



Musical tone recognition system for interval tone training on mobile devices

Hilmar Þór Birgisson



Faculty of Electrical and Computer Engineering
University of Iceland
2012

MUSICAL TONE RECOGNITION SYSTEM FOR INTERVAL TONE TRAINING ON MOBILE DEVICES

Hilmar Þór Birgisson

60 ECTS thesis submitted in partial fulfillment of a
Magister Scientiarum degree in Electrical and Computer Engineering

Advisors

Helgi Þorbergsson

Magnús Örn Úlfarsson

Faculty Representative

Anna Soffía Hauksdóttir

Faculty of Electrical and Computer Engineering
School of Engineering and Natural Sciences
University of Iceland
Reykjavik, September 2012

Musical tone recognition system for interval tone training on mobile devices
Development of tone recognition system
60 ECTS thesis submitted in partial fulfillment of a M.Sc. degree in Electrical and Computer Engineering

Copyright © 2012 Hilmar Þór Birgisson
All rights reserved

Faculty of Electrical and Computer Engineering
School of Engineering and Natural Sciences
University of Iceland
VR II Hjarðarhaga 2-6
107, Reykjavík, Reykjavík
Iceland

Telephone: 525 4000

Bibliographic information:

Hilmar Þór Birgisson, 2012, Musical tone recognition system for interval tone training on mobile devices, M.Sc. thesis, Faculty of Electrical and Computer Engineering, University of Iceland.

Printing: Háskólaprent, Fálkagata 2, 107 Reykjavík
Reykjavík, Iceland, September 2012

Abstract

Powerful mobile devices have quickly become almost a necessity for normal persons in everyday life. Smart devices such as mobile phones and computer tablets are able to serve much greater functionality than only communication. Their availability and multifunctional hardware allows them to be an accessible tool for educational purposes. For example, signal processing can be used for interval tone training where audio signals can be recorded, analyzed and presented as graphical data.

The main contribution of this thesis is a new brand of educational and leisure application system for mobile devices. It is a system capable of assisting beginners with learning tone intervals and musical notation. A fully functional prototype system was developed for the iOS platform. It implements real time musical tone recognition which enables it to transform stream of tones into presentable musical notation. The application is easily used to monitor whistling by users in order to see intervals between tones presented by graphical musical notation. It can further be used to monitor users playing musical instruments and present graphical musical notation of simple songs. The system then also offers the possibility of inspecting presented notes for further information. The application system runs on any devices that supports iOS 5.1 including iPhone 4, iPad 2 and later iOS devices.

Keywords

Musical tone recognition, interval tone training, Fourier transform, musical notation, mobile devices, smart devices, iOS, programming, development, optimization, testing, interface design, iPad, iPhone, Labview, Xcode,

Útdráttur

Öflug smátæki hafa á skömmum tíma orðið nánast nauðsynleg í lífi fólks. Farsímar og lófátölvur eru meðal svokallaðra snjall tækja sem þjóna orðið mun meiri tilgangi en aðeins samskiptum. Framboð þeirra og fjölbærni gerir þau aðgengileg og nýttast vel til menntunar. Sem dæmi má nefna að hægt er að nota merkjafræði til að greina hæð tóna þar sem boðið er uppá upptöku, vinnslu og framsetningu á hljóðmerkjum sem grafísk gögn.

Helsta framlag þessarar ritgerðar er ný tegund kerfis til menntunar og tómstunda fyrir smærri tæki. Kerfinu er fært að aðstoða byrjendur við að læra að greina tónbil ásamt því að lesa nótnaskrift. Fullkomlega virk frumgerð kerfisins var þróuð fyrir iOS grunn. Kerfið notfærir sér rauntíma tóngreiningu sem gerir því kleift að umbreyta straumi tóna í frambærilega nótnaskrift. Auðvelt er að nota kerfið til að greina tónabil blísturs og setja fram nótur þess á grafísku formi. Það getur enn fremur fylgst með notanda spila einfalt lag á hljóðfæri og skrifað út nótnaskrift þess. Að auki býður kerfið uppá möguleikan á því að skoða betur hverja nótu fyrir sig fyrir frekari upplýsingar. Kerfið keyrir á öllum tækjum sem styðja iOS 5.1, þar á meðal iPhone 4, iPad 2 og nýrri tæki.

Leitarorð

Tóngreining, tónhæð, Fourier umbreyting, nótnaskrift, farsímar, smátæki, smart tæki, iOS, forritun, þróun, bestun, prófun, viðmóts hönnun, iPad, iPhone, Labview, Xcode,

Contents

List of Figures	ix
Abbreviations	xiii
Acknowledgments	xv
1. Introduction	1
1.1. The Research Problem	2
1.2. Embedded platforms	4
1.3. Reusable code	5
1.4. Optimization	5
2. Background	7
2.1. How music works	7
2.1.1. Frequencies	7
2.1.2. Major factors of sound	9
2.1.3. Notes and notation	10
2.1.4. Tone length	12
2.1.5. Time signature	13
2.1.6. Frequency note mapping	14
2.1.7. Harmonics and overtones	14
2.2. iOS and Xcode	16
2.3. Hardware	17
2.3.1. iPhone and iPad	17
2.4. Fourier	18
2.5. The Fourier Series	19
2.6. The Fourier Transform	20
2.7. The Discrete Fourier Transform(DFT)	21
2.8. Fast Fourier Transform(FFT)	22
2.9. vDSP and Fourier transform	23
2.10. MATLAB and LabVIEW	23
2.10.1. MATLAB	23
2.10.2. LabVIEW	24
2.11. Related Work	24
2.11.1. Guitar Tuners and Related Applications on Mobile Platforms	24
2.11.2. Other Related Software	25

2.12. Musical Instruments	25
2.12.1. Piano	26
2.12.2. Guitar	27
3. Experimental Design	31
3.1. Proof of Concept: Deriving Tones from Sampled Data	31
3.1.1. A sample of the process	32
3.1.2. Note Derived from given Frequency	35
3.1.3. Completing the test System	37
3.2. Deciding on the Development Platform	38
3.3. Planning	39
3.3.1. Use Case Diagram and Flow Chart	39
3.3.2. Product Backlog	40
3.4. The Development	44
3.4.1. Startup	45
3.4.2. First Tests and Changes	46
3.4.3. Interface Ideas	48
3.4.4. Bringing the Idea to Life	50
3.4.5. Adding Tempo, Measures and Bars	52
3.4.6. Improving Notation	54
3.5. Improved Instrument Interface	55
3.5.1. Advancing the Product	56
3.5.2. iPad Improvements	56
4. Results and Analysis	59
4.1. Improvement and Refinement	59
4.1.1. Determing Silence	60
4.1.2. Maintaining Perfect Tempo	61
4.1.3. Harmonics Greater in Amplitude than Fundamental Frequency	63
4.2. The Application	63
4.2.1. Usability and Functionality	64
4.2.2. Computation, Accuracy and Resolution	71
4.3. Comparison	72
5. Conclusions and Future Work	77
5.1. Future Work	78
Bibliography	81
A. Appendix	85

List of Figures

1.1. Popular handheld devices and platforms [25]	2
2.1. WhistleDetection [14]	9
2.2. One way of tablature for guitar instrument(guitar tabs) [10]	11
2.3. 2 different staves which together form a grand staff [16]	11
2.4. C sharp note shown on a staff	12
2.5. Whole chromatic scale with the 13th note that completes one octave [2]	12
2.6. Notes Duration	13
2.7. Staff with time notations and notes shown in 3 measure	14
2.8. Frequencies for an equal-tempered scale [19]	15
2.9. Harmonic partials on strings [24]	16
2.10. iPhone and iPad screen size [4]	18
2.11. Piano keys mapped to notes [16]	27
2.12. Guitar Notes [12]	28
2.13. Guitar notes on a staff [26]	29
3.1. LabVIEW system setup screen	32
3.2. LabVIEW interface and output screen	33
3.3. Time domain plot of sampled note E	34

LIST OF FIGURES

3.4. Frequency domain plot of sampled note E	35
3.5. Frequency domain plot of sampled note E zoomed on X axis for closer inspection	36
3.6. User diagram	40
3.7. Flow chart	41
3.8. First interface	45
3.9. Testing interface	46
3.10. Test interface on iOS emulator	48
3.11. Test interface with note	49
3.12. Proposed interface	50
3.13. Interface idea in development running on iPhone emulator	51
3.14. Guitar neck with 6 strings	55
3.15. iPad interface	58
4.1. Whole-note octave test	65
4.2. <i>Mary had a Little Lamb</i>	66
4.3. Closer look at Figure 4.2(b)	66
4.4. Emulator output of <i>Mary had a Little Lamb</i> input	67
4.5. Overlaid output from four test cases	68
4.6. Overlaid output from four test cases and Reason visualization	68
4.7. Emulator output of <i>Mary had a Little Lamb</i> input with increased amplitude	69
4.8. iPad Application output of <i>Mary had a Little Lamb</i> input	70
4.9. Emulator output of <i>The Legend of Zelda theme</i> input with Reason visualization	70
4.10. SpectrumView [15]	73
4.11. FFT analyzer and setup screen [6]	74

4.12. Pro tuner on iOS [22]	75
4.13. SoundHound on iOS [18]	76
A.1. Product Backlog	86
A.2. Product backlog	87

Abbreviations

ANSI	American National Standards Institute
API	Application Programming Interface
ARC	Automatic Reference Counting
BPM	Beats Per Minute
CPU	Central Processing Unit
DB	Decibels
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
GHZ	GigaHertz
GUI	Graphical User Interface
Hz	Hertz
IDFT	Inverse Discrete Fourier Transform
LCD	Liquid Crystal Display
MHZ	Megahertz
OS	Operating System
SDK	Software Development Kit
vDSP	Vector Digital Signal Processing

Acknowledgments

I want to thank my advisors Helgi Þorbergsson and Magnús Örn Úlfarsson for their very helpful suggestions and support writing this thesis. I want to specially thank Sveinn Ólafsson which came up with the idea for the project, often helped when further ideas were needed, provided support and helped test the system while it was still in early development. I want to thank Háskóli Íslands and Matvice ehf. for providing the equipment and license that were needed to finish the development and deployment of the system as an application on the development platform. I then want to thank my dear friends, Kári Eyvindur Þórðarson for thoroughly proofreading the thesis, Ólafur Björgvin Sveinsson for his assistance in LabVIEW programming, Ægir Örn Ingvason for his musical assistance and provided sound samples and then Davíð Már Stefánsson and Matthías Már Ólafsson which helped encouraging my spirits and always knew better than most people what I could accomplish. My girlfriend Aldís Erna Pálsdóttir and my large family were as always very supportive and none of this could ever have been accomplished if it wasn't for my mother Sigrún Bjartmarz and sister Tinna Björg Úlfarsdóttir which have always supported me in whatever task I undertake. Lastly I'd like to thank my father Birgir Sveinn Bjarnason for his support and all the times I've borrowed his car (let's hope that he's not counting) and my dear little siblings who I know if were older would have made their valuable contribution to support me.

1. Introduction

Analyzing frequencies from sampled sound waves utilizes the same mathematical functions and theories as are used to analyze all types of physical objects that can be considered having waveforms. Heat, light and sound are examples of these objects which are waves that travel through a medium. Being able to analyze these objects is important for the whole of physics, engineering, electronics and even medical science. Using these mathematical functions and theories, various software applications have been developed with the task of analyzing and gathering information from sound-waves.

With the fast pace of technology growth, handheld devices have become almost a necessity for people living in most parts of what is called the first and second world. These devices are becoming more and more powerful and their usage in form of communication, education, work and leisure activities has broadened very fast. All manufacturers and developers compete to keep their devices and platforms updated with big changes and improvements made each year. Alongside this growth a huge market emerged for small application development which relies on these specified devices and their platforms.

From this advance of computational devices which have vastly increased in computation power, their power drain per arithmetic operation reduced, and not to mention their increase in memory size, an opening was made for power-hungry applications to be developed and deployed on smaller devices. Devices that were too computationally slow and small in memory have become powerful enough to perform complicated mathematical functions and rich graphics.

1. Introduction

Tone recognition using mathematical computation and theories has been explored in much detail and can be solved in multiple different ways which offers many approaches to compute valuable information from sampled sound-waves. It was only a few years ago that the first tone recognition software application, that was developed for a multifunctional hand-held device, was released and offered to an open user market. Since then tone recognition software applications for various handheld platforms have become more and more widely spread among various handheld computation platforms. Today most multifunctional handheld devices, that can process sound input and give graphical output from the device to the user, offer some software for tone recognition. Among these devices are the very common smartphones and computer tablets. A few popular handheld devices are shown in Figure 1.1.



Figure 1.1: Popular handheld devices and platforms [25]

1.1. The Research Problem

The research problem is to design and optimize a musical tone recognition system for interval tone training on mobile appliances. Furthermore the system should be able to recognize musical tones and distinguish tones from silence and even filter out unwanted low amplitude environmental sounds. The system's task is to work

with an input, analyze it and give an output that would help its user play the same notes on an instrument. The output should be displayed to users in a format that is easily understood by beginners learning to play an instrument. While designing the system it should be carefully considered that its deployment is strictly meant for mobile small and middle size handheld devices. The resulting system could serve as an helpful tool that does not just offer an activity leisure but also an interesting experience that helps the user reading and writing musical notation and playing notes on an instrument.

The problem is split up into few small parts. With most of the larger mathematical problems already having been explored by other researches and theories, there was not as much research to be found which described how the system should be set up to give the best results to its users. It should also be noted that most of those mathematical research mentioned is not particular to the purpose of this project. What is meant by that is that the system has its computation and power limitation and interface restrictions which have to be taken into consideration. A tradeoff between a well performing user friendly system and a high accuracy system has to be developed to fit the system profile. In conclusion, already known good mathematical computations theories will be used perform the basic frequency analysis, which then will be tweaked to perform well with tone training, and then the output will be developed and changed rapidly with small iterations throughout the system development to improve its usability.

Below is the list of research questions to be answered in detail in chapters 3, 4, and 5.

- Is the frequency analysis problem feasible and is the analysis able to output useful information for tone training?
- Are devices among embedded platforms such as common hand-held smartphones and tablet devices able to perform real-time tone detection?
- Are the devices able to give usable and accurate feedback to users in spite of their limitations?

1. Introduction

- How can we achieve the best results via optimization.
 - Functions and libraries.
 - Mathematics and transformation.
 - Interface and restrictions.

1.2. Embedded platforms

While the tone recognition concept has been thoroughly examined, the same problems have to be dealt with again and again on the many various platforms available. Not only the problem that different platforms for embedded devices use different standards such as code languages, specifications, inputs and outputs from devices but also the problem with hardware limitation among embedded devices. Hardware limitations such as processing power, battery life and available memory. Since it is necessary to take these problems into consideration it can not be expected that the same mathematical functions in an application that will run well on a octal-core processor Unix platform desktop computer with 8-16 gigabytes of fast memory which is plugged into home electricity will also run as well on a embedded device even if the embedded device offers new technology such as mobile dual- or even quad-core processors and other specifications.

In tone recognition it would of course in most cases be recommended to use the most accurate software available without adding tremendous computing demands. This is however not entirely the case when the desired application is real-time tone recognition on a embedded platform. The limitations have to be dealt with and one solution can often be arguably superior to another if it offers less latency or requires less computation power at the cost of some accuracy.

1.3. Reusable code

A smart way to work around the various platforms coding problems is to develop code which can be reused throughout different platforms. Since standards do not have to be the same from one device platform to the next we note that only functions that take input from a device in a certain format, work with that input, and then give output in a certain format to the device again can be used. For each platform and device which the application system is expected to run on, a new set of code must be developed which feeds the reusable code among the platforms with inputs of the right format and accepts again certain output. The interface to the user also has to be redone.

Most multifunctional embedded mobile devices are able to use and work with the common C/C++ programming language after it is compiled with its development suite which is why this project uses that programming language as well as the native platform programming language which the application is developed on. The native platform programming language is mostly used to service the interface and user actions as well as performing the main loop function and controlling the process.

1.4. Optimization

The act of optimizing offers the possibility of big improvements to the systems software and is never an easy task. The power of optimization is used to make some aspects of the software work more efficiently in terms of using less resources, offering less latency to users and improving stability as well as making the interface to the user simple and comfortable.

Optimization of the system should involve two parts. Firstly the system should be optimized for the mobile embedded platform to work well with the limitations set by the chosen platform and chosen embedded device. Secondly that the idea behind

1. Introduction

the system is to especially recognize whistle and humming by its users as well as a few set of chosen instruments. That's why the second part of the optimization process of the system is to reduce the computation required by the embedded device by narrowing down the input frequency range with filters and lessen the possibilities of outputs to users.

For applications that use complex mathematical functions it is very important that they are well optimized for the platforms and devices they are to be run on. Fast Fourier Transform as well as other transformations and functions have been optimized by many platform developers. This helps smaller application developers develop stable and fast applications which if done well might become popular among users. These popular applications will then increase sales of applications and devices that run said platform which again means more users that expect further applications development.

2. Background

This project can be said to span more than one field of work. It can for example be related to signal processing, math, computer programming, software development, embedded technology and music. The main purpose of the project is to develop fully functional software, customized and optimized for its users which runs on an embedded platform. It should also answer the main research hypothesis mentioned earlier in chapter 1.

2.1. How music works

For one to understand the basics of this project it is required to have a certain amount of understanding of music and how it works. In this section all the necessary basics related to music which are considered of value for the project are mentioned and discussed.

2.1.1. Frequencies

Starting with the very basics, frequency is used to tell how many times an occurring event repeats itself per unit time. In SI units hertz(Hz), 1 Hz equals that a certain event repeats itself 1 times a second which is then the frequency of that event.

Changes in air pressure in form of waves are what make up sound that humans and other animals hear. These waves are of crucial importance here, throughout the project their frequency will be mentioned in hertz.

The range of human hearing spans roughly from 20 Hz to 20 kHz, although there can

2. Background

be considerable difference between individuals [3]. With age it is considered normal for the higher end limit to gradually decline. Sounds that exceed these boundaries are not noticed by humans.

More than hearing is of importance to the project, it is also important to take a look at vocal frequency generated by humans. The combined frequency of typical male and female voice is about 80 to 1100 Hz. Further investigation shows that the fundamental voice frequency for men is around 120Hz and for women around 210Hz though with age the mean values change slightly[21]. The fundamental frequency concept will be explained in more detail later in this section.

Attention is also directed at whistling and humming frequencies of humans which differ from the vocal frequency range. Most humans are capable of whistling and often use it to grab attention from others while many people whistle songs for entertainment. A whistle is generated with constant airflow from the lungs and moderation of air by the tongue, lips, teeth or fingers to create turbulence where the mouth acts as a resonance chamber. Figure 2.1 shows an average power spectral density analysis that was performed to find the lower and upper frequency limits of a human whistle using Welch's method [14]. What Welch's method does is that it estimates power of a signal at different frequencies by converting a signal from the time domain to the frequency domain. The y and x axis of Figure 2.1 therefore represent magnitude in decibels (DB) and frequency in kHz. DB is given as a ratio in relation to the reference pressure level of 20 micro pascals (μPa) which is the limit of sensitivity of the human ear in most sensitive range of frequency [23].

