leaves the class to another activity, on-

Pause() is called using the method "stopRecording," which ends the created thread and recording function:

```
@Override
   public void onResume() {
      super.onResume();
      startRecording();


   @Override
   public void onPause() {
      super.onPause();
      stopRecording();
   }
private void startRecording() {….
new Thread() {….
```

4.1.3 Yin Algorithm

With a newly created thread, the main process of pitch detection can be implemented. A Short and a Float are newly created variables both containing the buffer size of 4096. A Short is an integer that is 2 bytes in size while a Floating point is 4 bytes with a fractional part.

A method contained within the recorder class called read, can read the audio data from the hardware for recording into the newly declared short array. It is encoded to 16bit and its constructor consists of three parameters. The first is audioData (sData), the second an off set in shorts which is referred to as buffer overlay with a value of three quarters the final sample rate size. The final parameter is the number of requested shorts, which is calculated as the difference of the buffer size minus the overlay.

```
final short[] sData = new short[BUFFER_SIZE];
final float[] fData = new float[BUFFER_SIZE];

final int diff = bufferSize - BUFFER_OVERLAY;


while (isRecording) {
   recorder.read(sData, BUFFER_OVERLAY, diff);

   for (int i = BUFFER_OVERLAY; i < diff; ++i) {
      fData[i] = (float) sData[i];
   }
```

The use of the float and short acts as the autocorrelation method where a point is picked in time and offset is set $(t + \tau)$.

In order for the Yin Algorithm to be executed the contents from the audiobuffer is used, however this value must be a float value. Audio could be read in as a float however its sample rate must be 32bit in size as opposed to 16bit pcm of a short.

```
fData[i] = (float) sData[i];
```

This conversion fits the criteria for the constructor of the Yin Algorithm requiring an audiobuffer in the form of a float. This audiobuffer can be processed in accordance with the FastYin library performing the Yin algorithm which was detailed in the previous chapter.

```
float currentPitch = yin.getPitch(fData).getPitch();
            if (currentPitch != -1 && currentPitch > 10) {
               pitch = currentPitch;
            }

            runOnUiThread(new Runnable() {
               public void run() {
                  updateNote(pitch);
               }
            });

            for (int i = 0; i < BUFFER_OVERLAY; ++i) {
            sData[i] = sData[i + diff];
            fData[i] = (float) sData[i + diff];
         }
      }
   }
   }.start();
}
```

4.1.4 Determining the Note

A returned processed pitch value in itself would be of no real use to an end user. In order to make it useful this value needs to correspond into what a musician would find to be a good associative value. Values which would be useful would be the note letter, what octave the note is and identifying when a note is sharp.

As said in the introduction an octave comprises of 12 separate semitone note values and are relative with the note C4 being double the frequency of C3 and half of C5. This relates very well when using mathematical terms to determine a pitch.

In the util class each possible note is declared in an enum, a special data type enabling a variable to be a set of predefined constants. Each one of the values must be equal to the ones predefined.

```
enum Note {
   A(false), AS(true), B(false), C(false), CS(true), D(false),
   DS(true), E(false), F(false), FS(true), G(false), GS(true);
```

It is known that for when playing notes in an ascending order once G# is reaching the next note in an ascending sequence would be A. Similarly if A was played the next note would be G# in descending sequence:

```
public Note next() {
        if (this == Note.GS) {
            return Note.A;
          return values()[ordinal() + 1];

public Note prev() {
        if (this == Note.A) {
            return Note.GS;
        return values()[ordinal() - 1];
```

Similarly in order to distinguish the octave number it is to be known once transcending from B to C a new octave range is reached which is represented in the class. This is done by using the obtained pitch frequency from the tuner class and inputting it into the method guessNote where the calculation starts at the base of 27.50 (which is A octave 0) and traverses using the held pitch and makes a rough "guess" on where the pitch in the audio buffer lies. The note, octave, offset offsetRatio and realPitch (frequency number) are assigned to a set value and used in a constructor method NoteGuessResult to be used to display the findings:

```
public static void guessNote(float freq, NoteGuessResult guess) {
   float lowA = (float) 27.50 * (float) Math.pow(2, 1.0/4.0)

   int octave = 1;

   Note note = Note.CS;

   final float a12 = (float) Math.pow(2, 1.0/12.0);

   float prevFreq = lowA;
   float noteFreq = lowA * a12;
   while (noteFreq < freq) {
      prevFreq  = noteFreq;
      noteFreq *= a12;
      note      = note.next();
      if (note == Note.C) {
         ++octave;

      }
   }

guess.note = note;
      guess.octave = octave;
      guess.offset = offset;
      guess.offsetRatio = offsetRatio;
      guess.realPitch = noteFreq;

 public NoteGuessResult(Note n, int oc, float of, float ofr, float real) {
        note       = n;
        octave      = oc;
        offset      = of;
        offsetRatio = ofr;
        realPitch   = real;
    }
```

4.1.5 Display the Findings

Having performing these operations to determine the frequency, octave, note and the offset ratio, these results need to be displayed to the end user. The creation of textview fields can facilitate this information and display what the frequency is and where it lies:

```
  private TextView freqTV = null;
…

 private  void updateNote(final float pitch) {
      long currentTime = System.currentTimeMillis();
      if (lastUpdateTime < currentTime - UPDATE_DELAY) {
        Util.guessNote(pitch, guess);

        updateColor(guess.offsetRatio);

        noteTV.setText(guess.note.noteLetter());
        String sharpText = guess.note.isSharp ? "#" : "";
        noteSharpTV.setText(sharpText);
        noteOctaveTV.setText(Integer.toString(guess.octave));
        freqTV.setText(String.format("%.1f (%.1f)", pitch, guess.realPitch));

        barView.setLength(guess.offsetRatio);

        lastUpdateTime = currentTime;
      }
  }
```

The feature of update colour allows the application to change the existing set colour of grey to green when the frequency from the inputted audio sound from the violin is played within a threshold of a set tolerance to a frequency (0.030).

```
public static int green = -1;
  public static int gray = -1;
  private void updateColor(float offsetRatio) {
      Util.drawColor = Math.abs(offsetRatio) <= 0.030 ? Util.green : Util.gray;
      setTVColor(Util.drawColor);
      barView.setColor(Util.drawColor);
   }

   private void setTVColor(int color) {
      noteTV.setTextColor(color);
      noteSharpTV.setTextColor(color);
      noteOctaveTV.setTextColor(color);
   }

}
```

As well as a frequency number to show a player how close they fall within a note, a bar representing a physical line shows how close to a notes frequency they are. When the bar falls in the centre the note is within the accepted frequency of the displayed note:

```
       canvas.drawRect(midW-1, 0, midW, height-1, drawPaint);

      canvas.drawRect(midW, midH-5, end, midH+5, drawPaint);
   }

     public void setLength(float length) {
      length = truncateLength(length);

      if (this.length != length) {
         this.length = length;
         postInvalidate();
      }
   }
        private float truncateLength(float length) {
      if (Math.abs(length) > 1) {
         return Math.signum(length);
      }
      else {
         return length;
      }
```

4.2 Home and Navigation

The tuning element provides the main functionality of the program. In order to navigate between the two main functions of tuner and scale page, a menu with navigational abilities is required. The main activity consists of an image of a violin used as a background with 3 text placements reading “Violin Tutor”, ”Tuner” and “Scales on top of it.

4.2.1 Picture Formating

The picture of the violin is for aesthetical purposes, in order to implement it an image must be first placed within the folder directory "application/resources/drawable." The image can then be referenced with "android:src."

```
ImageView
    android:id="@+id/violin"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@drawable/violin"
    android:scaleType="centerCrop"/>
```

The android:id function allows the ability to reference within the java class. The height and width of the image is stipulated to "match parent" which means to use the full capability of the outputted screen. The other option would be to "wrap content" which would only fill to the original dimension provided of the image. The picture of the violin is scaled to be centred and cropped rather than just centre. Figure 21 shows why this is needed.

