Frequency and Spectrum Analysis
to Compare Sounds
Elodie Thomann
Computer Science
Session : 2013/2014

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)_________________________
Abstract

Computer science nowadays is used to support people in a large number of diverse domains, including the domain of music. New technologies are often utilized as a guide to visualize music.

This project will study previous research undertaken in the domain and assess some currently available tools. Then it will go through the development of software that gives musicians another way of understanding musical sounds.
Acknowledgements

I would like to acknowledge and thank the following people whose support has been invaluable throughout this project:

- My Project Supervisor, Dr Kia Ng, for his constant and effective support throughout the project, and in particular for his advice to me during our weekly progress meetings, especially when I came across project turning points.

- My Project Assessor, Dr Nick Efford, for his valuable feedback on the Mid Project Report and during the Progress Meeting.

- All tester participants for having spent time in evaluating and giving me feedback on my software.

- My parents for their constant interest and their encouraging support throughout my project.
**Contents**

1 Introduction

1.1 Context and definition of the problem ........................................... 1
1.2 Project aim .................................................................................. 1
1.3 Motivation .................................................................................... 1
1.4 Relevance to the degree programme ............................................... 2
1.5 Objectives .................................................................................... 2
1.6 Minimum requirements .................................................................. 2
1.7 Possible extensions ....................................................................... 2
1.8 Delivery ....................................................................................... 3
1.9 Methodology ............................................................................... 3
1.10 Project schedule ......................................................................... 3

2 Background Research

2.1 Introduction .................................................................................. 6
2.2 Analysis of a musical note and timbre .............................................. 6
2.3 Signal processing methods ............................................................. 7
   2.3.1 Fourier series and Fast Fourier Transform (FFT) ....................... 7
   2.3.2 Yule-Walker method ............................................................... 8
   2.3.3 Short Time Fourier Transform (STFT) and windowing ............. 10
      2.3.3.1 Short Time Fourier Transform ........................................ 10
      2.3.3.2 Description and comparison of window types ................... 11
   2.3.4 Wavelets ............................................................................... 13
   2.3.5 Signal processing using the envelope of the signal .................... 14
   2.3.6 Conclusion on signal processing techniques ............................ 15
2.4 Interface design .......................................................................... 15

3 Design and Development

3.1 Introduction .................................................................................. 18
3.2 Design .......................................................................................... 18
   3.2.1 Context ................................................................................. 18
   3.2.2 Methodology ......................................................................... 18
   3.2.3 User requirements ............................................................... 18
   3.2.4 Specifications ....................................................................... 19
   3.2.5 Three-tier architecture: Presentation Application Database diagram 20
Chapter 1

Introduction

1.1 Context and definition of the problem

When people hear sounds, this is an ephemeral phenomenon. Musicians are appreciating their instrument sound quality only while they are playing. Sometimes, the note played does not sound as good as expected and a visualisation tool would be most useful to help they understand the different elements that together make the sound.

At the moment, there are not many tools available to musicians for analysing instrumental sound. The basic one is the tuner, to detect pitch. However, tuners are limited in function. Several other tools exist to analyse sound at a deeper level but they are usually too complex, hence are only used by specialists of sound.

A solution to face this lack of tools is to create some software which fulfils this need by combining a tuner and other features of sound analyser.

1.2 Project aim

The aim of this project is to design and develop a software programme which evaluates musical instrument sounds in terms of pitch and timbre. The software is intended to display the results in a way that will convey useful to the musician by using visualisation techniques.

1.3 Motivation

Given the project’s aim, the musician is in the best position to define the needs, and the computer scientist is in the best position to evaluate what is possible and to create the tool.

In the case where the musician and the computer scientist are one and the same person, working on the product becomes even more interesting and focused.

This is the situation here and provided the motivation to bring this project to completion.
1.4 Relevance to the degree programme

Skills from several Computer Science degree modules have been drawn upon in order to achieve this project. The main ones are:

- Computer Vision;
- Software Systems Engineering;
- HCI.

1.5 Objectives

The objectives are:

1. Perform a background research and a survey on existing methods for signals analysis with focus on musical sounds;
2. Design and develop a software prototype integrating audio input, analysis and a way of visualising results;
3. Develop a signal processing module for pitch analysis to be integrated into the software prototype;
4. Develop a signal processing module for spectral analysis to be integrated into the software prototype;
5. Design and develop a similarity measure module to be integrated into the software prototype.

1.6 Minimum requirements

The minimum requirements are:

- A prototype composed of a set of audio processing algorithms to detect musical pitches;
- A prototype to detect and follow the variations of a sound over time;
- A graphical interface to nicely display feedback to the user.

1.7 Possible extensions

The possible extensions are:

- A prototype to visualize the difference between sounds produced by two instruments of the same type using spectrum analysis;
- An interface to support real-time processing of the prototypes.
1.8 Delivery

The project as delivered will comprise of a report and integrated software which consists of a signal processing module for pitch analysis and visualisation.

1.9 Methodology

A suitable approach to tackling this project is to use an Iterative Design methodology [23]. Applied to this project the method is as follows:

1. plan how the project will be scheduled to keep track of time at all stages;
2. perform background research on the subject
3. determine the product requirements;
4. implement each minimum requirement before moving on to its possible extension;
5. evaluate the product to review and revise it.

Step 5 is repeated until the product is considered as completed.