From Figure 2.1 it can be easily noted that after around 4.5 kHz to 5 kHz the magnitude starts to rapidly fall. One assumption that can be easily drawn from this is that the human whistle is typically inside the range of 500 - 4.5 kHz although some might exceed these limits. By also adding that whistles above a certain threshold start to sound less like tones but rather more like high noises it can be used to further iterate and shorten the range that needs to be considered of value to the project.

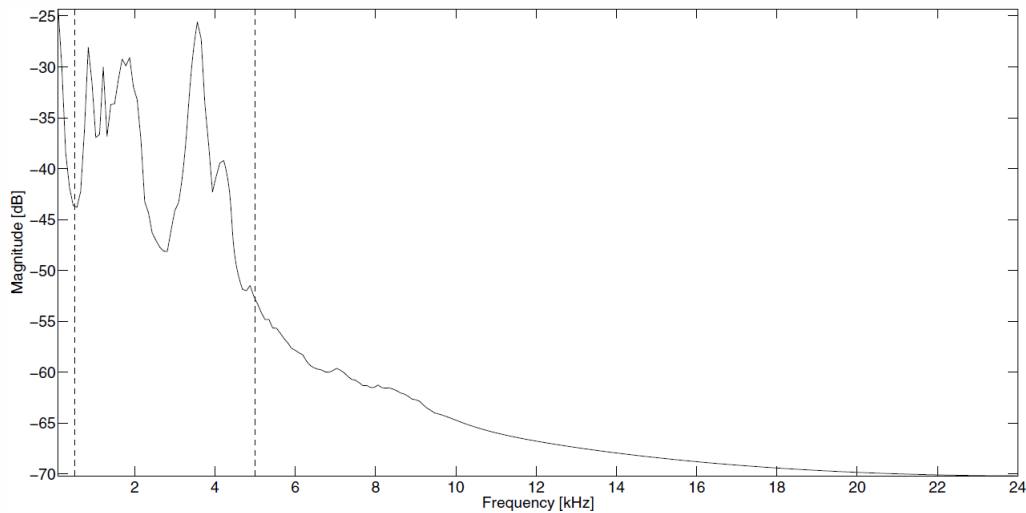


Figure 2.1: WhistleDetection [14]

2.1.2. Major factors of sound

Let's look into how some major factors that make up sound are classified and are to be considered for this project. The factors can be classified in many ways but not all are as relevant to this project as others. The three major relevant factors are: Height, depth and length of sound.

To begin with it is often said that a pitch is the height of sound, that a tone can be high or low pitched. Pitch can be easily confused with frequency, frequency is the cycle rate of the sound wave while pitch is how high or low it sounds when you hear it.

The sound equivalent of depth is harmony. A group of related tones played simultaneously, a chord, gives sound a depth like quality. Tones more related to each other provide a clearer sense of depth than tones less related to each other. Completely unrelated tones combine into noise[3]. This is a major factor of sound and an even greater factor of what is classified as music. Music with only one tone at each time might sound rather dull but it is exactly what is of interest in this project. Going beyond that scope is not of concern here since that would require more than limited time allotted to solve this project.

2. Background

It is said that the sound equivalent of length is beat or rhythm. Beat measures time, the duration or length of a piece of music. Each whistle by the user has its own timed duration. It could be the same as the last whistle or completely different. These various whistles together add up to a rhythm. The projects system is intended to be able to track if one note is longer than another and the notation and tabs will have a way of telling the user how long they should last.

Other factors which can also be used to classify tones and sound such as loudness, duration, color and more are not required to be analyzed by the system for it to perform as intended.

2.1.3. Notes and notation

"Music skill is normal in the human species. Not a rare talent. Most people have the potential to sing and play an instrument with reasonable competence, even if they've never tried. Ability to read or write music notation has nothing to do with it"[3, p.4]. Composing and playing music by ear doesn't require skills in reading and writing notation. It probably wouldn't hurt to know a thing or two but these abilities do not necessary make you a better songwriter. This argument has a good and a valid point but one way or another is needed for an individual to write down the song he's composing, for him to remember it correctly later and/or for others to play. To accomplish this task one needs to have a way of writing the notes down. They don't have to be written down in notation, it can for example simply be noted down as tablature or tab for short. Tab is a more simple form for many beginners who haven't learned how to read and write notation and instead of writing notes as musical pitches a musical notation indicating instrument fingering is used. An example of this form can be seen in Figure 2.2.

In this project some indication is needed to tell the user what note he's whistling. The idea is for indication to be both in notation and tablature simply to let the user

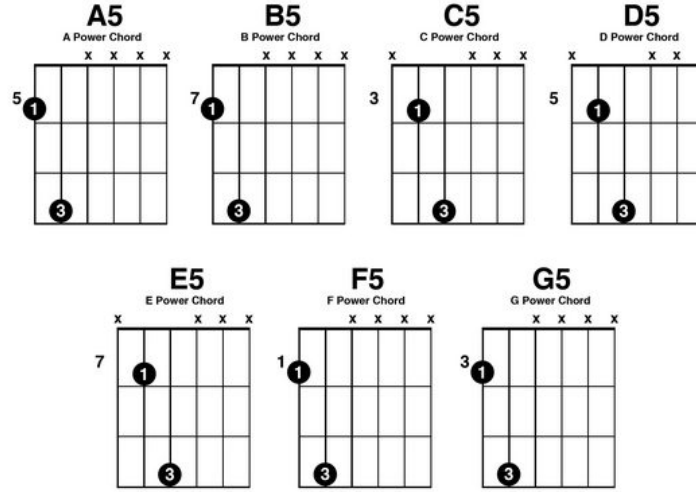


Figure 2.2: One way of tablature for guitar instrument(guitar tabs) [10]

choose which one he wants to use and also for the beginners to get to know them both. The notation and tablature are to be shown to the user so he can easily learn to play each note.



Notation performed in this project aims to help users read and write notes by seeing them in their right notation on the staff in the the project. Figure 2.3 shows all the notes as well as 2 out of the 3 clefs used in modern music notation on a piano staff. The clefs joined together form a grand staff and are used to indicate the name and pitch of the notes in that line. The  or G-glef sets the G4 note as the line that passes through its curl while  or F-clef sets the line as F3 that passes between the two dots of the clef.



Figure 2.3: 2 different staves which together form a grand staff [16]

2. Background

Additionally there are half steps between notes that raise notes by a semitone or half a step. When a note is raised it is written with a sharp, see Figure 2.4. The reason why not all notes have the sharp notation is that having more than one writing for the same pitch makes things more complex. Since for example raising B to B sharp is equal to the note C. When all such equivalences are assumed the whole complete chromatic scale adds up to 12 notes whereas the 13th note completes one octave, see Figure 2.5.



Figure 2.4: C sharp note shown on a staff

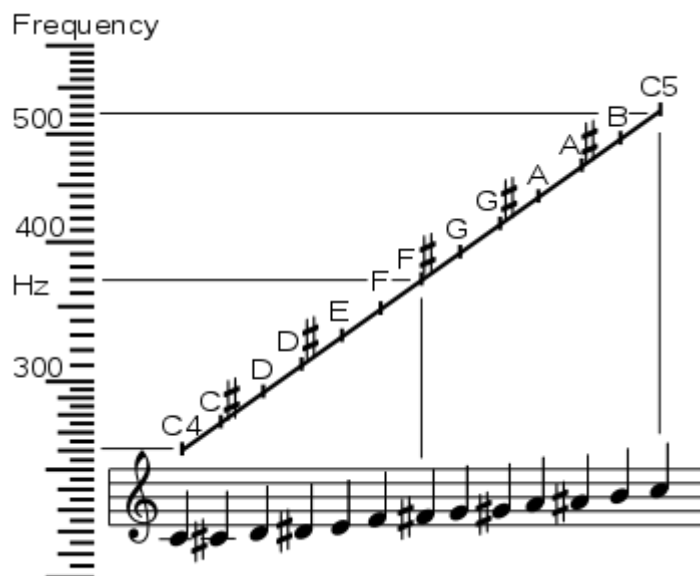


Figure 2.5: Whole chromatic scale with the 13th note that completes one octave [2]

2.1.4. Tone length

The same notes can have a different appearance when written in the most common form of notation. The difference tells the duration of each note which should not be

confused with an absolute value. The duration value is specified in relation to the speed of the beat which is indicated with a time signature which is described in the next subsection. Each note duration is always a simple whole number or a fraction like half or quarter of the beat.

The simplest form of a note is the whole note. The whole note is a hollow circle, then there are eight different durations of basic notes which all have relative durations in the power of 2 in-between them. The first 3 notes are shown in Figure 2.6. The first one is the whole note as described earlier, next one is the half note which is just the whole note with a stem and the last one is the quarter note which has its circle filled up. All these 3 different durations are used in this project to describe how long a note should last in relation to a certain beat.

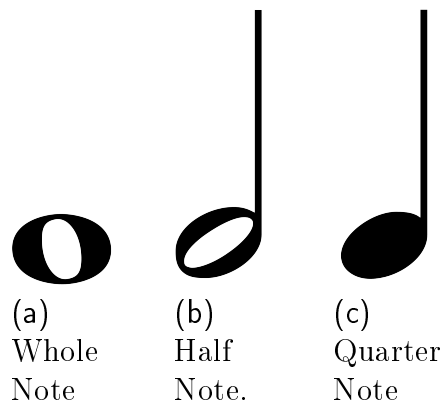


Figure 2.6: Notes Duration

2.1.5. Time signature

The time signature often follows clefs on a staff to specify beats in a measure. Measures are set by these time signatures as a given number of beats that each measure holds, typically of the same length and how many measures are on a staff. The time signature has 2 numbers, one on top and one on below. The top number tells how many beats are in each measure and the lower number how many beats each length of note gets in a measure in relation to the number on the top. As shown in Figure 2.7 a whole note gets 4 beats out of 4 so it's the only note that gets written inside a measure, the same goes for 2 half notes and 4 quarter notes.

2. Background

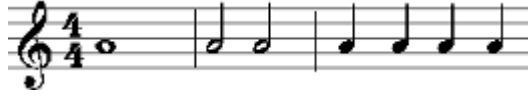


Figure 2.7: Staff with time notations and notes shown in 3 measure

2.1.6. Frequency note mapping

All frequencies that are monitored are mapped to notes. The frequency map showed in Figure 2.8 is created using equation 2.1 [19]

$$f_n = f_0 * (a)^n \quad (2.1)$$

Where f_0 is the frequency of one fixed note which is defined, n can both be positive and negative and is used to tell how many steps the equation is placed from the fixed note, f_n is the frequency of the note n steps away and a is the twelfth root of 2 as $\sqrt[12]{2}$ or $(2)^{1/12}$.

In this system the common $f_0 = 440Hz$ is used which places the A above middle $C(A_4)$.

2.1.7. Harmonics and overtones

A vibrating string on a passive oscillator such as a guitar splits itself so that a fundamental tone and it's harmonics form a single tone to the ear of a listener. Overtones are thus notes produced from different parts of a string which vibrate at different frequencies. These overtone frequencies are always whole-number multiples of what is called the fundamental frequency of a note, making them higher pitched. They're also usually lower in volume. When an untrained listener listens to a single note these overtones are sorted out by the brain and the untrained listener should only hear that single note. Figure 2.9 shows how a vibrating string splits itself and produces a fundemental note along with harmonics.

C0	16.35	A2	110	F#5/Gb5	739.99
C#0/Db0	17.32	A#2/Bb2	116.54	G5	783.99
D0	18.35	B2	123.47	G#5/Ab5	830.61
D#0/Eb0	19.45	C3	130.81	A5	880
E0	20.6	C#3/Db3	138.59	A#5/Bb5	932.33
F0	21.83	D3	146.83	B5	987.77
F#0/Gb0	23.12	D#3/Eb3	155.56	C6	1046.5
G0	24.5	E3	164.81	C#6/Db6	1108.73
G#0/Ab0	25.96	F3	174.61	D6	1174.66
A0	27.5	F#3/Gb3	185	D#6/Eb6	1244.51
A#0/Bb0	29.14	G3	196	E6	1318.51
B0	30.87	G#3/Ab3	207.65	F6	1396.91
C1	32.7	A3	220	F#6/Gb6	1479.98
C#1/Db1	34.65	A#3/Bb3	233.08	G6	1567.98
D1	36.71	B3	246.94	G#6/Ab6	1661.22
D#1/Eb1	38.89	C4	261.63	A6	1760
E1	41.2	C#4/Db4	277.18	A#6/Bb6	1864.66
F1	43.65	D4	293.66	B6	1975.53
F#1/Gb1	46.25	D#4/Eb4	311.13	C7	2093
G1	49	E4	329.63	C#7/Db7	2217.46
G#1/Ab1	51.91	F4	349.23	D7	2349.32
A1	55	F#4/Gb4	369.99	D#7/Eb7	2489.02
A#1/Bb1	58.27	G4	392	E7	2637.02
B1	61.74	G#4/Ab4	415.3	F7	2793.83
C2	65.41	A4	440	F#7/Gb7	2959.96
C#2/Db2	69.3	A#4/Bb4	466.16	G7	3135.96
D2	73.42	B4	493.88	G#7/Ab7	3322.44
D#2/Eb2	77.78	C5	523.25	A7	3520
E2	82.41	C#5/Db5	554.37	A#7/Bb7	3729.31
F2	87.31	D5	587.33	B7	3951.07
F#2/Gb2	92.5	D#5/Eb5	622.25	C8	4186.01
G2	98	E5	659.26		
G#2/Ab2	103.83	F5	698.46		

Figure 2.8: Frequencies for an equal-tempered scale [19]

Harmonic and overtones are very related and differ most when it comes to counting of harmonics or overtones. Even numbered harmonics are odd numbered overtones and vice versa. The fundamental tone is called the 1st harmonic while the 1st overtone is the 2nd harmonic.

2. Background

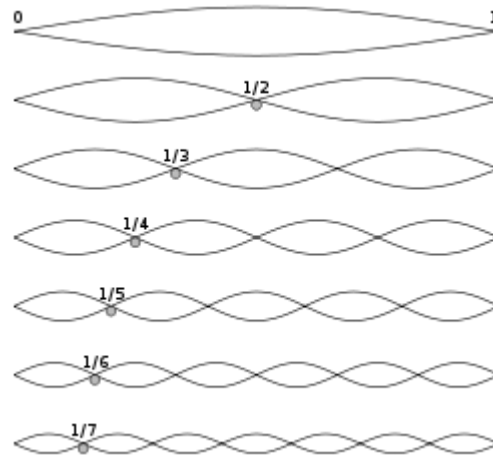


Figure 2.9: Harmonic partials on strings [24]

2.2. iOS and Xcode

iOS is a mobile operating system from the company Apple Inc. iOS was originally released for the iPhone and iPod Touch devices but has now been extended to other Apple devices such as the iPad and Apple TV. Apple does not permit the operating system to be run on other non-Apple devices. iOS is build on Mac OS X which it shares the Darwin foundation with and is therefore a Unix operating system.

Applications for iOS are usually written in the Xcode development environment which is a powerful integrated development environment for creating apps for both Mac OS X and iOS. It includes the instruments analysis tool, iOS simulator and the latest software development kits(SDKs). Xcode enables developers to quite easily design user interfaces for their applications which is integrated with the application code. It also offers testing and debugging of the application in run-time.

Apple's Mac OS X and iOS applications are written in the Objective-C programming language. It is a simple computer language designed to enable sophisticated object-oriented programming. Objective-C is an extension to the standard ANSI C language and it is mostly based on the first object-oriented programming language Smalltalk. iOS does also support code written in C and C++ and can compile it

alongside its Objective-C code. The application programming interface(API) for the Mac OS and iOS are Cocoa and Cocoa Touch respectively. Cocoa touch is used in this project to implement GUI for the application to its user. It provides powerful „base objects“ which can be used to place all kinds of objects in the GUI like labels and buttons with strings and even images.

2.3. Hardware

A few sets of different hardware were bought/borrowed and used in this project. For ease of development and deployment of the project written for iOS a MacMini was used with the latest update of Mac Os X(10.7.4). It supported the latest Xcode(4.3) development environment which enabled easy access to all tools to create, run and test the application on an emulator for iOS(5.1). The computer had an Intel Core i5 2.3 Ghz processor and 2GB 1333 Mhz DDR3 memory.

2.3.1. iPhone and iPad

iPhone and iPad devices were used when the application had progressed and further tests were desired.

- iPad2 with 1Ghz dual core A5 CPU, 512 MB DDR2 1066 Mhz memory and the display resolution of 1024x768 pixels.
- iPhone4 with 1Ghz A4 CPU, 512MB eDRAM and 960×640 display

The main difference between them being the large screen size difference shown in Figure 2.10 which enforces most application developers to create unique set of interfaces for each device.

2. Background

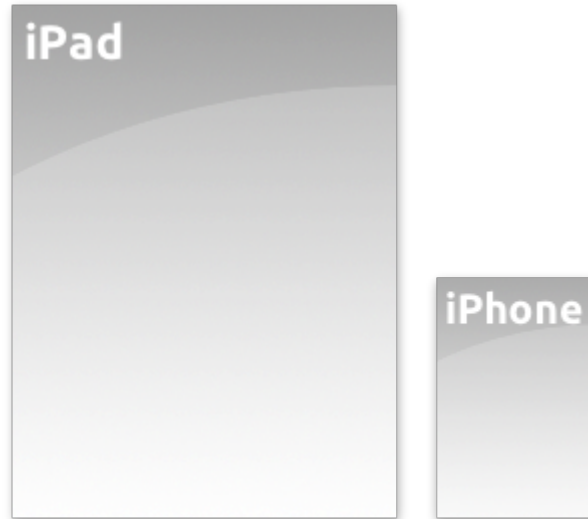


Figure 2.10: iPhone and iPad screen size [4]

2.4. Fourier

Jean Baptiste Joseph Fourier (1768 - 1830) was a brilliant mathematician who believed in the usefulness of mathematics to society. His theory of heat flow led to a new understanding of not only heat but also light and sound. Fourier knew that if he could answer the question how heat varies over time, the answer would be groundbreaking, it would not only add valuable knowledge about unknown properties of heat but expand our knowledge of physics. Joseph Fourier predicted: "Heat, like gravity, penetrates every substance of the universe, its rays occupy all parts of space. The object of our work is to set forth the mathematical laws which this element obeys. The theory of heat will hereafter form one of the most important branches of general physics " [8]. He was not wrong, his mathematical understanding of heat varies helps understanding more about all things that can be described as waves, which includes heat, light and sound. His theory proposed that every wave, however complex it was could be broken up and expressed as the sum of sin waves. The same theory can also be used to create complex waveforms where sin waves are added up using Fourier's mathematics.

The Fourier series and Fourier transform will be briefly described in next sections.