.

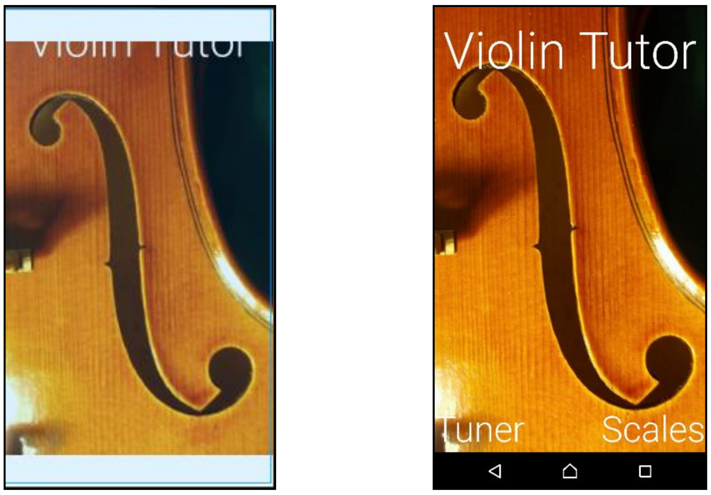


Figure 21. **Before and after scale type (21)**

4.2.2 Navigational Text

The text which reads "ViolinTutor" is again like the picture aesthetical, however the tuner and scales text are functional and are used to be navigational. Upon the declaration of the textview it is assigned an id along with its layout height, width, size and font. The declaration is for a box to be only as big as the text inside it (wrap_content) as opposed to fill the screen width and height.

The font is declared in "dp" which makes the transition better onto different devices and screen sizes. Dp stands for density independent pixels and is based on the physical density of the screen, 1 dp is 1 pixel on a 160 dpi screen. Using actual pixels would mean what working on one device could look very different on another and this makes porting to a range of devices potentially problematic. Going with density independent pixels makes the application uniform across devices regardless on the actual pixel count the screen has. This is illustrated in Figure 22 showing how 4 phones have different pixel counts and shows how a letter would look on each screen. Although detail increases with high pixel displays, the positioning would be consistent with screens that have a lower pixel count. This will mean consistency is maintained for all text declared in this tutor application.

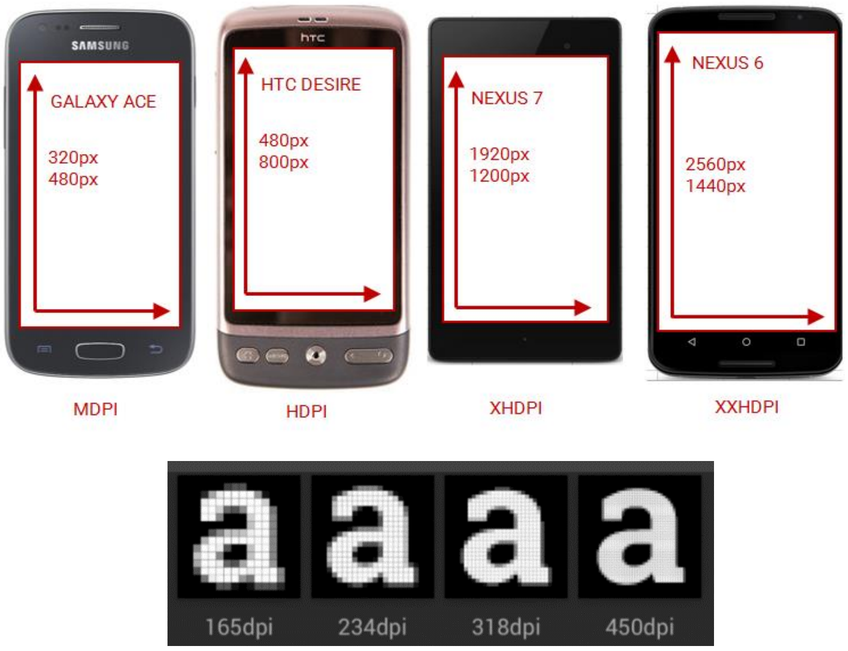


Figure 22. **DPI comparison (22)**

Upon creation of this text other information of positioning is introduced under "layout" aligning the text in accordance to the positioning of the screen (parent). The navigational part comes under the declaration clickable and onClick. Clickable true makes the area of the text an interactive feature and the onClick relates to a method declared in the main activity java file.

```
android:clickable="true"
android:onClick="openTuner"
```

```
public void openTuner(View view) {
    Intent intent = new Intent(this, Tuner.class);
    startActivity(intent);
}
```

4.3 Scales Page

The scales activity incorporates the tuning functionality as a means for the user to determine whether they are in correct tune while following a scale. Experienced violin players will be accustomed to reading sheet music and will have already developed a sense of finger placement on the fingerboard. For beginners and intermediate players a physical representation of finger placement would be beneficial in learning scales correctly.

In order to promote this, scales will be represented diagrammatically so users can refer to their finger placement with what is shown on the screen. Each vertical line corresponds to a particular string (G3, D4, A4, and E5) with circles on the string indicating where fingers should be placed, in order to produce a certain note. User accuracy can be cross referenced with the tuner/bar below it telling a user how close they are.

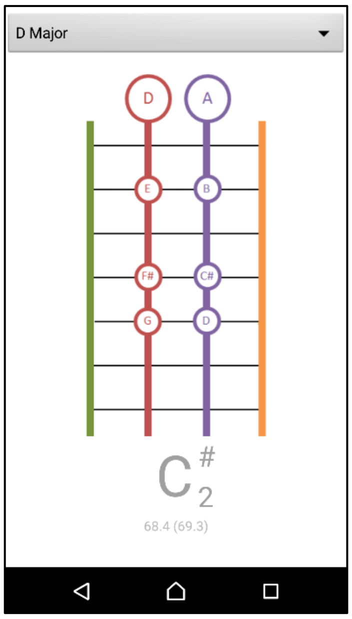


Figure 23. **The Scales Page (23)**

All of the scale diagrams were created within Word using the shapes and drawings feature. This involved me reading the sheet music of a scale and transposing it into the format stated. Each string is a different colour to help a user who is learning the instrument differentiate each string via association. Lines across the fingerboard are to indicate finger spacing although frets do not appear on the instrument like a guitar for example.

To illustrate the functionality, 4 scales were created and placed within the drawable folder, which will be accessed similarly to the home screen background earlier. The 4 scales chosen are the minimum requirements to be learnt in order to pass a grade 1 violin examination. The scales are D Major, G Major, A Major and E Minor. To be usable, a drop down menu is utilised (known as a spinner in android) which is populated with Strings of the named scales and linked with the same titled associative image. The end product allows the user to choose from the drop down menu with the image appearing in a fixed set location of the screen:

```
 spinner = (Spinner) findViewById(R.id.scale_spinner);

  ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
       android.R.layout.simple_spinner_item, scales);
  dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
  spinner.setAdapter(dataAdapter);

  spinner.setOnItemSelectedListener(new OnItemSelectedListener() {


            image.setImageResource(imgs.getResourceId(
             spinner.getSelectedItemPosition(), -1));
```

```
<string-array name="scales_list">

   <item>D Major</item>
   <item>G Major</item>
   <item>A Major</item>
   <item>E Minor</item>

</string-array>

   <integer-array name="scales_draw_list">
      <item>@drawable/d_major</item>
      <item>@drawable/g_major</item>
      <item>@drawable/a_major</item>
      <item>@drawable/e_minor</item>

   </integer-array>
```

5 Testing

In an attempt to fully exhaust possible testing methods there shall be multiple steps to test various aspects of the developed application. The first method shall be testing on a range of mobile devices to test compatibility, correcting any problems that occur. Further testing will be evaluated by other individuals of varying abilities of playing the violin, through the use of a questionnaire asking for both quantitative and qualitative answers to pre-defined questions (Appendix 3).