2.5. The Fourier Series

A periodic signal can be described by a Fourier decomposition as a Fourier series, i.e. as a sum of sinusoidal and cosinusoidal oscillations. By reversing this procedure a periodic signal can be generated by superimposing sinusoidal and consinusoidal waves [27]. What it means is that Fourier series can be used to create any periodic signal $f(t)$ in the time domain. This can be done using the sum of an infinite number of sinusoids that are integer multiples of the fundamental frequency f_0 .

Fourier series representation of a periodic function is defined as:

$$f(x) = \frac{a_0}{2} + \sum_{n \in N} a_n \cos nx + \sum_{n \in N} b_n \sin nx \quad (2.2)$$

where $f(x)$ is a periodic function of period 2π for the real variable x which is often written as t to represent time domain, a_0 the amplitude of the direct current component and a_n and b_n are the amplitude of the n th sine and n th cosine function respectively.

The even part is denoted in (2.3) and the odd part in (2.4) and (2.5)

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx \, dx, n \in N, \quad (2.3)$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx \, dx, n \in N, \quad (2.4)$$

$$b_n = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos nx \, dx, n \in N, \quad (2.5)$$

2.6. The Fourier Transform

The Fourier Series is only applicable to periodic signals. For other non-periodic signals the Fourier Transform which comes from the study of Fourier series is used instead. The Fourier Transform is a mathematical transformation used to transform one function into another. It is often used to transform a function from time domain to the frequency domain. The Fourier Transform is defined as: [9]

$$Ff(\xi) = \int_{-\infty}^{+\infty} f(x)e^{-2\pi i x \xi} dx, \quad (2.6)$$

where x is a real number and the function Ff is called Fourier form of function f . The variable t is often used instead of the independent variable x to represent time(in seconds), then the transform variable ξ represents frequency (in hertz).

As said earlier, the Fourier Transform is often used to transform a function from time domain to the frequency domain, while it is also often used the other way around. So under suitable conditions called the Dirichlet conditions, $f(x)$ can be reconstructed from $Ff(\xi)$ by the inverse transform defined as: [9]

$$f(x) = \int_{-\infty}^{+\infty} Ff(\xi)e^{2\pi i \xi x} d\xi. \quad (2.7)$$

Up till now both The Fourier Transform and Fourier Series have only addressed continuous and infinite signals and are only applicable to them. The system which is described by this thesis works on a limited hardware which does not offer storing continuous amount of signal information in its memory nor can the system listen for infinite time to the signals it wants to analyze. The next section therefore describes a transformation for finite discrete signals that can easily be stored in computer memory and the length of each sample broken up and chosen accordingly.

2.7. The Discrete Fourier Transform(DFT)

The Discrete Fourier Transform(DFT) like The Fourier Transform is often used to transform function from time domain to the frequency domain. The DFT requires an input function that is discrete and finite. Discrete functions are usually created by sampling continuous functions. The function being finite means it has to have a limited duration, that it can be a one period of a periodic function or even a windowed segment of a longer sequence.

This means that 2.6 needs to be changed to work with discrete rather than continuous time and frequency domains.

The Fourier integral is replaced by a discrete sum over N samples as:

$$\int_{-\infty}^{+\infty} f(x) \Rightarrow \sum_{n=0}^{N-1} x[n * T_s] \quad (2.8)$$

and the continuous domains are replaced by a sampled equivalent as:

$$f(\xi) \Rightarrow X[k * \frac{f_s}{N}], \quad (2.9)$$

where n and k from (2.8) and (2.9) represent the sample index for discrete time domain and discrete spectral index respectively. Equation (2.6) then can be rewritten as:

$$FX[k * \frac{f_s}{N}] = \sum_{n=0}^{N-1} x[n * T_s] e^{-i * 2\pi (k * \frac{f_s}{N}) * (n * T_s)}. \quad (2.10)$$

Sampling frequency f_s and accordant period T_s have the relationship:

$$T_s = 1/f_s \quad (2.11)$$

2. Background

so equation 2.10 can be simplified to:

$$FX[k * \frac{f_s}{N}] = \sum_{n=0}^{N-1} x[n * T_s] e^{-i\frac{2\pi}{N}kn} \quad (2.12)$$

N , f_s and T_s are fixed and can be considered as constants so the equation can be shortened to a easier form to read as: [17]

$$F(x)_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N}n} \quad (2.13)$$

And the relation that allows to recover the sequence $x(n)$ from the frequency samples is written as: [17]

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{-i2\pi \frac{k}{N}n} \quad (2.14)$$

Which is called the inverse DFT(IDFT).

2.8. Fast Fourier Transform(FFT)

DFT plays an important role in applications of digital signal processing. This includes linear filtering, correlation analysis, and spectrum analysis. Computing DFT directly from its definition often demands great computation time which is not desired when dealing with real time digital signal processing. To solve this problem the Fast Fourier Transform(FFT) was developed as a way to compute these same results much more quickly. A major reason for DFT importance is the existence of efficient algorithms to compute it.

A Fast Fourier Transform is not an independent time domain to frequency domain

transform like DFT but instead an efficient algorithm to compute DFT and its inverse transform. There exists numerous distinct FFT algorithms which involve wide range of mathematics. The difference between computation time can be very significant, for N points using DFT definition it takes $O(N^2)$ operations to compute results while FFT can compute the results in only $O(N\log N)$ operations.

2.9. vDSP and Fourier transform

Fourier transformation in this project using the iOS platform are processed using the vDSP API. The vDSP API provides mathematical functions for computer applications such as speech, sound, audio, and video processing and many other analysis and/or data processing.

The vDSP API provides Fourier transforms for transforming one-dimensional and two-dimensional data between the time domain and the frequency domain. To boost performance, vDSP functions that process frequency-domain data expect an array of complex exponentials (sometimes called twiddle factors) to exist prior to calling the function. Once created, this FFT weights array can be used over and over by the same Fourier function and can be shared by several Fourier functions.

2.10. MATLAB and LabVIEW

2.10.1. MATLAB

MATLAB is a programming environment for algorithm development, data analysis, visualization, and numerical computation. The creators of Matlab even assure users that Matlab can be used to solve technical computing problems faster than with traditional programming languages, such as C, C++ and Fortran. It can be used in a wide range of applications, including signal and image processing, communica-

2. Background

tions, control design, test and measurement, financial modeling and analysis, and computational biology [20].

A lot of software that can be found related to this project is written in MATLAB. This software is mostly intended for research purposes and is not ready to be published commercially on a platform.

2.10.2. LabVIEW

LabVIEW is a system design software that provides tools needed to create and deploy measurement and control system through unprecedented hardware integration. LabVIEW offers comprehensive system design environment, unique graphical programming language, built-in engineering-specific libraries of software functions and hardware interfaces and many other features [5].

2.11. Related Work

Real-time tone recognition has been developed for many kinds of systems and platforms for various purposes. Purpose ranging from displaying frequency spectrum and spectrograms to detecting tones played by a musician to assist in instrumental adjustments.

2.11.1. Guitar Tuners and Related Applications on Mobile Platforms

One very common purpose for real-time pitch detection applications on mobile platforms today is to assist in tuning guitars and/or other instruments. These applications can simply be downloaded, some for free and others for a small price via the devices platform market and can often replace the former guitar tuner embedded

device. The conventional former embedded device had to have an internal mic or a line in and a speaker, LCD screen or other gadgets to give accurate feedback to the user and even more gadgets only to fulfill this single purpose for the user. Yes of course it might be better for some users to have the accuracy that a device specially made for a singular purpose has to offer but it might be more convenient for a lot of other users to spend less and not have to carry around a device that's usually similar in size of mobile devices such as mobile phones today.

2.11.2. Other Related Software

Real time pitch detection has been programmed in many different programming languages and in common software such as Matlab and LabVIEW. Most of these applications use the Fast Fourier Transform function to transform from time domain to frequency domain where the frequency of a pitch is determined. Other applications have also explored tone recognition using different mathematics such as the Wavelet transformation to see if better approximations can be achieved using less computational power.

Tone Recognition Matlab and LabVIEW Software

Real-Time Time-Domain Pitch Tracking Using Wavelets [11] was done by Eric Larson and Ross Maddox. The outcome was a pitch tracker based on the fast lifting wavelet transform (FLWT) using Haar wavelet transform, in Matlab and C++. They developed their software with emphasis on low latency, high time resolution, and accuracy. The implementation used approximations in combination with intelligent peak detection to determine the pitch of vocal samples.

2.12. Musical Instruments

The definition of musical instruments is nothing else than devices created or adapted for the purpose of making musical sounds. All objects that can produce sound could

2. Background

serve as a musical instrument in theory but how well they sound and how easily they can be used to play different sets of sounds may differ a lot between classes of different objects.

To complete the system that's in design a notation that easily and directly tells the user how to play each note off the standard notation on a musical instrument is needed. The musical instruments that were chosen to be presented with the first results were piano and guitar. The reasoning being simply that that they are popular and rather common instruments and have a broad user base as well as being popular first instrument that people play on. Other instruments for example the flute were also considered but will be kept until later development.

Both chosen instruments are classified as string instruments. String instruments produce sounds by vibration of strings. They do have different construction where the guitar has strings that are supported by a neck and is therefore in the lute group while the piano has strings that are mounted on a body and is in the zither group.

2.12.1. Piano

The piano is one of the most common instruments in the world. The common modern piano has a multiple keys, 52 white and 36 black to be precise, which all have different strings related to them. Pressing a key on the piano causes a hammer to strike on a string which makes it start to vibrate. When these strings vibrate they play notes from over seven octaves from A_0 to C_8 .

Mapping the piano keys to notes can be rather easily done and Figure 2.11 shows how a little over 3 octaves are mapped. This is repeated in both directions for all 88 keys. The black keys are the sharp notes in-between which were mentioned earlier in section 2.3(Notes and Notation).

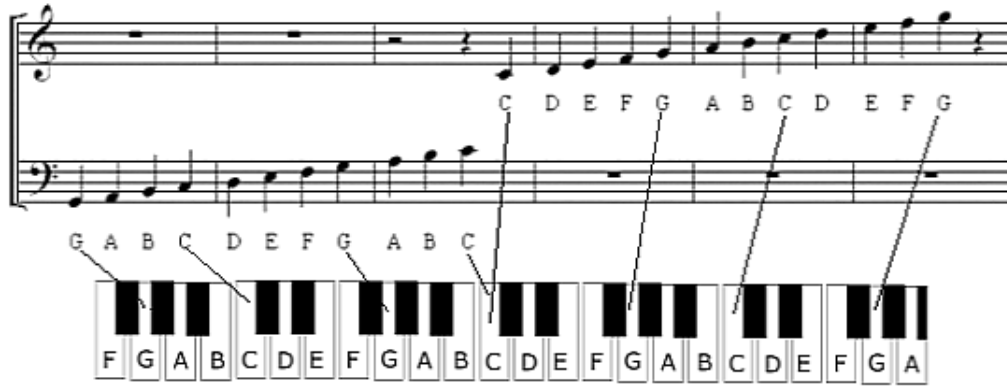


Figure 2.11: Piano keys mapped to notes [16]

2.12.2. Guitar

Guitar is a plucked string instrument which consist of a body with a rigid neck which usually has six nylon or steel strings attached. There exist two primary families of guitars which are acoustic and electric. Both are used in the same way by a user to create sound but each family does it in its own way.

Acoustic guitars have a hollow body that act as a resonating chamber which amplifies tones produced by the vibration of their strings. Electric guitars on the other hand create very low tones by the vibration of their strings since their body is solid instead of hollow. Electric guitars therefore rely on amplifiers that electronically manipulate tones from the guitar.

Mapping finger placement on a guitar neck to notes as the strings are plucked is a little bit more complicated compared to the piano. Figure 2.12 shows a basic set of this mapping. The topmost notes show which notes are played when strings are plucked with open strings(no finger placement). Beneath the open strings frets are counted down the guitar neck which represent half-steps between octaves. 12 half-steps equal to one octave and it should be easy to see that the notes on fret 12 are the same as played on open strings.

2. Background

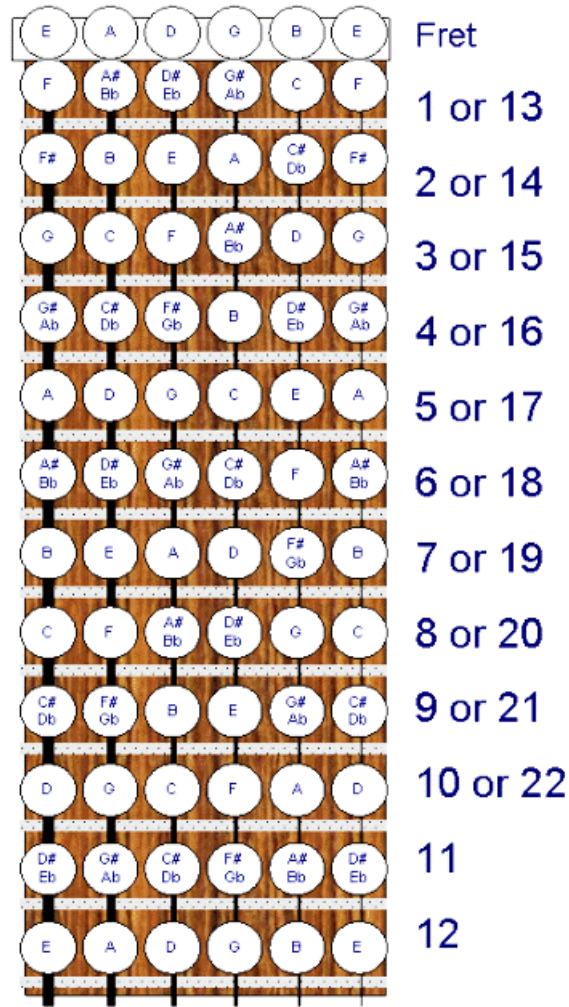


Figure 2.12: Guitar Notes [12]

Figure 2.13 shows how the open string notes and notes half-step away from them relate to the notes on a staff. Inspecting Figure 2.13 can help understanding Figure 2.12, for example it can easily be seen that note E which can be played by the first open string can easily be changed to note F by placing a finger on it's first fret, one half-note up.

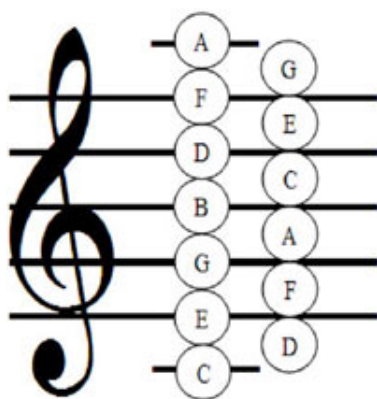


Figure 2.13: Guitar notes on a staff [26]

3. Experimental Design

This chapter goes through the process of creating, designing and implementing the system that is described in Chapter 1. In short:

- To design an application on a mobile hand-held system that processes sound input from users and produces results that assists users in producing the same sound/notes on a musical instrument.

Section 1.1 described a few research problems that were put forth when outlines for the project were being set up. Each problem will be individually explained and solved here in it's own section.

3.1. Proof of Concept: Deriving Tones from Sampled Data

To obtain some kind of proof of concept for the first problem described in Section 1.1 and see if the system that this project describes is really feasible, the first tests in deriving a tone from a sampled data using FFT were done in the system design software called LabVIEW. A program was written using already made program blocks that were connected together to test and see the predicted theory work in action. A finite sample was taken from both a computer data file and a mic recording and used as input in a FFT function. The output from the FFT function was then put through a few mathematical blocks to find the highest amplitude value of its frequency domain representation. This value was then the frequency value of the loudest tone being played in the sample which was the system's task to analyze. Figure 3.1 shows the system setup screen while Figure 3.2 shows the interface and

3. Experimental Design

output screen.

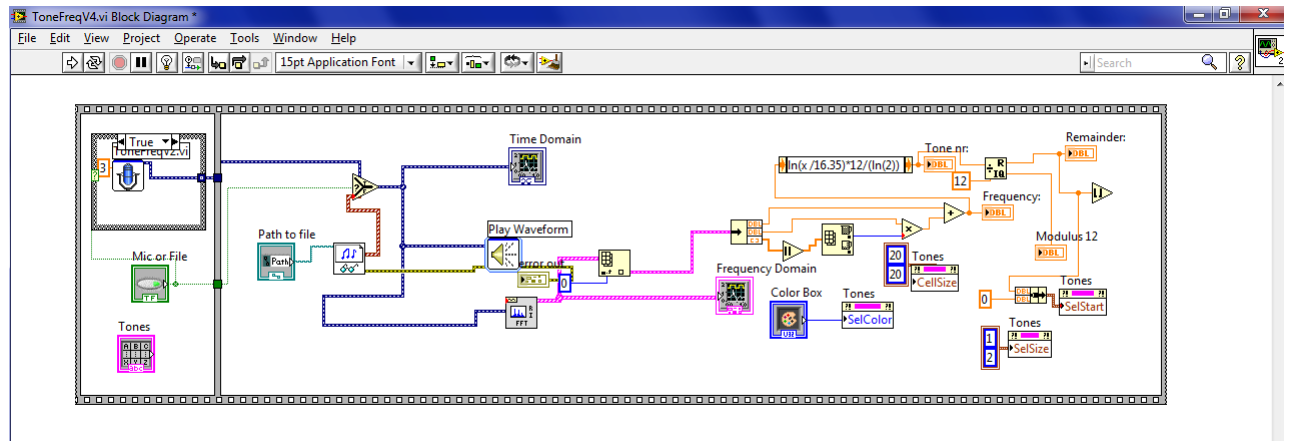


Figure 3.1: LabVIEW system setup screen

3.1.1. A sample of the process

A few samples of piano notes being played without much extra noise were sampled for testing to find out if this rather simple version of the system could indeed tell what note was being played. Let's walk through the process of processing one of those piano notes. The note E was used as a test input which was sampled for roughly 1.5 seconds. The input signal is shown in Figure 3.3

The sampled input was then transformed from the time domain to the frequency domain with a Fourier transform in LabVIEW. The Fourier transformed sample of Figure 3.3 is shown in Figure 3.4 where the concept of amplitude value is not meant in its strict sense as amplitude should not contain negative values. What can be seen on the plot are negative and positive real numbers that the FFT returns. These values are then squared to give only positive values which is then considered as the strict amplitude value.

The main difference between Figure 3.3 and 3.4 being that the former figure shows the sampled time to amplitude on the x and y axis while the latter shows frequency

3.1. Proof of Concept: Deriving Tones from Sampled Data

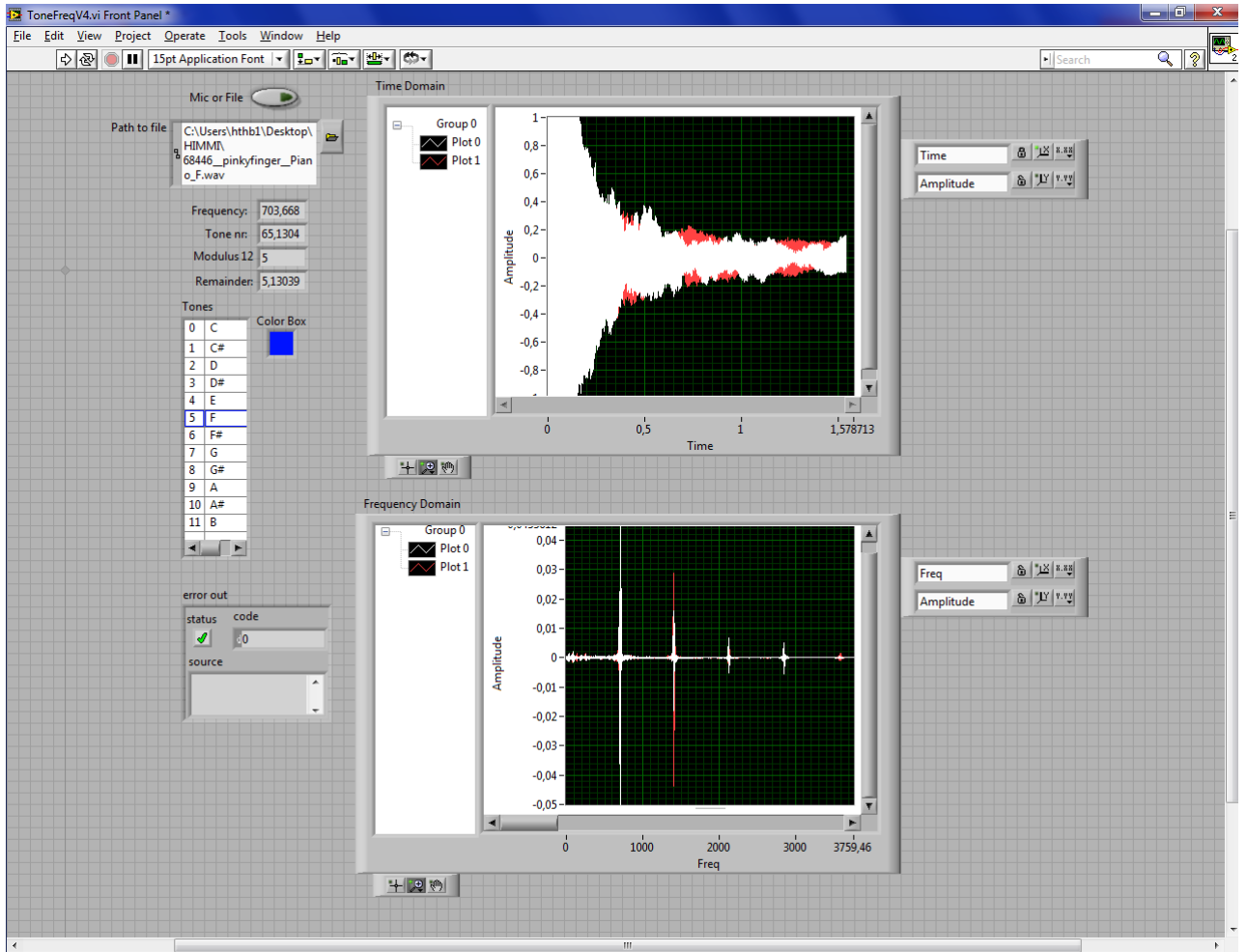


Figure 3.2: LabVIEW interface and output screen

to amplitude. The data used to plot the latter image can be utilized to derive the frequency of the sampled note.

In Figure 3.4 harmonics or so called overtones can clearly be noted. These overtones are described in section 2.1 as the whole multiplication of the fundamental frequency of the note in question. By looking closely at the figure it can be confirmed that these overtones are at least pretty close to where they should be taking their theory into consideration. These overtones don't say much but they can still be of value. The fundamental tone and all the overtones can be analyzed and used to find out what note is being played. Each overtone as well as the fundamental tone is enough on it's own to find which of the 12 notes were sampled. Here it has to be noted

3. Experimental Design

Time Domain

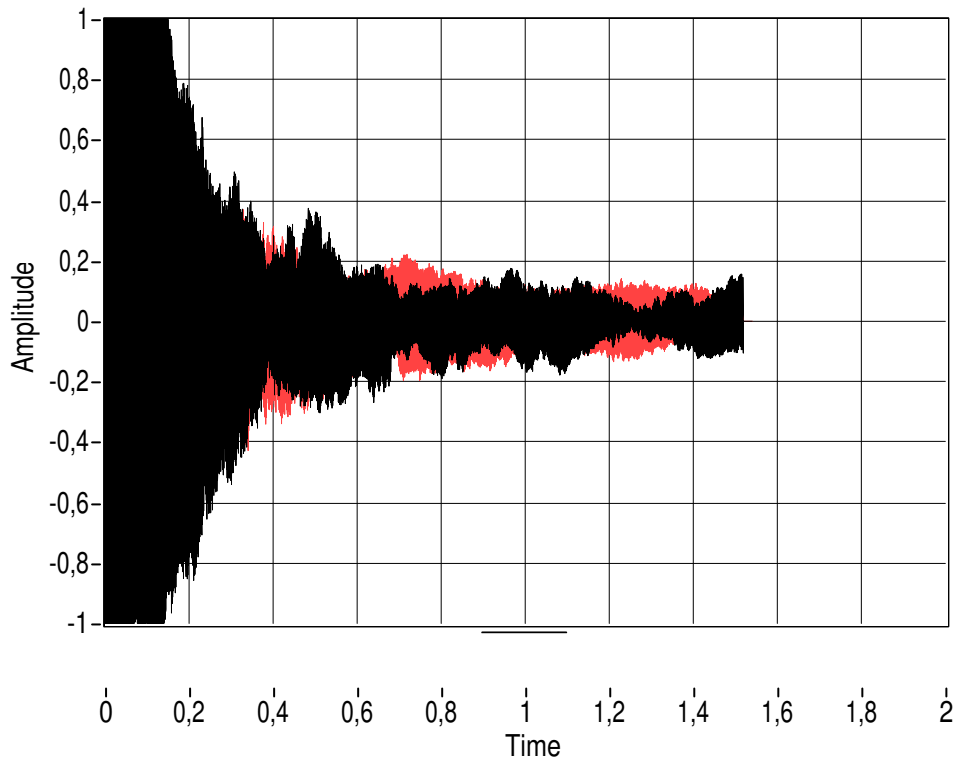


Figure 3.3: Time domain plot of sampled note E

though that this way of thinking can only give the accuracy of which note in an octave was sampled instead of which note of the multiple octaves that exists which is what is desired at completion of the project. What needs to be added later is a way to tell if a frequency with the greatest amplitude in a signal is really the fundamental tone or if it might be a overtone.

A math block in the system scans over Figure 3.4 data for the frequency with the highest amplitude value. If both negative and positive sides of the amplitude are taken into consideration it can be noted from Figure 3.4 that this frequency is around 600-700 Hz where the amplitude goes under -0.06. Figure 3.5 shows a closer look at the highest amplitude of Figure 3.4.

The system then defines this newly found frequency as the frequency value of the fundamental tone. The process that is left is only to calculate the right note from

Frequency Domain

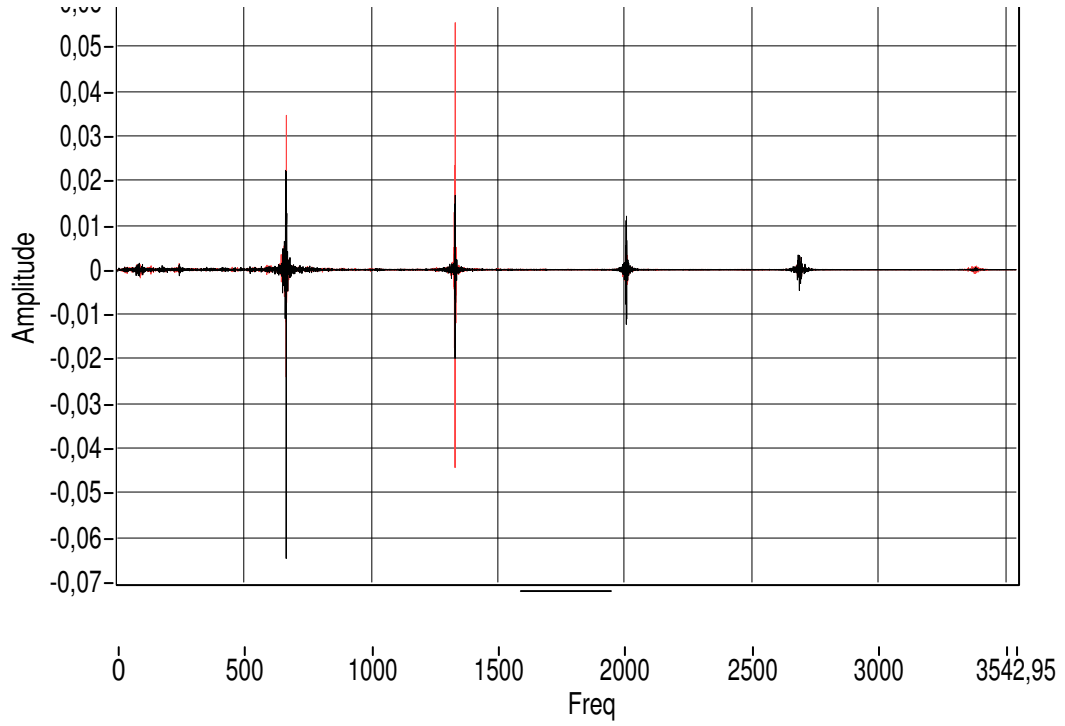


Figure 3.4: Frequency domain plot of sampled note E

this given frequency.

3.1.2. Note Derived from given Frequency

In an octave there are 7 lettered whole notes (C,D,E,F,G,A and B) with additional five lettered half notes (C#,D#,F#,G# and A#). Knowing this, equation 3.1 can be used to identify a note in relation to frequency.

$$f = 2^{(n/12)*N}, \quad (3.1)$$

where f is the analyzed frequency, n is the distance of our note in the chromatic scale from N which is the frequency of a note used for comparison.

3. Experimental Design

Frequency Domain

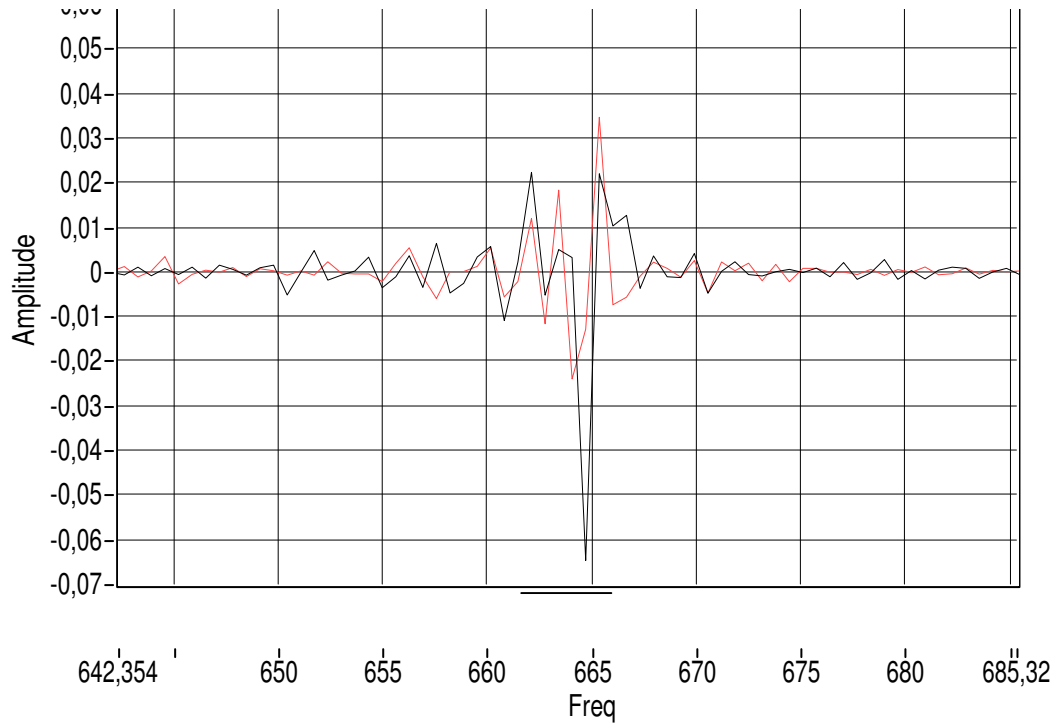


Figure 3.5: Frequency domain plot of sampled note E zoomed on X axis for closer inspection

Using (3.1) with a given constant N , the note for a certain given frequency can easily be derived as n with some small offset. Here 16.35 Hz is used which is the frequency of the first note in the chromatic scale, C_0 , and from there an equation for n can be derived as

$$n = (f/16.35) * 12/(\ln(2)). \quad (3.2)$$

After that it's easy to see which note is being sampled using the frequency that was derived from the frequency representation.

For example the piano sound file, used earlier, which plays the note E gives the

3.1. Proof of Concept: Deriving Tones from Sampled Data

resulting 664,72 Hz frequency which is substituted for f in (3.2)

$$n = (664,72/16.35) * 12/(\ln(2)) = 64.1446. \quad (3.3)$$

Again, knowing that there are 12 notes in each chromatic scale, we can perform modulus 12 on the resulting number which gives the remainder of 4.1446. This means that the note sampled is 4.1446 seats above the first C note in an octave, which is E with a small offset.

What has been solved here is but only a small part of the problem that has been outlined. A note has been derived from a sampled input using a FFT function. Only one note could be sampled per input and no noise filtering or threshold was implemented to see if there was no real input to analyze. Note that this is also not a real time tone analysis.

3.1.3. Completing the test System

A time function was later added to the LabVIEW program which decided when it had sampled enough input values to send them to the FFT function block. With a 22.050 kHz/s sample rate, after 100ms the input had sampled: $22.050 * 0.100 = 2.205$ samples or with a 44.100 kHz/s sample rate: $44.100 * 0.100 = 4.410$ samples. The sample rate could easily be adjusted for different results. With this minor add-on the system had been turned into a real-time tone recognition system which gave feedback to the user every time it had processed new data, telling what note was sampled.

After completing a test system which proved that the idea was indeed feasible a decision had to be made regarding which development platform was to be used.

3.2. Deciding on the Development Platform

Before choosing a platform, multiple factors had to be considered. For example, what kind of audience was being targeted by the application and what the system's basic requirements were. Before development the application was thought of as something for everyone that had at least a small interest in learning to play notes on a musical instrument. Other basic needs for the application were powerful specifications, sound inputs that could be processed and the means to represent a slick looking interface for its users. This meant that the application needed a platform that offered a wide audience and impressive graphics.

Today most of the platform giants offer all of the necessities for the application to be made on their platform. What truly differs between releasing the project on the different platforms could be the numbers of sales and downloads since the user database and how users access the application along with the feasibility of promotion for the application are not the same from one platform to another.

It was relatively easy to narrow the platforms down to only Apple's iOS and Google's Android. The main reasons for other platforms being ruled out are RIM's BlackBerry being mainly targeted to business users, Nokia's Symbian platform development being cancelled and rapidly losing users on the smartphone market and Microsoft's Windows mobile being new and unfamiliar to the developer.

The choice between Google's Android and Apple's iOS was not as easy. Both offer a large user base, powerful devices and really advanced platforms. Android on one hand is free and an open source platform which is taking the lead as the best selling mobile platform among smartphones and also recently started to participate in the tablet platform fighting, and iOS on the other hand which is a closed source platform with a large lead in market share especially if both smartphones and tablets are combined.

A closer look at the application stores, sales and downloads over the last years among these two platforms shows a bigger difference. Android's market share is rising in application availability, sales and downloads but iOS is clearly dominating these statistics and has been doing that since it's launch. Considering this, Apple's iOS was chosen as the development platform.

3.3. Planning

Before starting the whole development process of the system on a mobile platform it was necessary to organize and plan. Creating and maintaining some kind of a plan could really improve development, keeping it rapid and consistent. Planning involved listing and writing down the steps required to carry out the desired goal of the product.

As a result of planning a use case diagram, flow chart and then a little more detailed product backlog containing a prioritized features list were created.

3.3.1. Use Case Diagram and Flow Chart

A use case diagram was made to list up the few actions that a user should be able to perform. The use case diagram is shown in Figure 3.6. As can be seen the actions are somewhat limited but should still include all tasks needed for the system to work as intended.

A flow chart can often simplify complex processes and help with visualizing what is happening at each point in a process and what should happen next. Not to mention that they can also be used to find bottlenecks and flaws. A simple flow chart was done using processing steps (activity) denoted as rectangular boxes, decisions denoted as diamonds and arrows connecting these two. The flow chart is shown in Figure 3.7.

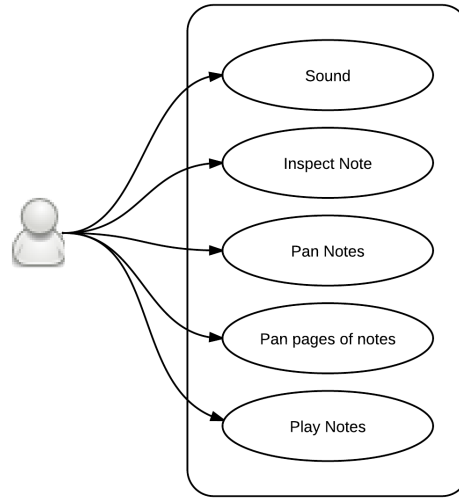


Figure 3.6: User diagram

3.3.2. Product Backlog

The product backlog is a list of features which contain a little description and information about all functionality desired to be implemented in the final product. The product backlog was not meant to be used for SCRUM iterative and incremental agile software development for which it is often used but instead to be used as a guideline to help manage the software project. This was because the product was only being developed by one developer which also was the product owner. No sprints were assigned and risk management was managed internally without being added to the backlog by the solo developer.

Each function is furthermore described in smaller tasks that needed to be performed to implement it. Following some functions are also comments where there was more information and description needed. Also included in the product backlog was the importance of each feature ranging from 1 to 3, 1 being necessary, 2 being important while 3 was thought as more of a fun to have feature in later versions of the product than something that had to be implemented from the beginning. The more important functions of the product backlog are shown in the list below and the full backlog can be found in the appendix.