5.1 Compatibility on Devices

When a new project is created within Android Studio, the minimum API level has to be declared as a requirement before any programming can begin. As discussed in Chapter 2 the minimum level of API 15 was adopted in order to target the highest amount of users possible. To fully test out the application three different devices have been chosen, each with different operating systems, screen size, processor speed and amount of RAM.

The purpose of testing the application on a range of devices is to find out any inconsistencies within the application for each component, before asking people to give feedback on improvements. The specifications were obtained from GSMArena.com [49].

1. A Samsung galaxy mini s3 running on API level 15 (Android 4.0)

   Key Features:

   - 4 inch screen at 233 ppi pixel density
   - 1.0 GHz dual core processor
   - 1 GB Ram
2. A Sony Xperia Z2 running on API level 21 (Android 5.0)

   Key Features:

   - 5.2 inch screen at 424 ppi pixel density
   - 2.3 GHz quad core processor
   - 3 GB Ram
3. A Samsung S7 running on API level 23 (Android 6.0)

   Key Features:

   - 5.1 inch screen at 577 ppi pixel density
   - 2.3 GHz Octa Core processor
   - 4 GB Ram

5.2 White-box Testing and Black-box Testing

The process of white-box testing involves carrying out tests on components within the application, while having access and knowledge about the source code and what the desired inputs and outputs should be. The other process is referred to as black-box testing, where the functionality of the application is examined without knowledge of the source code.

Both processes were utilised at the unit, integration and system stage of programming, with analysis of produced code and analysis of the code functioning on a device, along with user testing once a working prototype was produced.

Unit testing was carried out on an ongoing basis during the applications development, to ensure that each individual java class method or xml component performed and/or displayed as expected, and it was corrected when it did not perform as expected or within the parameters. An example of this can be illustrated from the following piece of code:

```
float prevFreq = lowA;
float noteFreq = lowA * a12;
while (noteFreq < freq) {
   prevFreq  = noteFreq;
   noteFreq *= a12;
   note      = note.next();
   if (note == Note.A) {
      ++octave;
   }
```

The code reads, that once the current note upon calculation has traversed from A-G# of the previously declared enum, to then increment the octave number. This is incorrect as a whole octave spans from C toB and should increment on the following C and the correction would be "**if** (note == Note.*C*)." This was identifiable when the note and its frequency did not correspond with the octave number.

With each unit performing individually as expected, Integration testing was performed to see these units performing together, before performing system testing in order to find any inaccuracies or errors of the application. The following steps identify the three key activities that a user would be using and the process used to ensure not only compatibility, but to test the functionality.

5.2.1 The Home Page Unit

Upon opening the application it can be seen with Figure 24, the text of “Violin Tutor” is forced to begin on a new line due to the screen limitations of the galaxy s3 mini. On the other 2 devices there is no difference between the two. To ensure a uniform application across the devices, the font size was decremented from 68dp to 60dp.

The navigation to the tuner and scales activity was functional with no issue on all three devices.

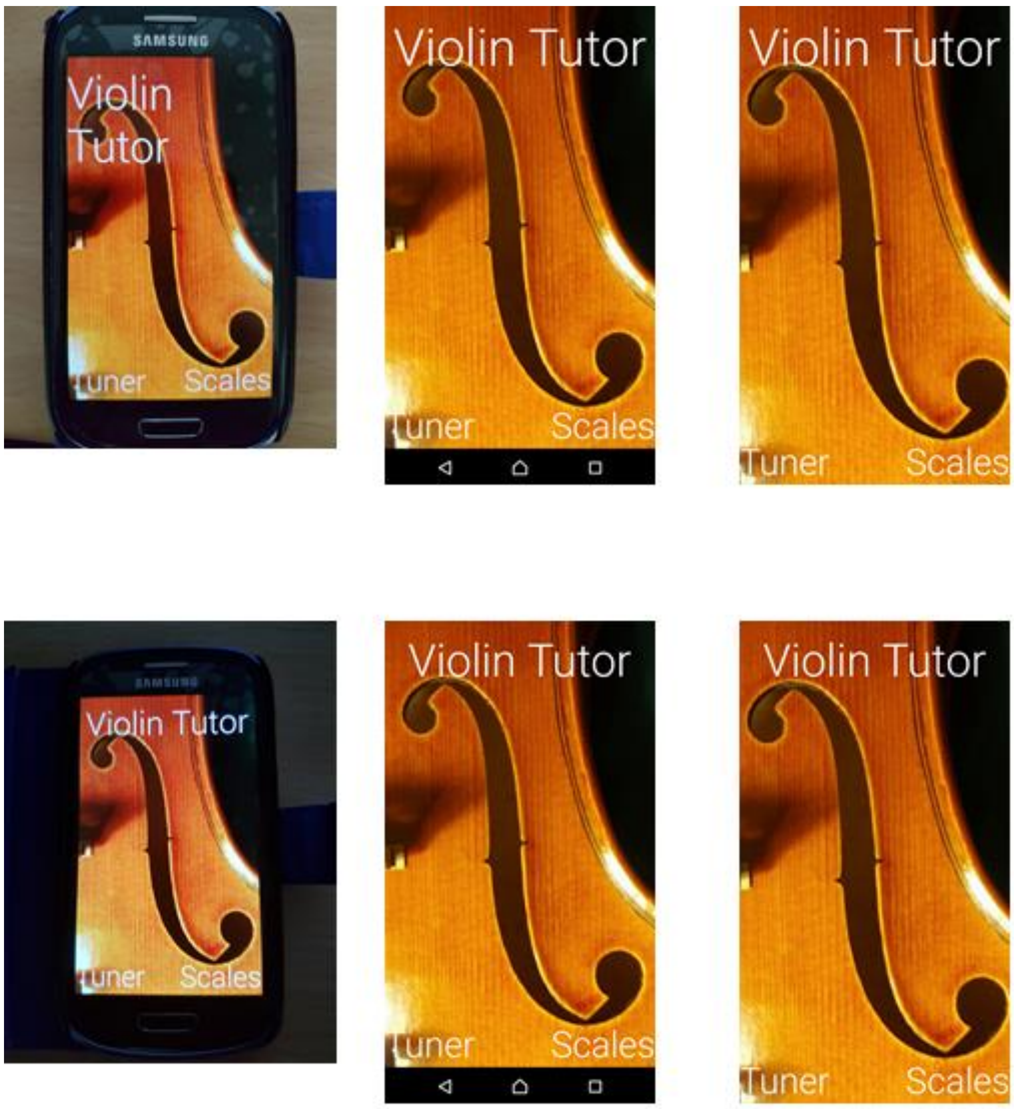


Figure 24. **Before and After Home activity code change (25)**

5.2.2 The Tuner Activity Tested

To effectively test the tuning capability of the application, each string on the violin was tuned in accordance with a store bought tuner to ensure that each device gave an accurate result in accordance with a tuner. Figure 25, illustrates each string being tuned one at a time and that the application gives a true reading of the frequency that corresponds with a store bought physical tuner.

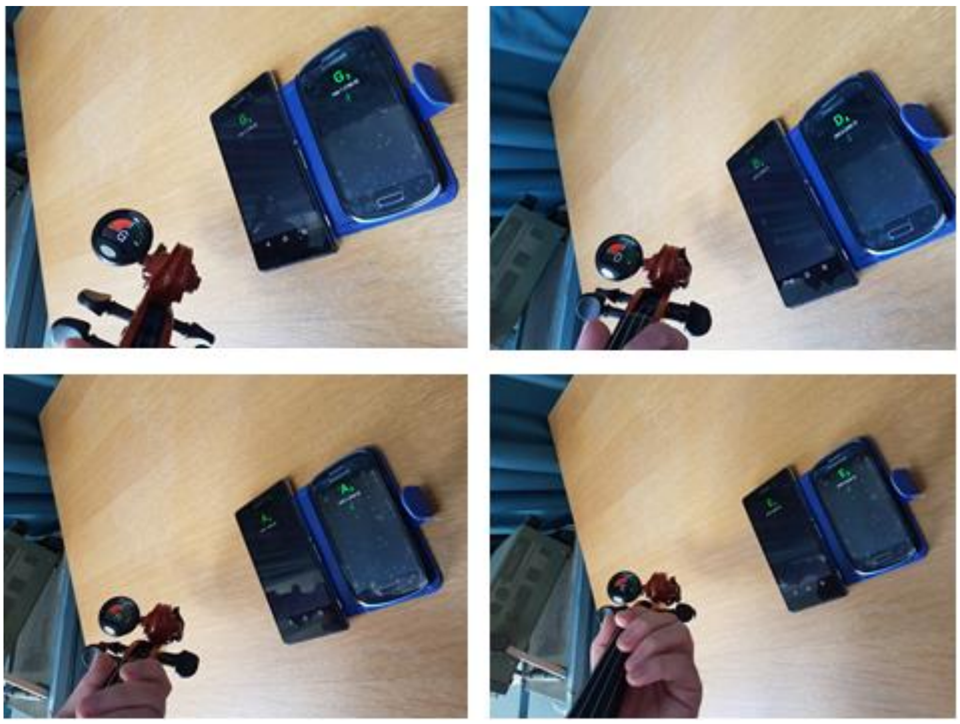


Figure 25. **Violin strings tuned to G3,D4,A4 and E5 (26)**

With this successful result a test with a fingered note was performed to show the application could register more than open tuning, and also tested the highest possible note frequency a violin is capable of producing C#7. (Note the conventional tuner could not pick up the frequency but the two devices could distinguish the frequency)

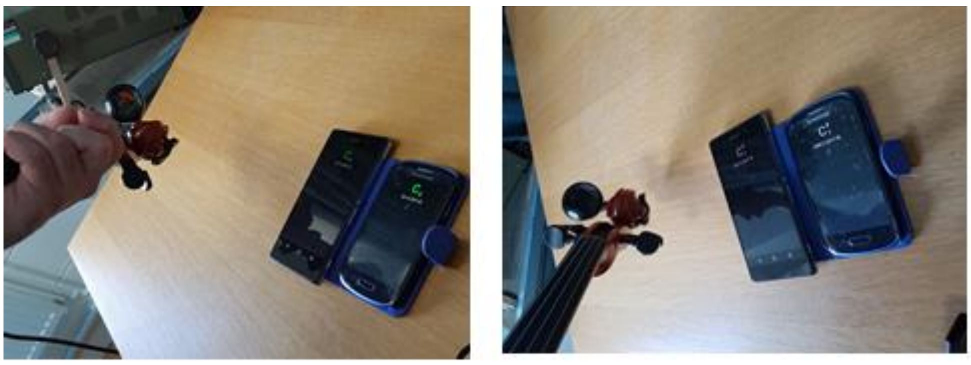


Figure 26. **Lowest and highest note performed on a violin (27)**

5.2.3 The Scales Page Tested

The scales page was loaded on all 3 devices and it can be seen that there was compatibility issues on the devices. Having the height of text and images originally in dp meant the frequency information was cut off of the smaller Samsung screen. This set height affected the larger Samsung screen due to having a higher pixel count, resulting in the scale being aligned to the left rather than centre. Also the information below the image was not set to the bottom like on the z2.

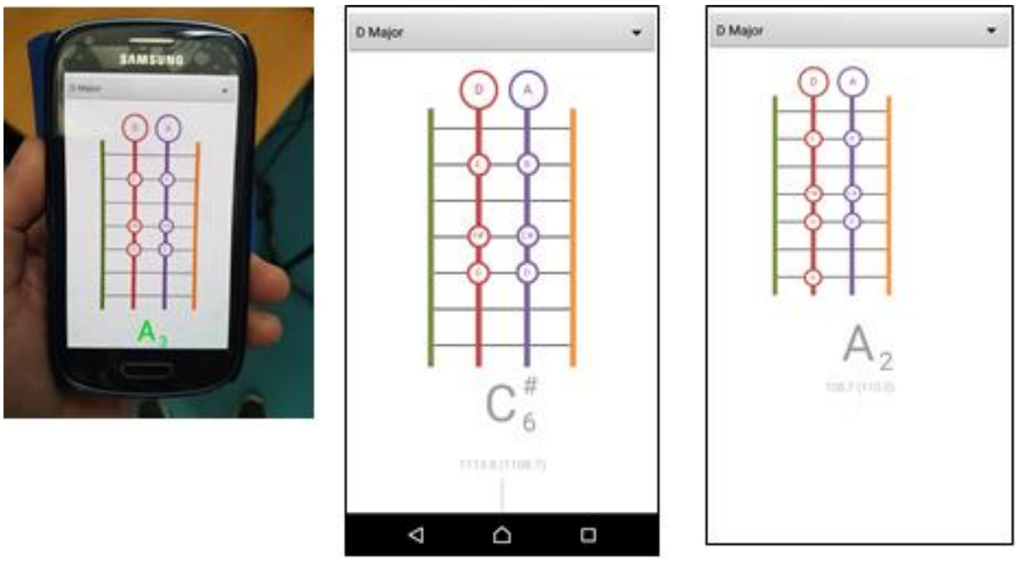


Figure 27. **Three devices displaying scales before activity code fix (28)**

By setting the height of each component to 0dp and adding a new value called weight, a value in the form of a percentage could be attributed. The result being the screen displays the information in relation to each individual devices capable output rather than a set value. All the information could be displayed on any device no matter on the pixel content or screen size.

```
android:layout_height="0dp"
android:layout_weight="0.70"
```

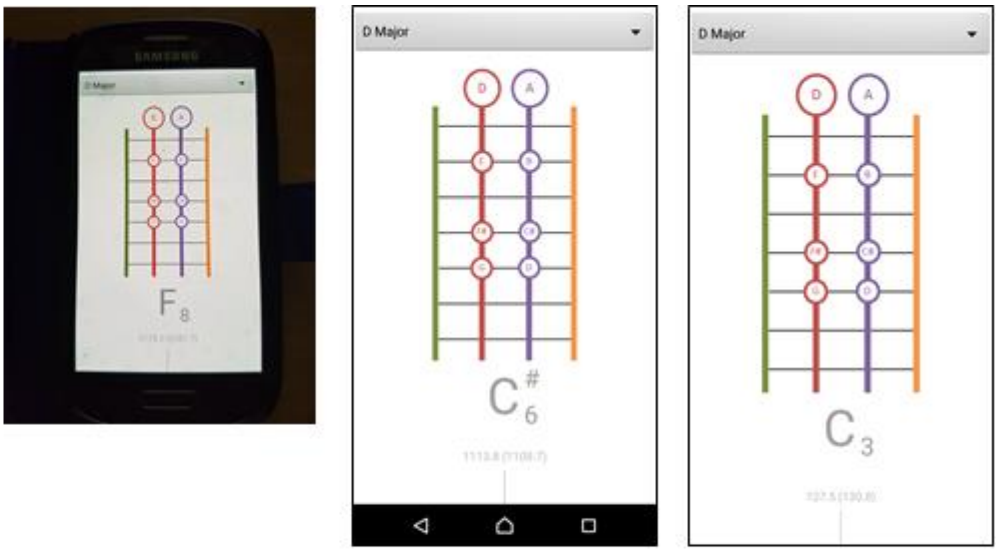


Figure 28. **Three devices displaying scales activity code fix (29)**

5.2.4 Final Fix

One other problem encountered while testing was that when the phone was put on its side the orientation made the whole application flip. In the application's current state this is not needed and the following line of code was added to the manifest of each of the activities to disable the feature for this version of software.

```
android:screenOrientation="portrait"
```