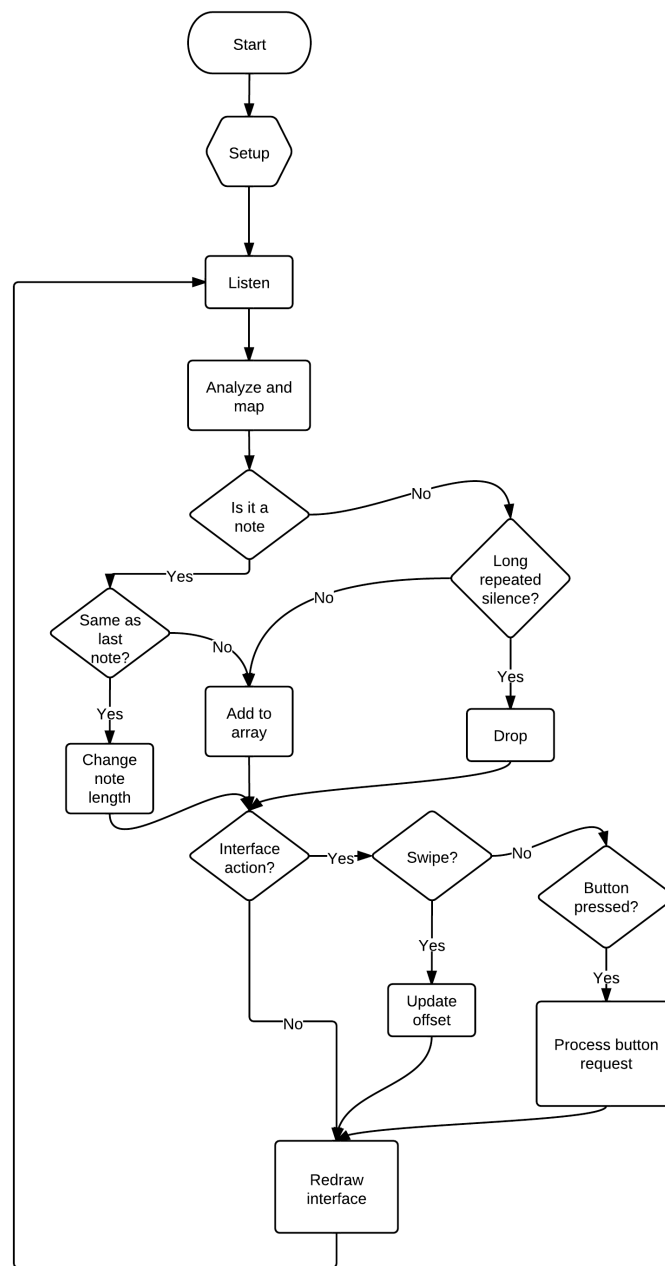


Figure 3.7: Flow chart

3. *Experimental Design*

1. Listen to input (**Imp: 1**)

- Setup audio listener/buffer and audio streams
- Set sample rate
- Start the listener

Comments: Listener should always be active and starts up with the program

2. Transform input with FFT (**Imp: 1**)

- Fill buffer with sampled data
- Run FFT using vDSP function on buffered data

Comments: When desired amount of input has been sampled: perform FFT

3. Analyze FFT transformed signal (**Imp: 1**)

- Find the dominant frequency in buffer

Comments: "Magnitude squared"

4. Store analyzed inputs (**Imp: 1**)

- Create an array for analyzed data and fill it up as the process proceeds

Comments: Possibly store both frequency and note

5. Map dominant frequency to note or silence (**Imp: 1**)

- Create "frequency to notes" map of all desired frequencies/notes
- Create a search function that finds the closest note to frequency in map
- Disregard small noise
- Treat silence differently

6. Design main interface (**Imp: 1**)

- Come up with ideas for the interface
- Discuss their pros and cons

Comments: Interface involving: Staff, notes, tablature and instruments

7. Check for duplicate notes (**Imp: 2**)

- Compare the note with earlier note
- Create a flag that counts how many times the note was repeated

- Create an array that stores multiple occurrences of notes (how many and what note)

Comments: If more than 2 of the same note are together then they should be drawn as a single note with a longer duration

8. Draw notes on GUI (**Imp: 1**)

- Create and place image holders for each note that can be displayed on screen
- Create graphical note and a sharp note
- Position note images on the GUI in relation to their mapped note (move them up and down)

Comments: Different notes have different position on the staff

9. Draw note length (**Imp: 2**)

- Create graphics for 3 different length of notes
- Change note image to represent the length of the note
- Position note images on the GUI in relation to their mapped note (move them up and down)

Comments: Whole/Half/Quarter notes have a different picture depending on their length (how many times they were repeated in a row) Duplicate notes are used to find out length of notes

10. Draw vertical lines between measures (**Imp: 2**)

- Add a bar after every 4 notes

Comments: After one Whole, 2xHalf or 4xQuarter notes there should be a bar (vertical line) that tells where a measure starts/ends (when beat 4/4 is used)

11. Choose a note to inspect and display its value (**Imp: 1**)

- Create a GUI note picker that user can use to pick a note to inspect from the list of sampled notes
- Create GUI display which displays more detailed information about the inspected note

Comments: Users should be able to choose which note they want to inspect from notes on the GUI staff

12. Draw helper points on piano/guitar (**Imp: 2**)

- Draw a dot that displays how to play inspected note on a piano

3. Experimental Design

- Create guitar tabs for oneoctave and display them for inspected notes (iPhone)
- Draw dots that display how to play inspected note on a guitar(iPad)

Comments: Users should be able to choose which note they want to inspect from notes on the GUI staff

13. Pan the staff for more notes (**Imp: 2**)

- Create gesture recognizer
- Adjust movements
- Update the notes drawn in the GUI and also the note picker

Comments: Create gesture recognizer so the list can easily be panned with finger movements and adjust movements so they scroll the list in a user friendly manner

14. Clear values (Reset) (**Imp: 2**)

- Create a clear/reset GUI button
- Clear values/lists/arrays and buffers and hide image holders

15. Upgrade iOS (**Imp: 2**)

- Upgrade the project to newest iOS
- Create a new project
- Import all files and libraries
- Resolve ARC issues

Comments: Upgrade the project from iOS 4.0 to 5.1

3.4. The Development

Instead of building an iOS XCode project directly from scratch it was decided to build on top of another open project written by Demetri Miller which uses iOS Accelerate framework's FFT function for frequency analysis. The project is a good proof of concept for pitch detection and in more details it uses the same functions and mathematics as are described and used earlier in Chapter 3.1. From this the project would be built up on and expanded [13].

3.4.1. Startup

Demetri Millers [13] project had a basic real time pitch detection that had the interface shown on Figure 3.8. A listener could be started that would sample input from a mic (mic that was connected to a personal computer running the emulator or a mic on a real mobile device if it was sandboxed there) and it would process and display the frequency of the sampled data when enough data had been sampled.

As can be noted, Demetri's project relates to the first functionalities that the project described by Section 3.3 Product Backlog wants to offer. The first 3 functions from the Product Backlog are solved and might just need some tweaks regarding sample speed, buffer size, refined frequency analysis algorithm and other minor changes.

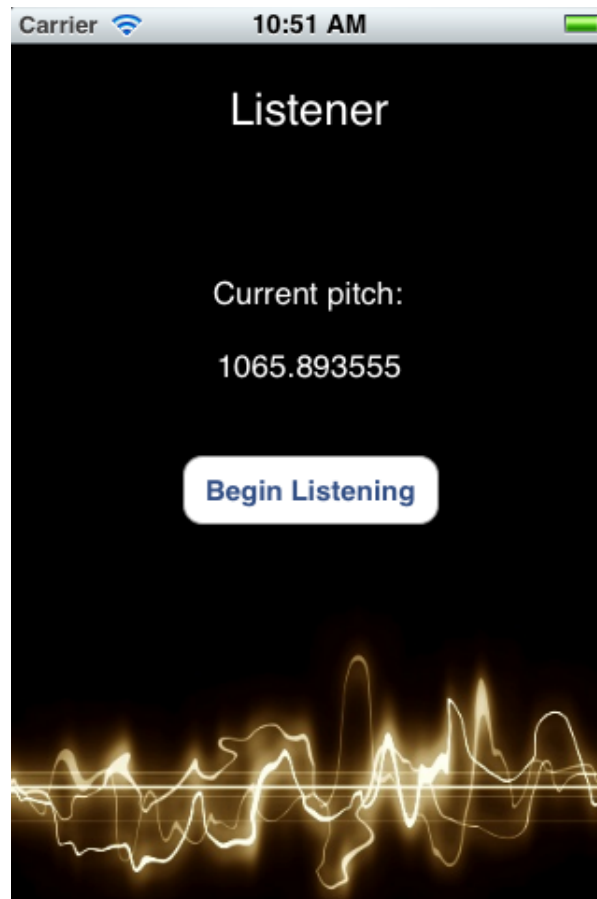


Figure 3.8: First interface

3. Experimental Design

As well as refining, next steps were adding more functionality to the project.

3.4.2. First Tests and Changes

Whenever a new frequency was processed the old one was dropped. An array was created to hold information on old frequencies to be able to display more than one frequency at each given time if more than one frequency had been sampled.

In the beginning the interface was not intended to be used by anyone else than the developer so the original interface was changed to a new test interface for the development system. It was implemented to be able to get the required feedback from an emulator or a device.

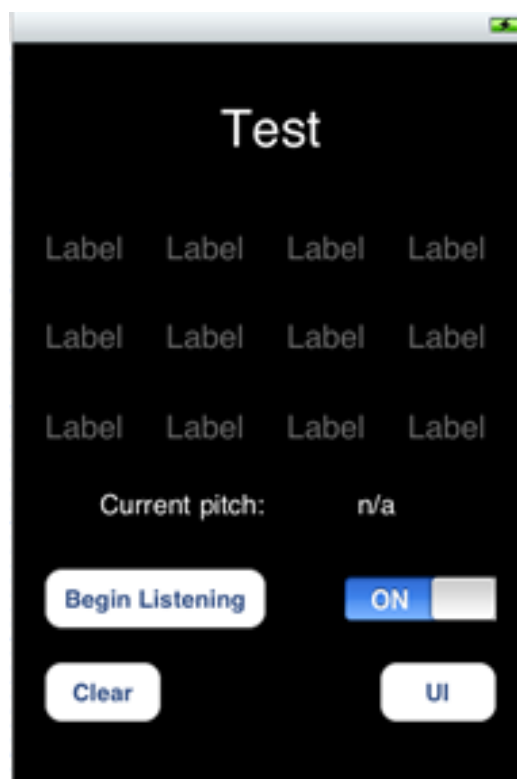


Figure 3.9: Testing interface

Figure 3.9 shows the first testing environment. The first tests of the system were implemented by using begin and stop listening buttons (the *Begin Listening* button changes to *Stop Listening* button after being pressed). This is not in line with the original design as the system should always be listening and know the difference from noise and tones. These changes were implemented a little later in the development of the system and its interface. The figure then shows a few labels which were meant to hold sampled frequencies and display them as decimal values in Hertz. These labels point to and show the value of a position in the array that was created to hold old frequencies.

A function to search for the note closest to each frequency value was then implemented. This was achieved by mapping each note to a frequency after the scale of frequencies for the equal-tempered scale, then performing a rather simple search to see what note had the closest frequency value to the sampled one.

The toggle On-Off button in Figure 3.9 changes the labels from displaying sampled frequencies in Hertz to displaying the actual note corresponding to the frequency for each label. The Clear button clears all labels and the UI button takes the user to a new interface view.

Figure 3.10 shows the interface and the system in emulation on an emulator. The system has picked up a few sampled frequencies which were whistled by a user after the Begin Listening button had been pressed and then stopped again. A threshold was implemented so that frequencies below that threshold were discarded as noise and not added to the array.

Figure 3.11 shows another run of the application where the toggle button is turned On, the frequency labels are now displayed as notes.

3. Experimental Design



(a) First.

(b) Second figure.

Figure 3.10: Test interface on iOS emulator

3.4.3. Interface Ideas

After some improved functionality and the first interface testing it was time to start working on how the fully designed application interface should look and feel like for the user. What had been an idea up till now needed to be implemented into some kind of slick design that would be easy to understand and utilize for different users. Simplicity was the number one priority, to minimize buttons that had to be pressed, settings to be set and actions that were required by the user.

Figure 3.12 shows the first interface idea for a small screen estate phone device that was designed for the application. The application would start up and show the screen that can be seen on Figure 3.12(a). It would right away start listening and after picking up some tones the interface would start to look more like Figure 3.12(b).

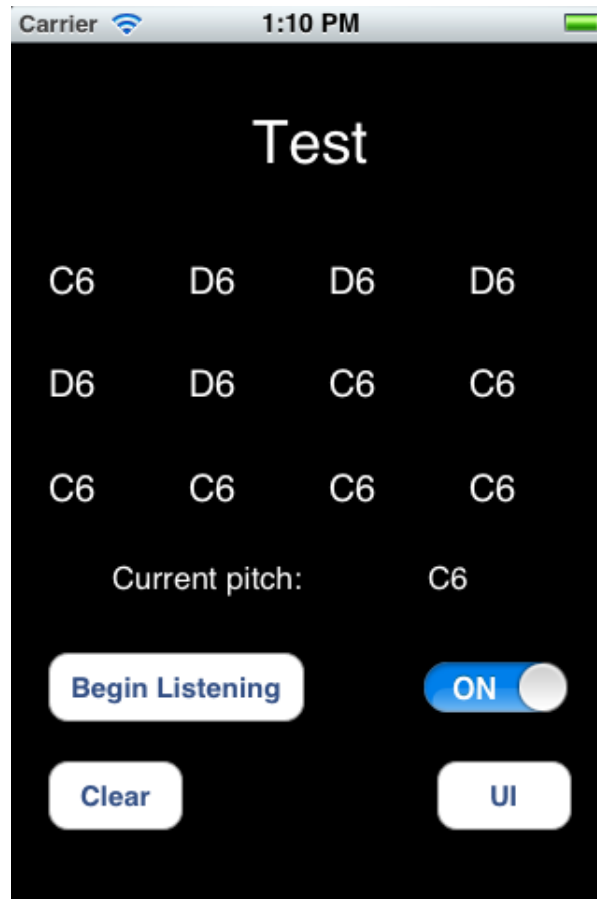


Figure 3.11: Test interface with note

The interface can be described rather easily, at the top there is a placeholder for the application logo. Next there is a slider to indicate what note the user is inspecting. Then there are two note staves with notes that have been analyzed and mapped. These notes should be in their right place considering correct musical notation. Next there are three guitar tabs, the middle one showing how the note that is being inspected is played on a guitar and the other two show the next and previous note from the array holding the notes. Finally on the bottom there is a simple piano that shows what piano key corresponds to the inspected note.

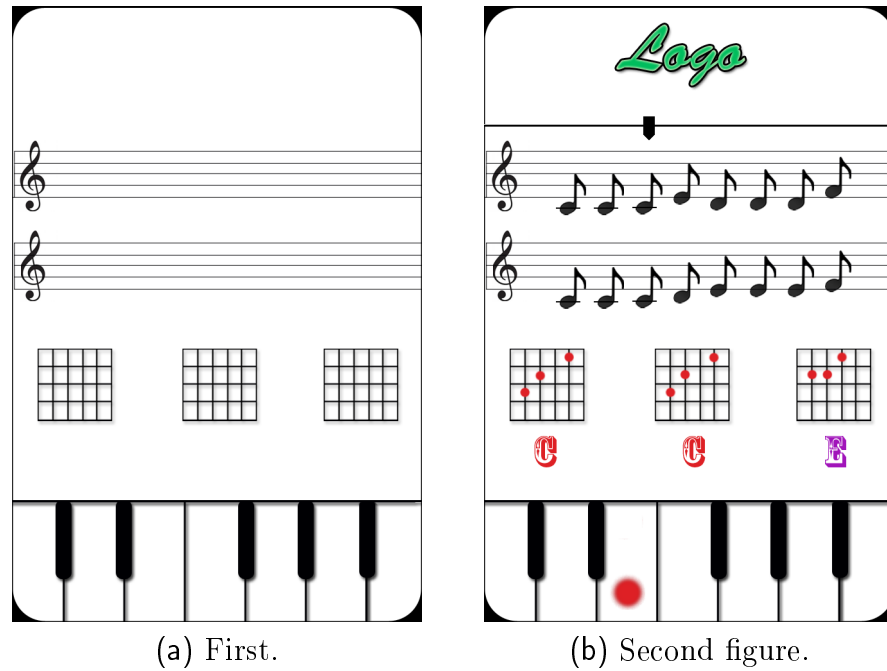


Figure 3.12: Proposed interface

3.4.4. Bringing the Idea to Life

The idea had taken form, something that could be worked with and used as a guideline. This was not a very refined form but it was a beginning.

Improvements and changes were made as well as the addition of new functionalities to the project. The work in development is shown on Figure 3.13.

A list showing short description of changes that were made:

- Figure 3.12(a) was set as interface background.
- Image holders for note images were added to the interface.
- Images for notes were added to the project.
- Notes were displayed on the background staff after they were sampled, processed and mapped.

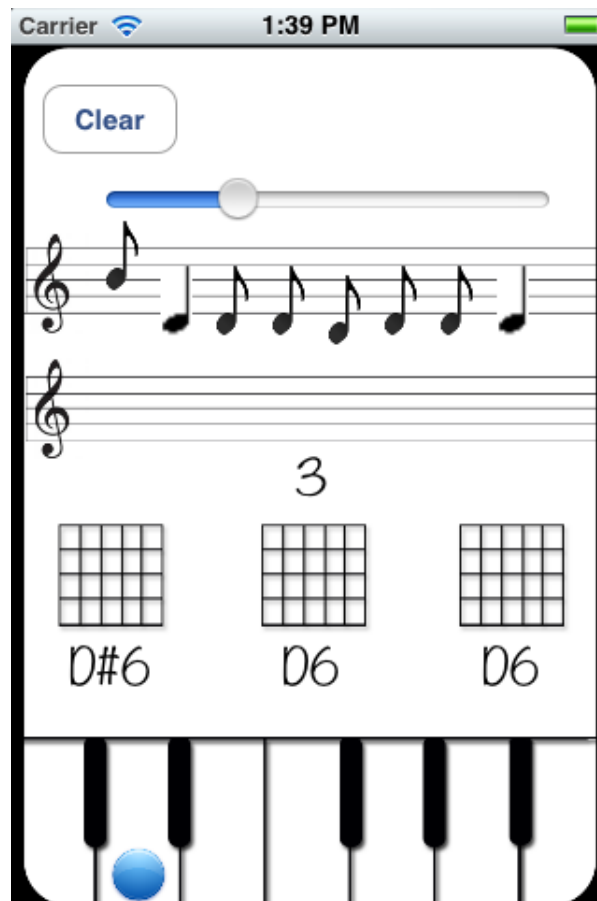


Figure 3.13: Interface idea in development running on iPhone emulator

- Notes were moved up and down the staff in relation to their mapping process.
- A selector was added that could be used to perform a selection inside the list of notes.
- A new label was added to indicate what note was selected (being inspected)
- Gesture recognizer was added to pan the list of notes when more notes than image holders had been added.
- Three labels for notes were kept and others were removed. These labels were coded to show the note that was being inspected as well as the next and previous note in the list.

3. *Experimental Design*

- A dot was shown on the piano background when a note was being inspected. The circle was placed in relation to the mapped note.
- Begin/Stop Listening button was removed and instead the listener was started (never stops) as soon as the program had performed setup and was ready.
- UI button was removed (for now there was only going to be a single interface).
- ON/OFF Switch was removed.
- Sharp notes were shown with a different image than normal note.

This was the status of the project in mid development. The notes on the staff were still far off from having correct musical notation. No beat/tempo had been added and notes were only put on the staff in relation to one octave. This lead to for example C5 and C6 having the same place on the staff without anything telling the user, except inspecting it further if it was C5 or C6. Also no bars between measures were being created. Without these improvements to the project it would be more confusing then helpful for new users which was obviously not the intention.

3.4.5. **Adding Tempo, Measures and Bars**

The musical notation had to be improved, it was incorrect and confusing. Adding a tempo could improve it a lot by giving each note a meaning. The decision was made to add a simple tempo of 80-160 BPM (beats per minute) with $\frac{4}{4}$ to the system where each measure would count 4 beats. This meant that 4 quarter notes would get placed inside one measure between two bars and 80-160 quarter notes would be written per minute. 80 notes showing only 1.333 updates per second while 160 shows 2.666 updates per second.

The system sample speed and buffer size used for FFT had to be considered here as they directly affected the tempo. Common sample speed for systems like this are 11.025 kHz/s, 22.050 kHz/s or 44.100 kHz/s depending mainly on what frequencies

need to be sampled. The Nyquist frequency is the highest frequency that can be expected to be found in a presented sample which can only be half of what the sample speed is. Therefore the frequencies to be sampled can not surpass 5.5125 kHz, 11.025 kHz and 22.050 kHz in relation to the former mentioned sample speeds. Nyquist frequency of 11.025 kHz is considered to be enough for this system to work and to be used for most notes so sample speed of 22.050 kHz/s or 44.100 kHz/s could both be used.

The amount of frames in a buffer that is then fed to the FFT function has to be in the power of two and can be used to give the system a tempo in relation to the sample speed. This of course also affects the accuracy of the system but in terms of such great sample speed and a buffer that is not too small the accuracy should not be too low for the system to perform well. The formula for this accuracy is simply the sample speed divided by frames.

To achieve roughly around 80-160 BPM, a frame size of 8192 and 16384 (both in power of 2) could be used for the 22.050 kHz sampling speed. This resulted in $22050/8192 \approx 2.617$ which gives 60 seconds $\times 2.617$ beats ≈ 161.499 BPM while $22050/16384 \approx 1.346$ which gives 60 seconds $\times 1.346$ beats ≈ 80.750 BPM. Both could be used and gave each a very different experience to the user. The former resulting in a rather slow update of notes on the interface while the latter was 2x as fast and felt like a better setting for now.

A tempo and a time signature had now been picked which set rules that the rest of the system must follow to achieve correct musical notation. Every four beats (a measure) should be separated by a bar (a vertical line). The measure should never hold more or less than these 4 beats, be they 4 quarter notes, 2 half notes, 1 whole note or other part notes.

The bar got implemented by adding extra "Bar Note" after every fourth sampled note. It was then drawn on the screen with the same function as all other notes but as a vertical line over the staff. With this rather easy implementation the bars get

3. *Experimental Design*

treated just as a note which simplified things for coding and without inducing any extra problems.

3.4.6. Improving Notation

Whistling a long note which kept it's frequency roughly would result in 2 or more quarter notes of the same note being shown after another. A better notation would be to take two quarter notes together inside the same measure and write them out as one half note. Then if 2 half notes of the same note had been sampled inside a single measure they'd be combined and written as one whole note.

This was achieved by implementing a function that checked for duplicate notes and stored how many duplicates each note had in a vector. The function simply checked if the note that was analyzed was the same as the one analyzed before it. If it was the same it gave a flag that indicated that a new note should not be added but instead a value in the duplication vector should be increased. Duplication could not occur right after a bar was placed since the sampled note could of course not be a duplicate of a "Bar Note" that got added after every fourth note. The duplicate vector was then used to know how long a duration each note should have ranging from 1 to 4.

Up until here in the development process only a small portion of notes had been mapped. Adding more notes could improve the functionality of the product. A lot more notes were therefore added to the note map to achieve a much bigger note range that could be sampled and analyzed. The system could then analyze notes ranging from C4 to C8(261hz - 4186 Hz). Inside this range are for example the most common whistle frequencies.

Since so many more notes were added the musical notation had to be changed in some way. Only around 2 octaves could fit on a single staff on the interface. This is where treble and bass clefs came in. By implementing 2 staffs, one for treble clef and one for the bass clef more than 4 octaves could sit rather comfortably on

these 2 staves. This meant that notes with very different frequencies could still sit together on the same interface without having special representation for each of them detailing their difference. This is often used in piano notation whereas one hand plays the bass while the other plays the treble.

3.5. Improved Instrument Interface

The first idea to display guitar notation for users was to display a simple guitar chord for the note that was being inspected. It looked good since it used up so little screen estate and could therefore also, for example, be used to show the guitar chord for the next and previous note of the inspected note. After a little bit of inspection the idea of having a guitar chord did both not fit well into the concept of learning basic notes on a guitar, as well as how would the application know which chord should be picked with each note. This would only lead to more misunderstanding and problems.

A new idea developed instead which was to display a guitar neck that could hold a whole octave of notes (12 half notes or 12 frets). Then each note could be displayed as a dot on the guitar neck which would indicate users where to put their finger and what string to pluck. An indicator could then be used to tell if the note was a whole octave lower on the guitar neck to save screen estate. A proposed visual can be seen in Figure 3.14.



Figure 3.14: Guitar neck with 6 strings

3.5.1. Advancing the Product

The code that was built upon from Demetri Miller [13] was written and updated more than year from when the development of this project started. It was written for earlier versions of iOS (4.0) and only with emphasis for the iPhone device. To continue even further with the project it was thought to be ideal to update the project and include full support for both iPhone and iPad devices.

A new fresh iOS 5.1 project was created with full support and a different interface storyboard for both iPhone and iPad devices. All necessary files from the old project were then copied over to the new project and all vital libraries linked. The biggest problem that came up was that the old code had done all memory management manually. In iOS 5.0 ARC (Automatic Reference Counting) was introduced which made memory management a job that the compiler did. Having manual memory management in the code along with ARC trying to work with memory management brought up a lot of conflicts. Either the code had to be fixed in many cases or ARC disabled.

It was decided that ARC could improve much of code that was added by this project to Demetri Millers project. Not much effort was spent thinking about retains and releases of calls mostly since no refinement had been done on the code. The manual memory management was therefore discarded and changed/fixed to work with ARC.

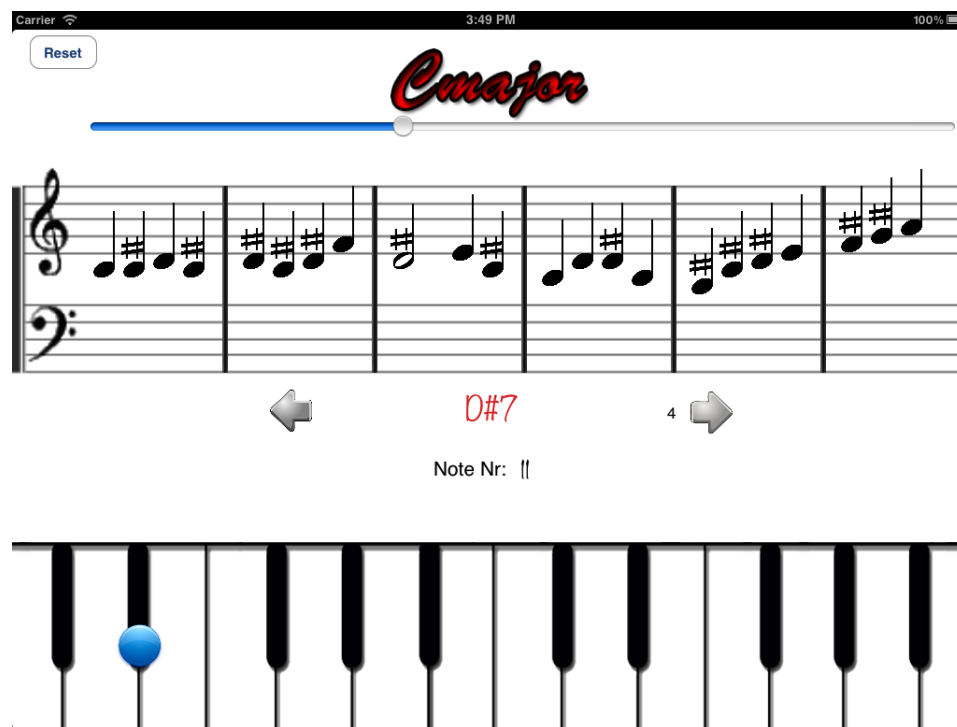
3.5.2. iPad Improvements

Much larger screen estate on iPad called for changes to the interface for that device. The best possible solution was to have 2 different interfaces for the same project, one for iPhone devices and one for iPad devices. From the experience collected so far it was decided to only change it a little from what it had already evolved into for the iPhone, this was also positive since users would still feel like they were using the same application if they had already learned to use one of the interfaces and

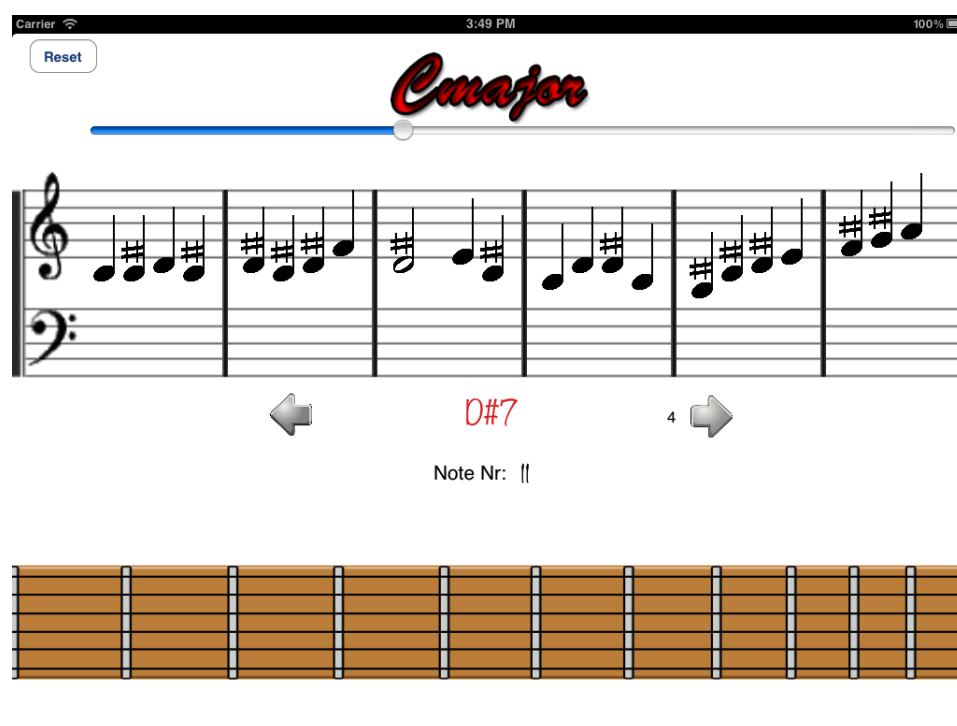
changed to the other.

The same program code was still handling the interface so all former functions were included. Since the iPad did not have the same very limited horizontal screen size as the iPhone, rotating the interface to a landscape orientation felt like the right thing to do and made the staff much longer, fitting more notes on it. The instruments could also be made bigger and 2 octaves could easily be shown on the piano. Then the idea came up that the interface would show either piano or guitar and offer notation for them, but not both at once. Showing both at the same time made the interface just more crowded. To swap between the piano and guitar a simple swipe up or down was sufficient. The new interface is shown on Figure 3.15.

3. Experimental Design



(a) iPad interface with piano



(b) iPad interface with guitar

Figure 3.15: iPad interface

4. Results and Analysis

Chapter 4 will be used to present improvements and refinements as well as results and analysis of the application system which was described, designed, implemented and advanced in Chapter 3. This chapter will state why further improvements which were not planned from the start of development, were necessary for the system output to be as was desired. How the the system was refined to fix issues that were not thought of until they were noticed during the testing phase, and then the system will be thoroughly tested and documented and its usage demonstrated. Accuracy, performance and usability of the system will be described as well as compared to other applications. Different performance and functionality will also be discussed which results from tweaking computation factors and changing the user interface.

4.1. Improvement and Refinement

The design and development of the system had been done and the result was a system that seemed to work as intended. It was an application that listened to sound which was then analyzed and output in the form of notes on a staff. It could tell apart 4 octaves of notes (48 notes) and from what simple test had been performed by whistling and playing single notes on instruments, the system seemed to work well. Notes were being drawn in their right place on the staff, they could be inspected to see their detailed information as well as to see how they could be played on instruments, the staff could be panned back and forth for more notes and all seemed well. A few optimizations and adjustments could do the system good but were not considered necessary at this point.

But some further user tests led to light very unfavorable results. A few issues came

4. Results and Analysis

up which clearly had not been thought of until they were encountered during larger test cases. When notes were played back to back some very noticeable faults could occur as well as other minor problems. These faults will be described in the next subsections as well as how they were addressed.

4.1.1. Determining Silence

Determining if a sampled batch of input should be treated and analyzed as a note had been implemented in a very simple manner. Firstly a sample was treated as a silence when no input was monitored and secondly if it was cut off by a high pass filter with a low frequency threshold. The filter cut off all low frequencies (below 250Hz) and treated them as if there was silence. It was considered as a very simple and quick way to solve this problem but of course it was likely that it might have not been the most efficient one.

The former described method worked well and was acceptable when the system was being testing and played with while it was run on a emulated device with a headset mic as the input sampler. What had not been taken into consideration was that different sets of devices pick up different noises and low sounds. For example, the mic connected to the personal computer that sand-boxed the application on a emulator clearly already had some noise canceling implemented as it did not pick up much environment sounds and provided very nice results with only the high pass filter. Then when the application was deployed on an iPhone device or an iPad device they proved to be much more sensitive to picking up environment sounds. Sounds that were not meant for the application to process such as speech from a small distance and other environmental sounds were being analyzed and shown as notes. This was not a desired functionality and rendered the application almost unusable unless it could be improved by developing a noise filter and/or using the application in a more quiet place.

It was not acceptable that the user had to be in a very quiet room well away from other sounds and that the application could not be running without sampling

and analyzing every low amplitude sound it could pick up. It had to be fixed. The quick fix which was added to improve usability on the mobile devices was a check in amplitude value. Basically another high-pass filter was implemented after finding the dominant frequency which checked if its amplitude was high enough to be considered as something that should be analyzed. Instead of cutting out low frequency valued samples like the former filter, it cut out samples that were low in amplitude. Hence these two filters mentioned were the new determination of samples that were considered as silence.

4.1.2. Maintaining Perfect Tempo

If inputs were at a faster or slower tempo than the application's system tempo a few very noticeable faults could occur. Change from sound to silence inside the same buffer feed to the Fast Fourier Transform would result in silence not being registered. Therefore if two to four tones with the same pitch were whistled fast with small silences in between they could still get turned into a longer single note just as would happen if a long single note had been whistled instead. Also, if a fast change in tone pitch was being monitored, two or more frequencies could be inside the same buffer feed and result only in having the frequency with more amplitude being shown as a resulting note. This had not been treated as a high risk until it was shown having too big of an impact on results. This had to be tweaked one way or another. By holding the right tempo these faults could be minimized but it proved to be a very hard thing to do. It often made small songs result in a very poor output of notes that didn't fit as they should.

Possible fixes for these faults that came to mind were that the system could be improved by finding silences inside buffers that also contained higher amplitude values which could possibly also give two or more tones as a result. Or could maybe the easy possibility of adjusting the tempo be enough? Results from tweaking the tempo were that no matter what tweaks and changes were made the results only came out similar or worse. The tempo speed went to being too fast or too slow and was always hard to follow without faults.

4. Results and Analysis

Completely analyzing the buffer for low time gaps that could possibly be silences was not considered a possibility here as it required a big set of calculations, tests and tweaks for different factors which could take too long to design and could have some impact on system performance. While not surrendering to these usability faults another possible idea was born. What if the underlying tempo of the system was speed up while the output to the user was still given at the same tempo? This meant a few FFTs were done for each note that was displayed as a result. Each FFT could hold different notes and/or silences. Then they would all be analyzed together and turned into one single note output.

The idea was implemented by speeding up the system analysis by the factor of four. Each four results were then monitored and the new output took all these values into consideration. By doing this, short silences in between notes could hopefully be monitored as well as fast change in frequency.

A small sample of the new process which results in an output of two notes:

- Four samples give an array of $[74, 74, \textit{silence}, \textit{silence}]$
 - 74 representing the 74th note on the frequency list for equal-tempered scale and *silence* tells that whatever input was monitored got cut out by either one of the implemented filters. The array is then analyzed and the corresponding note for it is D6 with a raised flag indicating that silence was also monitored.
- Next four samples give an array of $[74, 74, \textit{silence}, \textit{silence}]$
 - again the output should be D6 but since there are silences monitored the older D6 note does not just get longer but instead a new note is drawn which means that the user should play two shorter notes instead of a longer one.

4.1.3. Harmonics Greater in Amplitude than Fundamental Frequency

Harmonics with a greater amplitude than the fundamental frequency cause the application system to display notes octaves higher than they are. For example C4 could be displayed as C5 or higher. The system would still always show C but a higher C could get analyzed instead if it had greater amplitude than the fundamental frequency.

A fix could be implemented by always checking if there was a clear but smaller amplitude monitored in half the frequency of what was analyzed. If there was indeed some amplitude value there it should be further checked if it also had amplitude in half of the new frequency value and so on until a decisive fundamental frequency had been found.

This problem did not affect the test results as much as the previous problems and is kept as is until further advancement of the system.

4.2. The Application

The application had been developed with its original intended functions and its two different sets of interfaces, one for iPad devices and another for iPhone devices. The underlying source code for both interfaces is as said before the same Object C source code which implements and uses libraries and other code classes to perform mathematical operations, mapping, drawing and other functions.

In the main Object C class that represents the interfaces are functions that detect which hardware it is being deployed on and in each case define a few constants and run a few different setup configurations as well as displaying the right interface. Both interfaces will now be discussed as well as their usability monitored. The source code will though only be addressed as a single application.

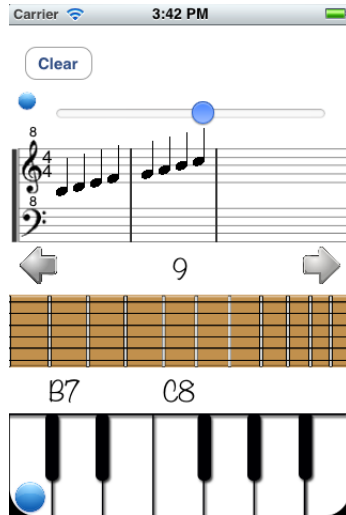
4.2.1. Usability and Functionality

Usability and functionality of the application which was run on an emulator and two different mobile devices were tested and monitored by noting down outputted results by the application which were results from whistling and playing both single notes and then well known songs on instruments.