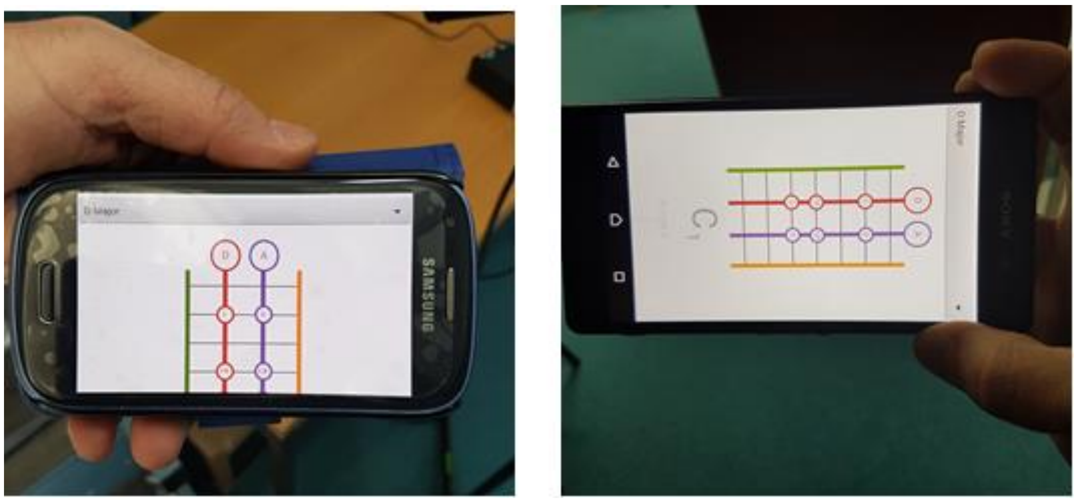


Figure 29. **Before and After Screen Orientation (30)**

5.3 User Testing

Having built a functioning application, user testing was required in order to evaluate each aspect both positive and negative, in order to improve future versions of the application before being available for the general public on the Android store. A questionnaire was devised (appendix 3) which asked both quantitative and qualitative questions about the application as a whole, as well as each individual component of the home, tuner and scale page.

User testing along with accompanying questionnaire was facilitated by 6 different users, who were chosen based on their previous experience of playing the violin.

- Individual who had never played the violin
- Two individuals who were studying in preparation to attain grade 3 graded examination (deemed beginner)
- Two individuals who had achieved grade 5 qualification (deemed intermediate)
- One individual who had achieved grade 8 qualification, the highest possible grade.

5.3.1 Results

The following quantitative results were collected which scored on a value of 1 representing bad to 5 representing excellent).

3: Overall Application:	4, 5, 4, 3, 4, 4–Average - 4
4: Home Page:	5, 5, 5, 5, 5, 4– Average – 4.83
7: Tuner Page:	4, 5, 4, 3, 4, 4– Average - 4
10: Scales Page:	4, 5, 5, 5, 4, 5– Average – 4.67

The average ratings for each component of the application show amongst the 6 testers, they thought the application was pretty good. The important aspect to analyse is the qualitative responses, where there is more depth into what the testers thought of the application and its components.

The responses will be broken down by the ability level of users:

5.3.1.1 Never Played

One big issue presented to me was that the application had no interface or prompts to teach someone in how to play a violin who has never played before. I was on hand to teach the tester correct technique in holding the violin, the bow and how to produce notes on the fingerboard.

After 15 minutes of this brief tuition, the tester was better able to use the testing functionality and to perform a scale.
In this current state, this application is not user friendly to someone who has not played the violin before, and therefore cannot claim to target a complete beginner. Further activities need to be implemented, such as a tutorial teaching the very basics in tuition in order to be beneficial to someone with no playing background.
In relation to the overall schema the tester liked:

- Colour coordinated strings and general diagram to teach scales being intuitive and the ability to learn a scale, liked the ability to have learnt a scale after spending time with the application.
- The ease of navigation between menus
- The simple layout of the tuner

The tester however found the sensitivity of the tuning to be very high and found it frustrating getting the tuner on the scales page to turn green but liked the horizontal bar.

5.3.1.2 Beginner

Having had two and three years of lessons respectively, both of the beginner testers already possessed the ability to play the violin with reasonable technique.

- Both testers liked the layout and navigational aspects of the application.
- In relation to the tuner function both testers commented in how they liked the ability to see the frequency of what they were playing. High precision in tuning could be attained, they currently struggle in their everyday practice to tune in accordance with other instruments.
- In relation to the scale page both said they liked the ability to stay in tune while practising a scale. One tester liked the way strings were individually coloured as it helped with identifying where to play.

5.3.1.3 Intermediate

Having both had six years of lessons respectively; both players were capable of playing the instrument.

- They both liked the fact that the tuner was similar to tuners they used and liked how fast the tuner response time was in pitch detection.
- They liked the scale functionality, although it was understood grade 1 scales were used to demonstrate the application worked, they would still definitely like to see their specific grade 5 scales implemented in the future.
- They would like to see the ability to have a scale grade to see how well they performed and to see sheet music incorporated into the application as well.

5.3.1.4 Expert

The tester had achieved grade 8 and was proficient in playing the violin. She liked the application as a whole and each individual page. The following suggestions of improvement that could be implemented:

- Understanding that this version had been populated with grade 1 scales exclusively, she would like to see the ability to select a user's grade to be attained, which showed the relevant scales needed to be learnt in the spinner menu.
- She thought that the tolerance level for when the note turned green on the tuning facility was excellent. However on the scale page may be too precise and potentially frustrating for a player and could have higher tolerance level.
- She was impressed in that when a scale was played at a fast tempo, that the tuning facility identified each note.
- She would like the future versions to have a grade specific layout so that a player's ability could be selected in an options menu.

6 Conclusion

6.1 Summary

The end result of this project has led to the creation of an application that has two distinct functions, the first being a functioning tuner designed for string related instruments. Through testing this has been shown to be both accurate and more importantly faster in pitch detection than many applications on the market.

The second function provides a user a diagrammatical picture of a chosen scale which is accompanied by the tuning facility. The user can determine whether or not they are playing the scale in tune, and remedy their finger positioning if necessary. This is helpful and a better way to improve playing than applications currently available that show how to play a scale with no reference indicating whether a player is correct in playing. Improvements can be made into future versions which could record and grade what a user is playing.

6.2 Evaluation

Each objective stated in section 1.2 shall be evaluated individually to see if and how these objectives were met. This is done through the evaluation of the application development process comparison to other apps currently on the market, and through analysis of feedback received from user testing.

Objective 1: A user can tune a violin without needing additional resources for tonal reference

The most important aspect to implement into the Violin Tutor application was a functioning tuner. The use of the Yin algorithm delivered this objective and exceeded personal expectations with a very quick response time and correct classification (section 5.2.2). Once the tuner page activity was accessed from the home page, no further interaction with the screen is necessary and the user can focus on playing the violin.

This reduced need for physical interaction with the device is a big advantage over other applications currently available, as it frees up the users hands for use on the instrument, something not accomplished by other apps currently on the market. The application "Violin" (section 2.6) produces the sound of a particular string upon touch of a button, similar to a tuning fork method previously discussed. This method requires repeated button touches and a trained musical ear to match frequencies of both the app and the instrument.

Other apps require a specific note to be selected before tuning. The assumption is the input from the instrument is compared to a stored integer within the app. Each different tuning requires a new selection, and it is limited to what has been programmed into the application.

The created application Violin Tuner determines the pitch of a user's violin and displays the current note being produced. The display of both frequency and a bar provide a reference to a player of how close their produced note is, which when falls within a set range of tolerance, turns the colour of the note displayed from grey to green. With this information a player can tune the violin with high accuracy and precision, without the need of additional resources.

Due to this functionality, the first objective has been fulfilled. The application is also an improvement in relation to identification of notes at faster tempos over other apps such as "gStrings". This was confirmed with user testing with two testers from intermediate level commenting on the response time, and an expert tester commented on how playing at a fast tempo correct note detection was maintained.

Objective 2: A musician can verify whether the scales that are being played are correct with on screen reference to tuning facility.

The scales page on the application allows a user to choose a scale to play from a drop down menu. This selection triggers the corresponding diagrammatic view of the chosen scale to be displayed. In addition, all the features contained within the tuner page appear underneath a chosen scale, allowing users easy navigation to other aspects of the app. Similarly to how a violin would be tuned, the same functions and principles can be applied when performing a scale. A player can visualise the proximity of what they are currently playing and correct their finger positioning if needed, in order to produce the desired note.

With this tuning ability on the scales page, this objective has been fulfilled as a user can cross reference to verify correct intonation while playing a scale. This was confirmed with user testing. where it was commented that from various testers that having a tuning facility underneath the scale, was very beneficial and allowed them to perform a scale correctly (Section 5.3.1).

Objective 3: It allows a user to learn the basics of violin finger positions and practice simple scales independently.

The way scales were created for the application is visually similar to the application “Violin Scales” (Figure 9). Having a representation of the violin fingerboard with relevant spacing between notes shows a user correct finger placement in order to perform a given scale. Going beyond the layout in “Violin Scales” strings were colour coded in an attempt to be more distinguishable, one of my testers confirmed that this was the case.

For the intended audience this objective could be seen as met, with prior knowledge in how to play the violin coupled with the basic knowledge in how scales are constructed, a user could use this app to practice and learn new scales independently. In order to target a user who has never played the instrument, the objective has not been met due to insufficient learning material. Although the scale imagery combined with the tuning reference provided users who had previously received violin lessons the facility to practice scales independently, for someone who had never played before that this information is not enough. Additional information or a tutorial would be required in order to target potential users looking to uptake the instrument and practice independently without interaction from additional sources.

Overall these objectives have been fulfilled and the development of the application can be deemed a success. However user testing highlighted several implementations that could be considered for integration in future versions of the application.

6.3 Future Work

Although the objectives for this project were successfully met, further development can be implemented into the Violin Tutor application, so that functionality can be expanded and improved.

By demonstrating that the scales page of the application is functional and useable as demonstrated by testing and positive user testing feedback, more scales can be populated into the list to make the app more beneficial for more players, rather than those looking to conquer and learn grade 1 scales. This could be done by making an overlay with user preferences and options. Such a facility could show only the relevant scales attributed to users selection of which graded examination is to be achieved. This would avoid having a spinner populated with hundreds of scales, which would require the user to scroll through to find the scale they wanted, a process which could be time consuming to the user.

The scales page allows a player to verify they are playing scales in tune. This tuning facility on the page was commented by one of the testers to be too accurate in when the note turns green on a successful note being played. The tolerance could be expanding slightly so that it is still accurate but not too distracting for a player focusing on getting the note to turn green, rather than learning a scale. What would be useful to implement would be a feature that records the player while they performing a scale and grade how well it was played e.g. how many notes were hit successfully. The app could store information of played scales and highlight weak points based on the player's history.

It was apparent through testing that this application was not suitable for users who had never played the violin before. With videos or a series of still screens, basic information could be taught to a user to in how to hold a violin, how to hold a bow, how to play notes on the fingerboard so that the user can perform the scales contained within the application.

After managing to successfully complete the object of this project, to create a fully working app for those in the early stages of violin teachings, a new overall goal could be established to create an application which encompassed all aspects of learning the violin and made to be so that players of all abilities could use the application. The end product could be similar to the game Rocksmith mentioned in section 2.6. To deliver on this potential objective, sheet music could be incorporated into the application that could be played along with a backing track. Options such as adjustable beats per minute (bpm) could be implemented for a piece which a player may find difficult before increasing the speed upon proficiency. Upon completion of a piece a percentage score could be computed in how many notes were successfully hit. This feature would be a big undertaking and could take a long time for one person to develop.

The development of this application has been highly successful. The application is in fully working order and has had positive reviews during user testing. In terms of future work on the application in its current form, very little needs to be done. So long as the app is maintained, monitored for bugs being flagged from users and kept compatible for any updates to Android software, this app can continue to be fully functional. Once the application has been populated with scales required for all of the eight graded examinations, and organised in accordance with the proposed options facility, publication onto the Android play store will be pursued.

7 References

Illustrations:

(1) *Figure 1*. Christian Pacheco (2015) Note Frequencies [Table] Retrieved from:<https://www.seventhstring.com/resources/notefrequencies.html> (Accessed on 01.09.16)

(2) *Figure 2*. Wood, Sienna M.(2015) Chromatic Scale in C [Image] Retrieved from: <http://www.musiccrashcourses.com/scores/ChromScale.png> (Accessed on 01.09.16)

(3) *Figure 3.* Table showing the total range of notes and corresponding frequencies that can be played on a violin [Table]

(4) *Figure 4.* Sotakeit (2006) Labelling of a violin [Image] Retrieved from: <https://en.wikipedia.org/wiki/Violin#/media/File:Violinconsruction3.JPG> (Accessed on 01.09.16)

(5) *Figure 5.* Hirst. Richard John (2016) Percentage breakdown of devices and which version of android used. [Screenshot] Retrieved from : Android Studio 2.1

(6) *Figure 6.* Hirst. Richard John (2016) Message from starting a new project in android studio showing that API level 15 will target 97.4% of devices. [Screenshot] Retrieved from: Android Studio 2.1

(7) *Figure 7.* Nicoguaro (2016) Polar Pattern Cardioid [Image] Retrieved from: <https://upload.wikimedia.org/wikipedia/commons/thumb/9/93/Polar_pattern_cardioid.svg/801px-Polar_pattern_cardioid.svg.png> (Accessed on 01.09.16)

(8) *Figure 8* Hirst. Richard John (2016) Two different electronic tuners [Image] (Photographed with Richards personal Samsung S7 Phone)

(9) *Figure 9* Hirst. Richard John (2016) “Violin” Application Screenshot [Image] (Screenshots from Richards personal Samsung S7 Phone)

(10) *Figure 10* Hirst. Richard John (2016) “G Strings” Application Screenshot [Image] (Screenshots from Richards personal Samsung S7 Phone)

(11) *Figure 11* Hirst. Richard John (2016) “Violin Scales” Application Screenshot [Image] (Screenshots from Richards personal Samsung S7 Phone)

(12) *Figure 12.* Hollis, Benjamin (1999) Overtones [Image] Retrieved from: < https://method-behind-the-music.com/mechanics/physics/> (Accessed on 01.09.16)

(13) *Figure 13* Barbosa. Lucas V. (2013) Visualisation of the relationship between the time domain and the frequency domain [Image] Retrieved from:< https://upload.wikimedia.org/wikipedia/commons/5/50/Fourier_transform_time_and_frequency_domains.gif/> (Accessed on 01.09.16)

(14) *Figure 14.* Public Domain (2006) Illustration showing the point of a zero crossing [Image] Retrieved from: < https://upload.wikimedia.org/wikipedia/commons/thumb/0/03/Zero_crossing.svg/576px-Zero_crossing.svg.png/> (Accessed on 01.09.16)

(15) *Figure 15* B. H. Suits (1998) Image of autocorrelation process [Image] Retrieved from: < http://www.phy.mtu.edu/~suits/AutocorrOboeSound.gif/> (Accessed on 01.09.16)

(16) *Figure 16.* J Cardiovasc Electrophysiol (2007) Illustration showing how cumulative sum of sine waves form a square wave [Image] Retrieved from: < http://img.medscape.com/fullsize/migrated/560/221/jce560221.fig1.gif/> (Accessed on 01.09.16)