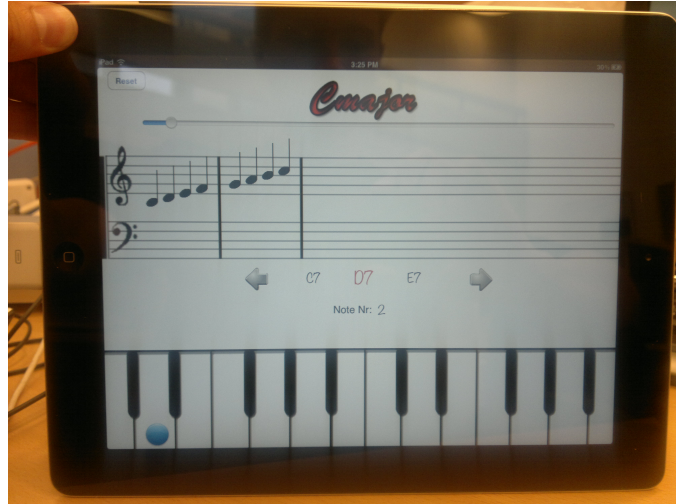
The application had mostly been tested during the design phase for single notes which didn't really add up to any greater meaning. Silences could be detected as the amplitude faded away or was stopped and both short and long notes were output at the right times. Whistling detection had been tested and performed as intended by giving higher and higher notes as the user whistled higher frequencies as well as piano and guitar tones which could be analyzed and shown on a note staff as their corresponding notes. A short test involving a full high in pitch whole-note octave played on a piano was monitored on both an emulated iPhone device and an iPad device and are the results shown on Figure 4.1.

Figure 4.1 shows 8 notes C, D, E, F, G, A, B and another C which is one octave higher than the former C. These first tests including the results from Figure 4.1 gave promising results and it was decided that from here the tests could be scaled up and were moved on to simple songs.

Going from short single notes to songs introduced many new possible faults, especially if the songs were played by a not so experienced musician. The song's tempo, and duration of each note could vary each time a song was played. Users are not always consistent and duration of notes which make up a song almost always change in some way from one play-through to another. Let's also not forget different instruments and that their fine-tuning might also possibly give different results. Many different paths to test the application were open and at least one had to be explored. The path taken was to play a song with a piano keyboard hooked up with a line into a PC. The tones which summed up a song were recorded and could be monitored



(a) Application running on a emulated iPhone device



(b) Application running on iPad device

Figure 4.1: Whole-note octave test

with the computer program Reason which is a music recording and production studio program and comes with instruments and mixing tools[1]. The same song could be played over and over to see if the application would be consistent, and Reason made it easy to move tones up or down an octave for reference. Reason also made it possible to change the output so it would behave as a different instrument.

The first song test started with the very basic and well known song *Mary had a Little Lamb*. Its full song sheet music is shown in Figure 4.2(a).

The first test cases which included playing the song and gathering information was done using an emulated iPhone device on a PC. The input to the device was as mentioned before a played sound file which had been recorded with the program Reason. The sound file could be visualized in Reason and is shown in Figure 4.2(b)

The orange boxes in Figure 4.2(b) represent what key (y axis) on the keyboard was struck at each time and also show the duration (x axis). The song was played according to and should be in somewhat good relation to the music sheet shown on

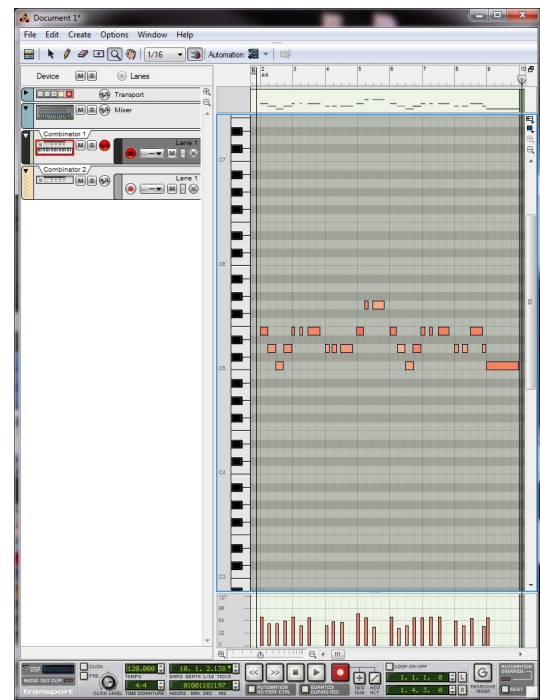
4. Results and Analysis

Mary Had a Little Lamb



Edition Copyright © 2010 Music-for-Music-Teachers
For Use by Music Teachers

(a) Music sheet [7]



(b) Recorded with Reason [1]

Figure 4.2: Mary had a Little Lamb

Figure 4.2(a). A better look at the useful information from Figure 4.2(b) is shown in Figure 4.3.

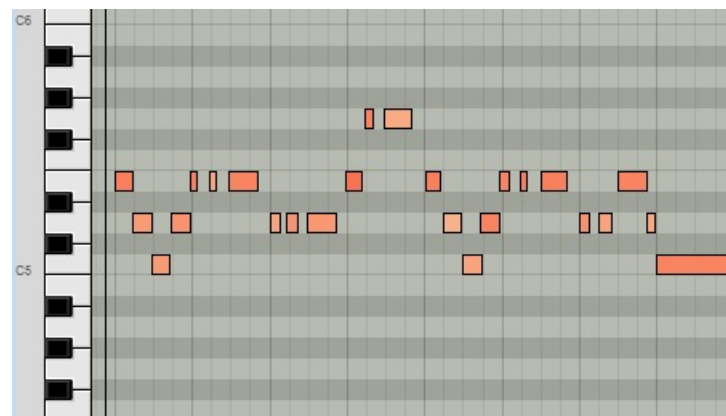


Figure 4.3: Closer look at Figure 4.2(b)

The results from four independent test cases which all used the same setup, settings

and same sound file as input on a emulated device are shown in Figure 4.4

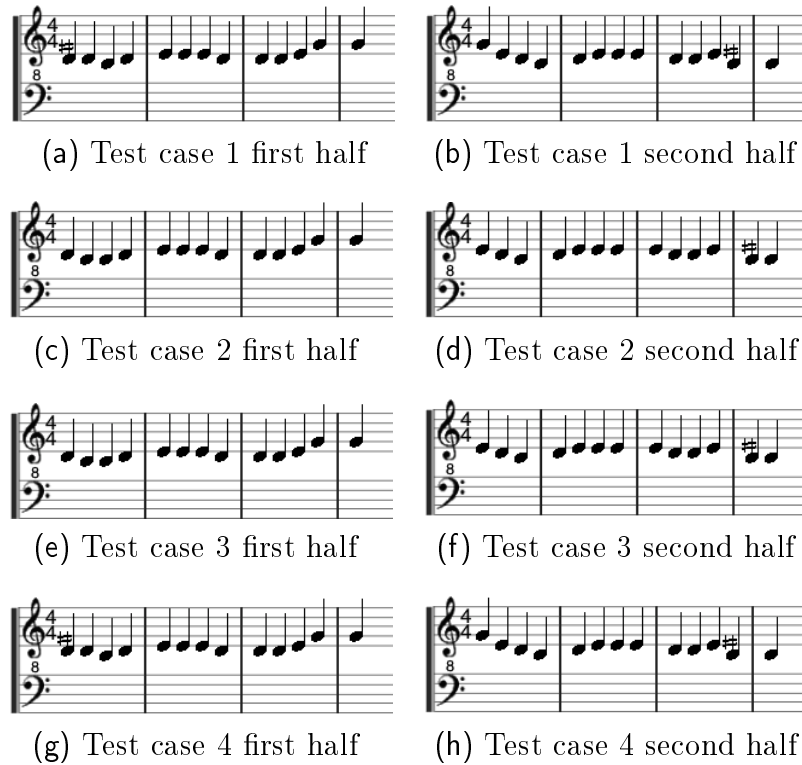


Figure 4.4: Emulator output of *Mary had a Little Lamb* input

Test case 2 and 3 gave exactly the same output which can also be said about test case 1 and 4. Lets bear in mind that the first note of the second part of test case 1 and 4 is the same note as their ending note of the first half since those test cases were one note shorter than the others. The difference between the two different test cases is then only three notes, two very closely placed to the different outcome and one extra added note. The close notes are the first two notes of the first part and the extra note is in the second part in test case 2 and 3 (note 5-8, one extra E). For ease of comparison the results from figure 4.4 were overlaid over each other and are shown in Figure 4.5.

The consistency test had been completed and was considered a good success. Receiving only two outcomes from four different tests which were only three notes or $3/26 = 12\%$ apart from each other. Next up was to see how correctly placed the

4. Results and Analysis



Figure 4.5: Overlaid output from four test cases

notes were, how many notes were missed, how many notes were a direct hit and so on. To further analyze the correctness of the output from the input, the Reason visualization of the sound file used in the test cases was cut up and matched with Figure 4.5 and the results shown in Figure 4.6.

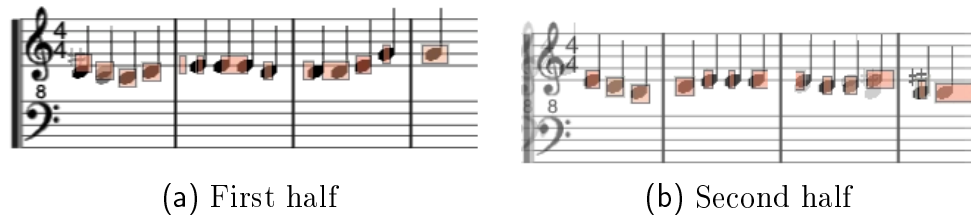


Figure 4.6: Overlaid output from four test cases and Reason visualization

Now everyone can judge for themselves but the application developer was mildly happy with these test results. The application was never intended to fully succeed 100% in analyzing and tracking the notes of a song but what matters the most was that it could be very close and could be further improved and optimized for better results. If the first two notes of the first half of Figure 4.6 are removed it leaves all the rest of the songs notes at their right height on the note staff when compared to the sound file visualization except for one extra sharpened note which should be half an octave higher. Each of these notes was placed in their right place and even better, all orange boxes have a note which represents them if they are allowed to be moved around a little since no accurate time scale between the compared figures was available.

The duration of notes might not be as consistent to the input file and the music sheet as some might have wanted. Each test case could have improved a little with tone duration sampling, for example added a longer duration of a note in the end,

but like everything else it has a good reason for being so. While each test case could have improved a little it can not solely be blamed on the application. Other variables such as the user playing the song having not held the right duration of a note or possibly the amplitude of a tone not being powerful enough to stay over the set threshold of an implemented filter. Some factors and changes here and there could drastically change the outcome. For clarity the sound file was played again where every tone was amplified to a much higher amplitude and the results shown in Figure 4.7.

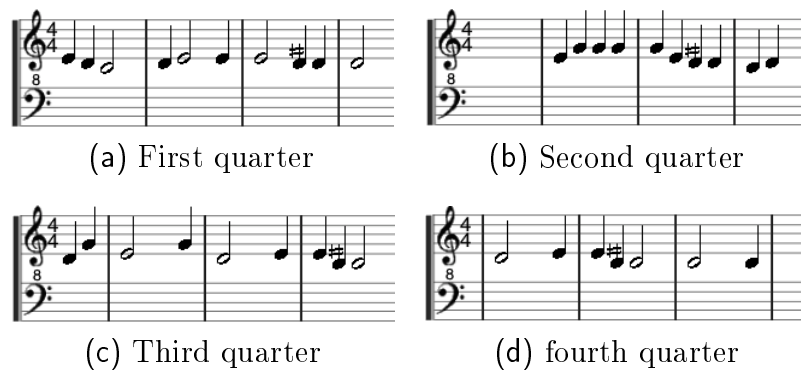


Figure 4.7: Emulator output of *Mary had a Little Lamb* input with increased amplitude

It can easily be seen that there exists a large relation between test cases shown in Figure 4.4 and the single test case shown in Figure 4.7. The main difference being that in the latter mentioned test, notes lasted much longer since their amplitude of dying notes were being pushed over the set amplitude filter when in the earlier tests they had already been filtered out. This resulted in the latter test having around 2x more notes and multiple half notes instead of only quarter notes. Was this a better output? While being a more detailed output it had way to many notes and could easily cause confusion among users.

Another single test with *Mary had a Little Lamb* was performed with the application system running on an iPad device to see if it would result in similar output as on the emulator, hoping that equipment such as speakers and mic and their software would not affect the outcome too much. The results from the test are shown in Figure 4.8

4. Results and Analysis

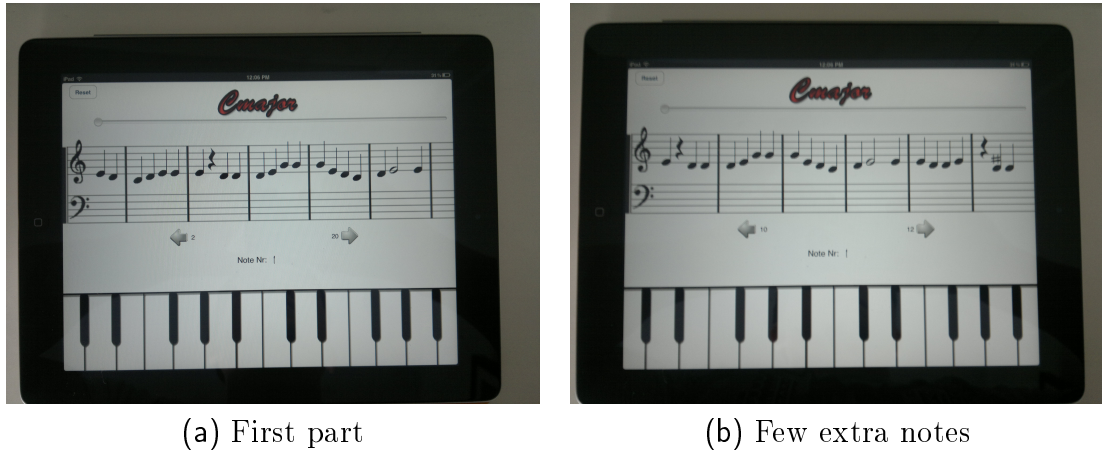


Figure 4.8: iPad Application output of *Mary had a Little Lamb* input

The results were close to perfect, almost each note was a direct hit. A few monitored differences between the former tests and this one: Two strange extra silences were added as well as one extra note and one note got lengthened. This test case could even be argued to have a better output than the former test cases on the emulator due to how well it follows the Reason visualization of the sound file and the sheet music itself.

The tests were finalized with one extra song. A similar test case was performed again with the song *The Legend of Zelda* theme. Results compared to the Reason visualization of the sound file input are shown in Figure 4.9

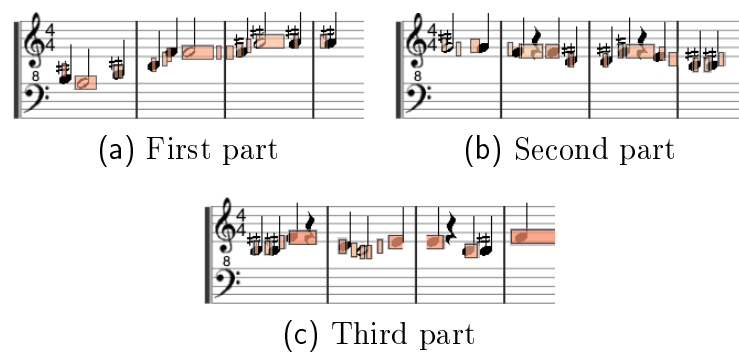


Figure 4.9: Emulator output of *The Legend of Zelda* theme input with Reason visualization

The test shown in Figure 4.9 spanned more notes, and a much larger note range which also had a lot of sharp notes. The application did a little worse in analyzing this test than the tests before but can still be considered to have done well. Only a few notes were half a note away from where they should have been and only the first note was further away than that. A few very short duration tones went undetected (small orange boxes) which might have been played too fast for the analysis but it kept a good track overall.

Running these tests and working on their outcome gave useful information which can be used to further improve the system's analysis. Apparently the first note of each test was not very accurate. Something was wrong with the first note analysis which shall be addressed in improvements to the application.

4.2.2. Computation, Accuracy and Resolution

Accuracy and resolution changed as the sample rate and sample buffer size was changed. Their relation can simply be described by the formula

$$Resolution = \frac{SampleRate}{BufferSize}, \quad (4.1)$$

The resolution had been set in subsection 3.4.5 which was decided from achieving the system tempo of 80-160 BPM. In development these constants were set as Sample Rate = 22050 and Buffer Size = 8192. These constants lead to the system having 80 BPM which was not too fast and not too slow. By putting these constants in (4.1) resolution of 2.692 was achieved. This meant that since more than the resolution constant was between notes that were being analyzed the analysis should be mostly right. A larger resolution percentage of the gap between notes meant worse accuracy performance, since even if the gap between notes was large enough the frequency could still be analyzed as just around the edges of a note and the resolution not be good enough to tell the right note.

4. Results and Analysis

C4 is the lowest note that can be analyzed as the system is set up and the frequency gap between C4 and the next half note of C#4 is 15.55. After that the gap increases with each note and with a resolution of only 2.692 the resolution is well within acceptable limits.

In subsection 4.2.1 the system analysis was however sped up by a factor of four, which had to be done by decreasing the sample buffer size and/or increasing the sample rate. This could simply be achieved by decreasing the buffer size by factor of four and increasing the resolution by the same amount or by decreasing the buffer size by factor of two and increasing the sample rate by same amount. No matter which route was taken the resulting resolution value was increased by the factor of four to 10.77.

The changes meant less resolution and more computation but gave valuable information that had to be used to increase the system's usability. At higher notes the resolution is well within limits but at the lowest frequencies better resolution is desirable. Since both can not be achieved here the option of having more information to work with was taken.

4.3. Comparison

The resulting application system which has been tested and monitored in earlier sections can be said to have a relation to and be compared to many other systems which include some kind of frequency analysis and/or tone training on mobile appliances. It was much harder though to find another product that offered all of the above together in one package to compare with. That is, offering tone training based on the frequency analysis for mobile appliances.

There exist many kinds of frequency analysis and tone training products for all kinds of platforms. Let's not forget that this is a huge market and today many people's

work depends on income from application sales of products such as this one. A few will be mentioned here which all run on the iOS platform and were considered popular and to be in a similar category and related to the system designed in this thesis.

- **SpectrumView** helps users visualize frequencies that the device can pick up in its environment. It has a really high frequency range and enables its user to set their own sample rate and frequency resolution. It is a freeware application developed by Oxford Wave Research Ltd. for the iOS platform [15]. Screenshots from the application are shown in Figure 4.10.