(17) *Figure 17* Zhiyao Duan (2015) Yin Step 2: Peak Amplitude [Image] Retrieved from: < http://www.ece.rochester.edu/~zduan/teaching/ece477/lectures/Topic%204%20-%20Single%20Pitch%20Detection.pdf/> (Accessed on 01.09.16)

(18) *Figure 18* Zhiyao Duan (2015) Yin Step 3: Lowest Dip [Image] Retrieved from: < http://www.ece.rochester.edu/~zduan/teaching/ece477/lectures/Topic%204%20-%20Single%20Pitch%20Detection.pdf/> (Accessed on 01.09.16)

(19) *Figure 19* Zhiyao Duan (2015) Yin Step 4: Greatest Dip [Image] Retrieved from: < http://www.ece.rochester.edu/~zduan/teaching/ece477/lectures/Topic%204%20-%20Single%20Pitch%20Detection.pdf/> (Accessed on 01.09.16)

(20) *Figure 20* Zhiyao Duan (2015) Yin Step 5: Absolute Threshold [Image] Retrieved from: < http://www.ece.rochester.edu/~zduan/teaching/ece477/lectures/Topic%204%20-%20Single%20Pitch%20Detection.pdf/> (Accessed on 01.09.16)

(21) *Figure 21* Hirst. Richard John (2016) Before and after scale type [Image] (Screenshots from Sony Xperia Z2)

(22) *Figure 22* Garofalo, Raffaele (2015) DPI comparison [Image] Retrieved from: < http://blog.raffaeu.com/archive/2015/03/04/understand-density-independent-pixels-dpi.aspx/> (Accessed on 01.09.16)

(23) *Figure 23* Hirst. Richard John (2016) The Scales Page [Image] (Screenshots from Richards personal Sony Xperia Z2)

(24) *Figure 24* Hirst. Richard John (2016) Before and After Home activity code change [Image] (Screenshots from Richards personal Samsung S3 Mini, Sony Xperia Z2 and Samsung S7 Phone)

(25) *Figure 25* Hirst. Richard John (2016) Before and After Home activity code change [Image] (Screenshots from Richards personal Samsung S3 Mini, Sony Xperia Z2 and Samsung S7 Phone)

(26) *Figure 26* Hirst. Richard John (2016) Violin strings tuned to G3,D4,A4 and E5 [Image] (Photographed with Richards personal Samsung S7 Phone)

(27) *Figure 27* Hirst. Richard John (2016) Lowest and highest note performed on a violin [Image] (Photographed with Richards personal Samsung S7 Phone)

(28) *Figure 28* Hirst. Richard John (2016) Three devices displaying scales activity before code fix [Image] (Screenshots from Richards personal Samsung S3 Mini, Sony Xperia Z2 and Samsung S7 Phone)

(29) *Figure 29* Hirst. Richard John (2016) Three devices displaying scales activity code fix [Image] (Screenshots from Richards personal Samsung S3 Mini, Sony Xperia Z2 and Samsung S7 Phone)

(30) *Figure 30* Hirst. Richard John (2016) Before and after screen orientation [Image] (Photographed with Richards personal Samsung S7 Phone)

Text:

[1]. Nave.Rob "Sensitivity of Human Ear", *Hyperphysics.phy-astr.gsu.edu*, 2016. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/sound/earsens.html. [Accessed: 02- Sep-2016].

[2]. Pilhofer, Michael (2007). Music Theory for Dummies. p. *97.* Wiley

[3]. Latham. A (2004).*The Oxford dictionary of musical terms*. Oxford: Oxford University Press

[4]. Kamien. Roger (2008) Music: An Appreciation,6th Edition, p46, McGraw-Hill

[5] Violin School, "Arpeggios | ViolinSchool", *Violinschool.org*, 2016. [Online]. Available: https://www.violinschool.org/arpeggios/. [Accessed: 02- Sep- 2016].

[6]. ABRSM "ABRSM: Bowed Strings exams", *Gb.abrsm.org*, 2016. [Online]. Available: http://gb.abrsm.org/en/our-exams/bowed-strings-exams/. [Accessed: 02- Sep- 2016].

[7] ABRSM Scales "ABRSM: Violin Grade 3", *Gb.abrsm.org*, 2016. [Online]. Available: http://gb.abrsm.org/en/our-exams/bowed-strings-exams/violin-exams/violin-grade-3/. [Accessed: 02- Sep- 2016].

[8] Associated Press "First cell phone a true 'brick'", *msnbc.com*, 2005. [Online]. Available: http://www.nbcnews.com/id/7432915/ns/technology_and_science-wireless/t/first-cell-phone-true-brick/#.T2o-y3mP5JE. [Accessed: 02- Sep- 2016].

[9] Braggs Steven, "80s brick phones", *Mobilephonehistory.co.uk*, 2012. [Online]. Available: http://www.mobilephonehistory.co.uk/arcticles/80s_brick_phones.php. [Accessed: 02- Sep-2016].

[10] Bonnington. Christina, "In Less Than Two Years, a Smartphone Could Be Your Only Computer", *WIRED*, 2015. [Online]. Available: http://www.wired.com/2015/02/smartphone-only-computer/. [Accessed: 02- Sep- 2016].

[11] Boren, Zachery "There are officially more mobile devices than people in the world", *The Independent*, 2014. [Online]. Available: http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html. [Accessed: 02- Sep- 2016].

[12] Smartphone OS Share "IDC: Smartphone OS Market Share", *www.idc.com*, 2016. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed: 02- Sep- 2016].

[13] Sims, Gary "I want to develop Android Apps - What languages should I learn?", *Androi-dauthority.com*, 2016. [Online]. Available: http://www.androidauthority.com/want-develop-android-apps-languages-learn-391008/. [Accessed: 02- Sep- 2016].

[14]"Swift - Apple (UK)", *Apple (United Kingdom)*, 2016. [Online]. Available: http://www.apple.com/uk/swift/. [Accessed: 02- Sep- 2016].

[15] Price, Rob. *Uk.businessinsider.com*, 2015. [Online]. Available: http://uk.businessinsider.com/idc-global-smartphone-sales-slow-android-81-share-2019-iphone-2015-8. [Accessed: 02- Sep- 2016].

[16]Eason, Jamal. "An update on Eclipse Android Developer Tools | Android Developers Blog", *Android-developers.blogspot.co.uk*, 2015. [Online]. Available: http://android-developers.blogspot.co.uk/2015/06/an-update-on-eclipse-android-developer.html. [Accessed: 02- Sep- 2016].

[17] “Free audio recording, editing software | Download free Adobe Audition CC trial", *Ado-be.com*, 2016. [Online]. Available: https://www.adobe.com/products/audition.html. [Accessed: 07- Sep- 2016].

[18] “AudioRecord | Android Developers", *Developer.android.com*, 2016. [Online]. Available: https://developer.android.com/reference/android/media/AudioRecord.html#AudioRecord%28int,%20int,%20int,%20int,%20int%29. [Accessed: 07- Sep- 2016].

[19] SweetWater, What’s the Difference between Electret and Capacitor Mics?", *inSync*, 2016. [Online]. Available: http://www.sweetwater.com/insync/whats-the-difference-between-electret-and-capacitor-mics/. [Accessed: 07- Sep- 2016].

[20] “Violin Tuner Google Play” *Play.google.com*, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.alvinyu.violintuner&hl=en. [Accessed: 07- Sep- 2016].

[21] “gStrings Google Play” *Play.google.com*, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=org.cohortor.gstrings&hl=en_GB. [Accessed: 07- Sep- 2016].

[22] “Easy Violin Google Play” *Play.google.com*, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.ViolinTuner.Think&hl=en. [Accessed: 07- Sep- 2016].

[23] “Violin Scales Google Play” *Play.google.com*, 2016. [Online]. Available: https://play.google.com/store/apps/details?id=com.micahdetamore.violinscales&hl=en_GB. [Accessed: 07- Sep- 2016].

[24] “Scales Practice Google Play” *Play.google.com*, 2016. [Online]. Available:

https://play.google.com/store/apps/details?id=com.andymstone.scales&hl=en. [Accessed: 07- Sep- 2016].

[25] "Rocksmith® 2014 - Learn to Play Guitar & Bass | Ubisoft® (US)", *Rocksmith.ubi.com*, 2016. [Online]. Available: http://rocksmith.ubi.com/rocksmith/en-us/home/. [Accessed: 07- Sep- 2016].

[26] L. Revolvy, *Broom02.revolvy.com*, 2016. [Online]. Available: http://broom02.revolvy.com/main/index.php?s=Quasiperiodicity&item_type=topic&overlay=1 . [Accessed: 07- Sep- 2016].

[27] Reddy, K.R. and Balasubramanian. V. (1994) Oscillations and Waves p206 Universities Press Pvt, Ltd, India

[28] Benade. A.H, "The Oscillations of a Bowed String", *Zainea.com*, 2016. [Online]. Available: http://www.zainea.com/Oscilationsofbowedstring.htm. [Accessed: 07- Sep- 2016].

[29] Lee, Y W, Cheatham T P and J. Wiesner, (1949) The application of correlation functions in the detection of small signals in noise p1165 [Cambridge]: Massachusetts Institute of Technology, Research Laboratory of Electronics.

[30] Broughton, S.A.; Bryan, K. (2008). Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing. p. 72. New York: Wiley.

[31] Suits. B.H, "Autocorrelation (for sound signals)", *Phy.mtu.edu*, 1998. [Online]. Available: http://www.phy.mtu.edu/~suits/autocorrelation.html. [Accessed: 07- Sep- 2016].

[32] Fourier, J.B. Joseph (1822), Théorie analytique de la chaleur (in French), p525 , Paris L Firmin Didot, pere et fils

[33] Ahmad. Ayaz (2016) Smart Grid as a Solution for Renewable and Efficient Energy p270, LUMS School of Science & Engineering, Pakistan

[34] Cooley. James W and Tukey. John W (1965) *An Algorithm for the Machine Calculation of complex fourier series* Mathematics of Computation Vol. 19, No. 90 p297 American Mathematical Society

[35] "FFT Filters", *Originlab.com*, 2016. [Online]. Available: http://www.originlab.com/doc/Origin-Help/FFT-Filter. [Accessed: 07- Sep- 2016].

[36] "Inharmonicity", *Liquisearch.com*, 2016. [Online]. Available: http://www.liquisearch.com/inharmonicity. [Accessed: 07- Sep- 2016].

[37] Cheveigne. Alain de (2002) *YIN, a fundamental frequency estimator for speech and music* J.Acoust.Soc.Am 111 (4) 1917-1930

[38] Ritsma, R.J. (1962).Existence region of the tonal residue. I"J.Acoust.Soc.Am 34, 1224-1229

[39] Yost, W.A. (1996). Pitch strength of iterated rippled noise J.Acoust.Soc.Am 100, 3329-3335

[40] Hess, W. (1983) Pitch Determination of Speech Signals Springer-Verkag, Berlin

[41] Licklider, J.C.R. (1951) A duplex theory of pitch perception Experientia 7, 128-134

[42] "JorenSix/TarsosDSP", *GitHub*, 2016. [Online]. Available: https://github.com/JorenSix/TarsosDSP/blob/master/src/core/be/tarsos/dsp/pitch/FastYin.java. [Accessed: 07- Sep- 2016].

[43] Schulzrinne, Henning. “Explanation of 44.1 kHz CD sampling rate” w*ww1.cs.columbia.edu*, 2008. [Online]. Available: http://www1.cs.columbia.edu/~hgs/audio/44.1.html. [Accessed: 07- Sep- 2016].

[44] "What buffer size should I use?", *Soundonsound.com*, 2010. [Online]. Available: http://www.soundonsound.com/sound-advice/q-what-buffer-size-should-i-use. [Accessed: 07- Sep- 2016].

[45] “JTransforms” *Sourceforge.net*, 2016. [Online]. Available: https://sourceforge.net/projects/jtransforms/files/jtransforms/2.4/. [Accessed: 07- Sep- 2016].

[46] "Introduction to Android | Android Developers", *Developer.android.com*, 2016. [Online]. Available: https://developer.android.com/guide/index.html. [Accessed: 07- Sep- 2016].

[47] "Sound Systems: Mono vs. Stereo", *Mcsquared.com*, 2016. [Online]. Available: http://www.mcsquared.com/mono-stereo.htm. [Accessed: 08- Sep- 2016].

[48] McLeod, Phillip. (2005) “A smarter way to find pitch” ICMC 05 ,138-141

[49] “Full phone specifications", *Gsmarena.com*, 2016. [Online]. Available: http://www.gsmarena.com/samsung_galaxy_s7-7821.php. [Accessed: 07- Sep- 2016].

Appendix 1 – User guide

In order to use the violin tuner touch the Violin Tutor icon where the home page will load:

The Home Page

- To navigate to the Tuner page press the word tuner
- To navigate to the Scales page press the word Scales
- To return to the home back press the back button on your device

The Tuner Page

1) By playing the violin the microphone will detect frequency being produced from your violin
2) A rough estimate of what note is being produced will be displayed indicating it's letter name and octave
3) Below the note is two numbers. The number on the left is the frequency currently being played. The number on the right is the set frequency of the note being displayed. When the number on the left is equal to the one on the right, the note being produced is in tune.
4) A bar view underneath the frequency numbers is a visual view of the frequency number, the aim is to get the bar in the middle.
5) Once a note produced from the violin is in tune, the on screen display text will turn green.

The Scales Page

1) Click on the spinner at the top to select a scale to play
2) To play a scale start at the first note that appears on the left hand side
3) Each coloured corresponds to a particular string. A circle on the string corresponds to a finger position
4) Play each note one after another in an ascending order

5) Once reached the last note on the right hand side, play each note in the reverse order descending fashion.

6) Below the Scale is the exact same tuning function found on the tuner page.

Appendix 2 – Installation guide

The Violin Tuner application is currently not available on the Android market.

In order to install on a device the program must be loaded within Android Studio.

The application is contained with the folder "Violin Final Lite" with the Project called "My-Application."

Plug in the required device via a USB cable and Android Studio should detect the device. If not new drivers may be needed which can be downloaded from: https://developer.android.com/studio/run/win-usb.html.

Once Android Studio has detected a successful device connection following the following steps

1) On the toolbar click "Run"
2) Under "Run" click "RunApp"
3) A Diaglog box will appear with your connected device, select the device and click ok
4) The APK will be built and gradle generated on the device.
5) The Application will run on the device and can be selected in future within you app list titled "Violin Tutor"

Appendix 3 – Questionnaire Sample

1. Please state your experience with the violin

   Never Played   Beginner   Intermediate   Expert

2. If you have attained a graded certification what is your highest achieved grade?

   ____

3. Overall how would you rate the application overall (1 being bad, 5 being excellent)

   1 2 3 4 5

4. How would you rate the layout of the home page? (1 being bad, 5 being excellent)

   1 2 3 4 5

5. What do you like about the home page?

   ____

6. How would you improve the home page?

   ____

7. How would you rate the layout of the tuner page? (1 being bad, 5 being excellent)

   1 2 3 4 5

8. What do you like about the tuner page?

   ____

9. How would you improve the tuner page?

   ____

10. How would you rate the layout of the scales page? (1 being bad, 5 being excellent)

    1 2 3 4 5

11. What do you like about the scales page?

    ____

12. How would you improve the scales page?

    ____

13. Are there any other comments about the application you would like to make?