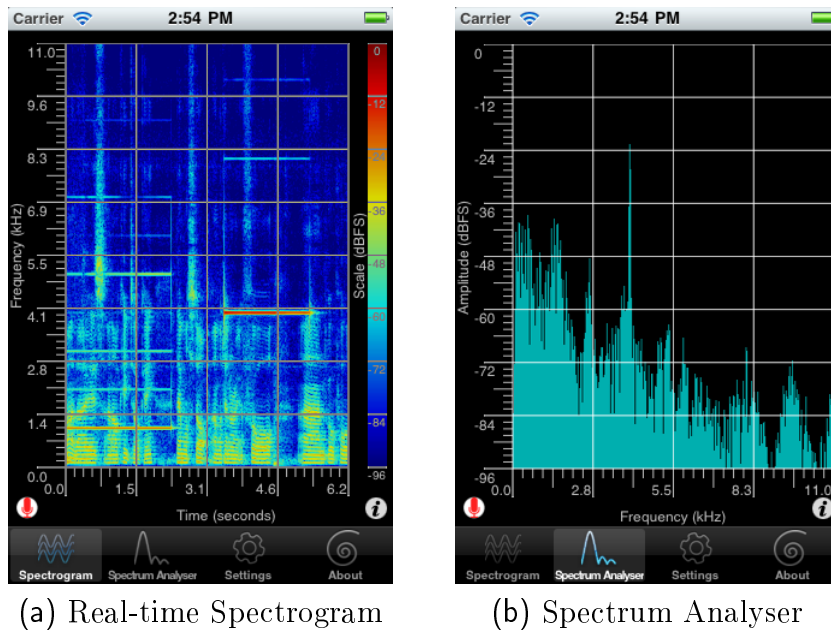


Figure 4.10: *SpectrumView* [15]

- **FFT** is a high-resolution audio analysis tool which uses the Fast Fourier Transform to analyze incoming audio, and displays a very detailed graph of amplitude vs. frequency. It is only one of many commercial application developed by Studio Six Digital and can be purchased in the iTunes store for \$24.99 for the iOS platform [6]. Screenshots from the application are shown in Figure 4.11.

SpectrumView and FFT do the same analysis as this thesis system but instead of focusing on turning the analyzed output into notes they focus on giving accurate

4. Results and Analysis

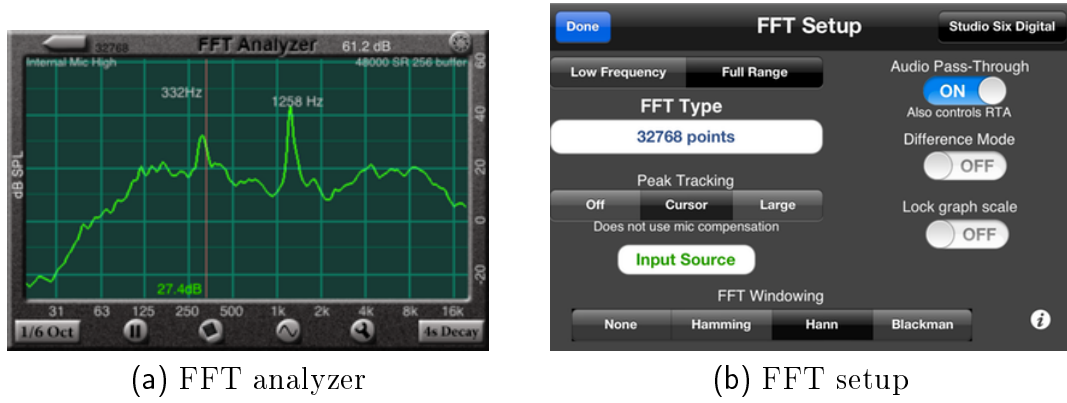


Figure 4.11: FFT analyzer and setup screen [6]

spectrum and spectrograms as well as allowing many kinds of tuning of the instrument.

- **Pro Tuner** is a tuner application especially designed for guitar and bass players. It can be used to give reference sounds when tuning the instrument as well as having input-level indicator support for line in and mic. It is a commercial software developed by Xiao Yixiang which can be purchased for 2.99\$ at the iTunes store for iOS platforms [22]. Screenshots from the application are shown in Figure 4.12.

There exist many tuners for all kinds of instruments today. The most basic ones offering only references for tuning notes and more advanced ones offering impressive accurate analysis by using line-in or mic as samplers, offering different tuning modes and support for more than one instrument. The thesis system is not that different from a tuner which tells which note is being played and if a string should be tuned lower or higher. It might not give the same accuracy as a tuner tool which can tell accurate tuning inside note boundaries and also not offer tuning of tones lower than 250 Hz since they would not even show up but that's mostly since the focus of the these projects are very different. Tuners usually take much larger samples for high resolution outputs while not focusing on fast response as well as not saving older notes in history. The interface is also in no way useful if a user wants to track a whole song.



Figure 4.12: Pro tuner on iOS [22]

- **SoundHound** is a music recognition application. It only needs singing or humming as an input to perform music recognition which should return more information along with lyrics for the song. SoundHound by SoundHound, Inc. can be downloaded free or bought with extra features and no commercials for \$6.99 at the iTunes store for the iOS platform [18]. Screenshots from the application are shown in Figure 4.13.

SoundHound is probably by far the largest application, most downloaded and used amongst the applications mentioned. It's one of a kind whereas no other application has its enormous database which it uses to compare songs with its analysis. Its analysis is not shown in detail and its only output to the user is a song that matches the input in some way. The output focus here is very different compared to this thesis project but if SoundHound can find the right song it should also be able to present the user with a note sheet that could be used to play the song. These note sheets could easily be part of their database and offer sheets for different instruments. SoundHound could therefore easily be a competitor in a similar market for users that are not creating their own songs but trying to find music sheets for songs that already exist by whistling or playing a few notes.

All the above applications perform sound analysis and even more they all use some

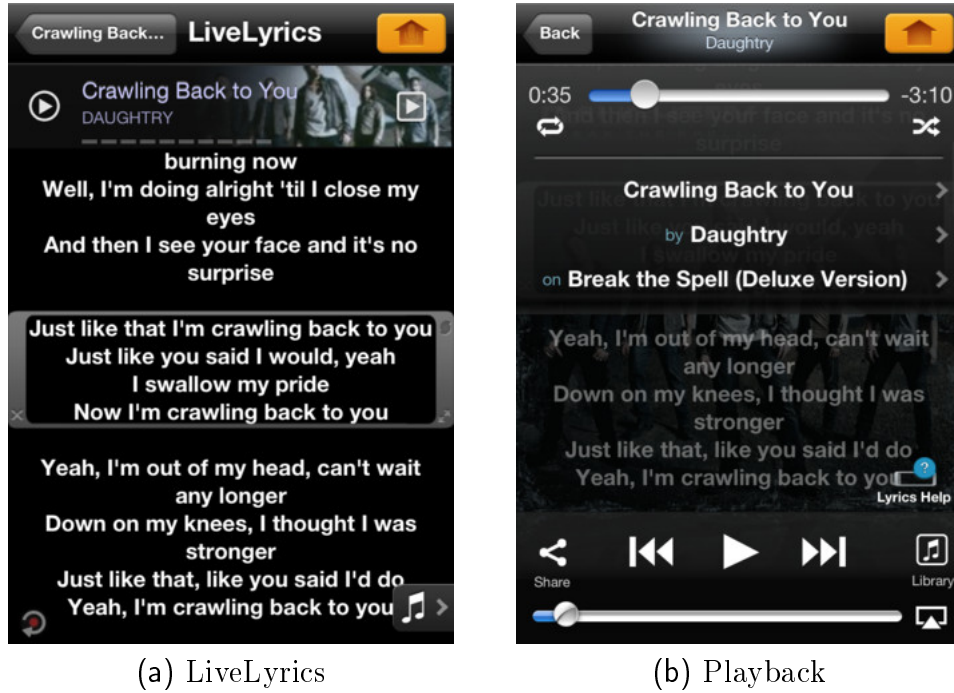


Figure 4.13: SoundHound on iOS [18]

Fast Fourier Transform algorithm to do it. These parts of the programs are similar to this thesis project but what differs is what is done with the analysis. The FFT returns valuable information that can be used in many ways. What this thesis project offers which none of the above have implemented so far is gathering information from the FFT and representing them as helpful notes on a staff. The notes are kept in history and can be inspected for more detailed information. The accuracy may not be as detailed or the user as free to change settings as in some of the above applications mentioned but instead its settings and setup are meant to be optimized for its purpose. Further iterations of the application could of course improve many of its lacking features to compete with the above mentioned applications as well as other applications on the market. Some improvements which have already been thought of are mentioned in the next chapter.

5. Conclusions and Future Work

A fully functional musical tone recognition system for interval tone training has been designed, developed and tested. It was developed with emphasis on higher frequency (250 Hz+) inputs for useful and accurate analysis of fast and rapid changes in frequency. The inputs to the system are only limited by the mentioned frequency threshold and an amplitude filter, apart from that any tone input ranging from whistles to different instruments is sufficient to produce a valid output. The system performed well in tests done by analyzing inputs played by a sampled piano keyboard sound file. Most tones that were monitored were analyzed and registered as their corresponding note on a musical staff with only a few exceptions. The outputs from each test were then compared to the original sound file which could be visualized with a more sophisticated recording studio tool to evaluate the performance of the system.

The two interfaces performed similarly except that the iPad interface offered a much larger staff for notes which proved to be a very useful advantage. Both interfaces worked well and were consistent in their outputs. Both offered the same functionality such as panning the staff for more notes as well as showing two instruments and some small helpful information and notation of how to play each note. The interfaces then also offer useful information that can be used to assist users in learning the basics of musical notation. It can further assist users learning different pitches of tones and see how they correspond in a note on a musical staff.

5.1. Future Work

Below is a list of ideas for future work

- Some users might want to dive into the technical details and settings of the program. Those who have learned Fast Fourier Transform might want to set their own sample buffer size and even change the sampling speed of the application, resulting in changing the application tempo. Other less knowledgeable about FFT might want to change the tempo which could automatically change the sampling speed and/or sample buffer size to result in the desired tempo change for its user.
- Playing the recorded notes in history could add tremendous usability to the system. Users who had the chance of trying out the application all mentioned that they would like to be able to make the application play back all the notes it had recorded. The application could play the sound file it recorded as well as the notes it had analyzed. This could prove convenient for many users and help displaying how accurate the analysis is without having to go through much extra effort.
- Being able to save and load a song from memory as well as to store it on a public server to exchange with other users.
- A line drawn in real time could be turned on and off showing the FFT analysis in some useful way. It could help with debugging as well as give the user more detailed and useful output of data.
- Different settings for different purposes. A few predesigned settings could be offered to users. These settings would have optimized parameters that would perform better in case of certain purposes such as usage of analysis with different instruments. Different instruments, whistling and further inputs may differ a lot in frequency range and some might need more accuracy and more calculations regarding outputs than others.
- The first note analysis in many test cases were worse than their following

note analysis. How the first note is analyzed, processed and mapped should be rethought and redone. It is programmed to not wait for a full array of sampled values but instead use the first sampled value and analyze it for quick response. This has proven unsuccessful and could be improved by waiting for more samples and respond a little later.

- Use silences and pauses to give accurate musical timing. If an analyzed sound didn't meet the requirement of a filter it was not kept as information but instead trashed. It was mainly at a certain time in development that it seemed not useful to monitor pauses. It was thought as to only make the system save a lot of information in memory regarding empty pauses for a song which were unintentional pauses. In early testing this made notes often be far from each other and whenever the user was viewing the system outputs, more empty notes were being added at the back of the list. By implementing user controlled settings which would set how long a maximum pause of silence should be we could add valuable information to the outputs. This could make the system have an accurate timing diagram of the notes, so users could play songs at their right tempo while playing after the systems presented musical notation.

Bibliography

- [1] Propellerheads Software AB. Reason — everything your music needs today. and tomorrow. <http://www.propellerheads.se/products/reason/> Last accessed: 22/08/2012.
- [2] APRONUS.COM. Melodies, beats, bars, timeline, notes. <http://www.apronus.com/music/lessons/unit02.htm> Last accessed: 07/02/2012.
- [3] Wayne Chase. *How Music Really Works!* Roedy Black Publishing Inc, second edition, 2006. ISBN = 1897311559.
- [4] The Sparrow Framework Community. Universal app development. http://wiki.sparrow-framework.org/manual/universal_app_development Last accessed: 21/08/2012.
- [5] National Instruments Corporation. Labview system design software, 2012. <http://www.ni.com/labview> Last accessed: 30/08/2012.
- [6] Studio Six Digital. Fft. <http://www.studiosixdigital.com/audiotools/fft/> Last accessed: 15/08/2012.
- [7] Music for Music Teachers. Mary had a little lamb. <http://www.music-for-music-teachers.com/support-files/mary-had-a-little-lamb-no-keyboard-fixed.pdf> Last accessed: 15/08/2012.
- [8] Jean Baptiste Joseph Fourier and Alexander Freeman. *The Analytical Theory of Heat*. Cambridge, 1878. ISBN = 9781108001786.

BIBLIOGRAPHY

- [9] Jack D. Gaskill. *Linear Systems, Fourier Transforms, and Optics*. John Wiley and Sons, Inc, first edition, 1978. ISBN = 0-471-29288-5.
- [10] Guitar It. The online resource for guitarists. <http://www.guitarit.com/beginner-guitar-chords/> Last accessed: 13/07/2012.
- [11] Eric Larson and Ross Maddox. Real-time time-domain pitch tracking using wavelets. Technical report.
- [12] Music Maker Guitar Lessons. (Guitar fretboard). <http://www.guitar-lessons-for-beginners.co.uk> Last accessed: 20/03/2012.
- [13] Demetri Miller. Pitch detection in ios 4.x. <http://demetrimiller.com/2011/01/25/pitch-detection-in-ios-4-x/> Last accessed: 09/07/2012.
- [14] Mikael Nilsson, Josef Strom Bartunek, Jorgen Nordberg, and Ingvar Claesson. Human whistle detection and frequency estimation. Technical report.
- [15] OxforWaveResearch. Spectrumview. <http://oxfordwaveresearch.com/joomla/spectrumview> Last accessed: 15/08/2012.
- [16] Piano-Lessons-Info.com. <http://www.piano-lessons-info.com> Last accessed: 07/09/2012.
- [17] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing Principles, Algorithms, and Applications*. Prentice-Hall, Inc, third edition, 1996. ISBN = 0-13-373762-4.
- [18] SoundHound. Soundhound instant music search and discovery. <http://www.soundhound.com/> Last accessed: 15/08/2012.
- [19] Michigan Tech. Physics of music - notes. <http://www.phy.mtu.edu/~suits/notefreqs.html> Last accessed: 26/08/2012.
- [20] Inc. The MathWorks. Matlab, the language of technical computing, 2012. <http://www.mathworks.se/products/matlab/> Last accessed: 30/08/2012.

- [21] Hartmut Traunmüller and Anders Eriksson. The frequency range of the voice fundamental in the speech of male and female adults. Technical report, Department of Linguistics, University of Stockholm.
- [22] Pro Tuner. Pro tuner add a new pedal to your iphone. <http://eumlab.com/pro-tuner/> Last accessed: 15/08/2012.
- [23] Australia UNSW School of Physics Sydney. db: What is a decibel? <http://www.animations.physics.unsw.edu.au/jw/dB.htm> Last accessed: 22/08/2012.
- [24] Own work by uploader Wikipedia. Harmonic series (music). http://en.wikipedia.org/wiki/File:Harmonicpartials_on_strings.svg Last accessed: 15/01/2012.
- [25] Chris Yoko. What is mobility. <http://www.yokoco.com/2012/02/what-is-mobility/> Last accessed: 27/06/2012.
- [26] Guitar Lessons 4 You. Guitar scales). <http://www.guitarlessons4you.com/guitar-scales/> Last accessed: 20/03/2012.
- [27] Chik Z, Islam T, Rosyidi S.A, Sanusi H, Taha M.R, and M.M Mustafa. *European Journal of Scientific Research*, 2009. ISSN = 1450-216X, Comparing the Performance of Fourier Decomposition and Wavelet Decomposition for Seismic Signal Analysis.

A. Appendix

A. Appendix

№	Name	Imp.	Tasks	Comments
1	Listen to input	1	-Setup audio listener/buffer and audio streams -Set sample rate -Start the listener	Listener should always be active and starts up with the program When desired amount of input has been sampled: perform FFT
2	Transform input with FFT	1	Fill buffer with sampled data Run FFT using vDSP function on buffered data	
3	Analyze FFT transformed signal	1	-Find the dominant frequency in buffer	"Magnitude squared"
4	Store analyzed inputs	1	-Create a array for analyzed data and fill it up as the process proceeds -Create "frequency to notes" map of all desired frequencies/notes -Make a search that finds the closest note to frequency in map -Disregard small noise -Treat silence differently	Possibly store both frequency and note
5	Map dominant frequency to note or silence	1	-Come up with ideas for the interface -Discuss their pros and cons -Conclusion	
6	Design main interface	1	-Compare the note with earlier note -Create a flag that counts how many times the note was repeated -Create a array that stores multiplications of notes (how many and what note) -Place image holders for each note that can be displayed on screen -Create graphical note and a sharp note -Position note images on the GUI in relation to their mapped note(move them up 1 and down)	Interface involving: Staff, notes, tablature and instruments. If more than 2 of the same note are together then they should be drawn as a single note with longer duration
7	Check for duplicate notes	2		
8	Draw notes on GUI	1		Different notes have different position on the staff Whole/Half/Quarter notes have different picture depending on their length(how many times they were repeated in a row) Duplicate notes are used to find out length of notes After one Whole, 2xHalf or 4xQuarter notes there should be a bar(vertical line) that tells where a measure starts/ends (When Beat 4/4 is used)
9	Draw note length	2	-Create graphics for 3 different length of notes -Change note image to represent the length of the note	User should be able to choose which note he wants to inspect from notes on the GUI staff
10	Draw vertical lines between measures	2	-Add a bar after each 4 notes	Could display both frequency and/or Note
11	Choose a note to inspect	1	-Create a GUI note picker that user can use to pick a note to inspect from the list of sampled notes	
12	Display value of inspected note	1	-Create GUI display which displays more detailed information about the inspected note	
13	Draw helper points on piano/guitar	2	-Draw a dot that displays how to play inspected note on a piano -Create guitar tabs for one note stair and display the for inspected notes (iPhone) -Draw dots that display how to play inspected note on a guitar(iPad)	

Figure A.1: Product Backlog

14	Pan the staff for more notes	2	-Create gesture recognizer -Adjust movements -Update the notes drawn in the GUI and also the note picker	Create gesture recognizer so the list can easily be panned with finger movements Adjust movements so they scroll the list in a user friendly manner
15	Clear values (Reset)	2	-Create a clear/reset GUI button -Clear values/lists/arrays and buffers and hide image holders	
16	Upgrade iOS	2	-Upgrade the project to newest iOS -Create a new project -Import all files and libraries -Resolve ARC issues	Upgrade the project from iOS 4.0 to 5.1
17	Switch between piano/guitar (if needed on either device)	3		
18	Play notes on GUI piano	3		Play piano keyboard notes with sound files
19	Play notes on GUI guitar	3		Play guitar notes with sound files
20	Play all sampled notes	3		
21	Draw notes with quartz instead of images	3		