Due to time restriction, only 2 iterations of step 5 have been undertaken.

After these 2 iterations, the project content is fixed and the project products are finalized.

This method has the advantage of setting clear checkpoints and deadlines during the development stages. It also ensures that time is set by for revisiting areas of the system to make improvements.

1.10 Project schedule

The initial schedule of the project is presented on Figure 1.1.
At week 12 (from 14/04 to 21/04), due to positive feedbacks on the evaluation of both functionality and user questionnaire, the two weeks scheduled for implementing the second prototype were overestimated. In fact, the implementation was achieved within a week. The second week was used to review the write-up of the project report. The planning had to be revised. The second version is presented in Figure 1.2.

This section has addressed Step 1 of the methodology.
Figure 1.2: Reviewed Grant Chart
Chapter 2

Background Research

2.1 Introduction

Given the aim of the project, the background research focused on signal processing. There are only two ways of analysing a sound: pitch tracking and envelope analysis. Hence, this chapter includes first a summary of the properties of musical sounds (Section 2.2), second, understanding and analysis of pitch tracking methods as well as an investigation on sound using the envelope analysis approach and a conclusion on signal processing techniques (Section 2.3), finally an evaluation of the current tuner interfaces (Section 2.4).

To date there have been few attempts to redesign a tuner or to integrate additional tools. Tuners and the related tools dedicated to users have evolved to a standard design and little research has been devoted to developing a new layout. On the other hand, a great deal of interest has been shown in researching alternative methods for analysing sound, particularly musical sounds. Some examples of research have been carried out by the International Society for Music Information Retrieval (ISMIR), the New Interfaces for Musical Expression (NIME) and many others.

Given the scope of this project, these reports have been read more for interest than for their usefulness.

2.2 Analysis of a musical note and timbre

A sound is an approximately periodic signal or wave characterised by its frequencies and related amplitude [31][12]. According to the Fourier representation, a signal is composed of a fundamental frequency and harmonics. The fundamental frequency has the highest amplitude, hence it is the one we hear the most. It also gives the pitch of the sound. The harmonics are frequencies with lower amplitudes. They make the sound richer and thereby determine the timbre [12].

Harmonics are distributed along the signal following a pattern. Knowing the fundamental frequency
Apart from the presence of harmonics, timbre is also characterized by other parameters. The main ones are the logarithm of the attack time (see Section 2.3.5), the spectrum gravity centre, the spectrum variation (also called spectral flow) [8] and the internal rate of return (IRR).

The logarithm of the attack time is very specific to each type of instrument, but minute differences become apparent when two instruments of the same type are studied. This parameter may not be suitable, considering the scope of that project.

The gravity centre is highly correlated to the brightness of the sound. ‘Brightness‘ is usually used to describe sound quality and timbre, using a rough analogy with visual brightness. Timbre researchers consider brightness to be one of the perceptually strongest distinctions between sounds, and formalize it acoustically as an indication of the amount of high-frequency content in a sound [15] [35]. Furthermore, in [7], Barthet confirms that this parameter can be used to study the difference in timbre between two instruments of the same kind, a factor which falls within the project scope. The spectral flow cannot account for more than 34% of the spectrum variance [15]. Because of its low influence, the spectral flow is not sufficient as explanation. The discussion in [15] outlines the IRR. This parameter describes the degree of irregularity in a spectrum. There is a high degree of inverse correlation to the spectral flow and this accounts for 73% of the spectrum variance. IRR is interesting to use.

The quality of timbre is complex and has not been entirely mastered. In any case, there are no standards to describe a ‘good’ sound as it really depends on the human ear and personal appreciation. Hence it is impossible to develop a timbre model for all sounds. However, the parameters introduced in [15] and [7] can be put to use in order to compare timbres. Section 3.3.4 presents a way of using these parameters properly.

2.3 Signal processing methods

2.3.1 Fourier series and Fast Fourier Transform (FFT)

Fourier Series is an algorithm which decomposes a periodic signal into frequencies and corresponding amplitudes. The use of complex values slows down the computation. The FFT is an improvement in terms of the speed of the traditional Fourier Series. The speed owes this improvement to a data length of power of 2 which reduces the number of iterations performed on complex values. This algorithm runs in $O(n \times \log(n))$ instead of $O(n^2)$ [12][5][11].

Stromboni and Le Roux present in [16] a demonstration of a Java FFT implementation applied to track a guitar sound pitch. They use these results to visualise the spectrum distribution and find the fundamental note played. Squire in [32] uses FFT to decompose and perfectly recompose a sound. According to these studies the FFT algorithm can be considered as a good tool for analysing general pitch while losing very little information. Therefore, FFT is adequate for meeting the first minimum requirement of the project. The main disadvantage of FFT is the global analysis of a signal. The method has
no time tracker, unlike the Short Time Fourier Transform (STFT) (see next Section). Hence FFT is only efficient while analysing a single pitch signal that is sustained and smooth enough to be processed by the algorithm [12].

The following figures give an appreciation of the FFT working.
As shown in the left graph of Figure 2.1, a pure sound is composed of only one frequency, the fundamental.
As shown in the right graph of Figure 2.1, a complex sound is composed of a fundamental frequency having the highest amplitude together with harmonics having lower amplitudes.

![Image of FFT output using Matlab on a pure and a complex sound](image)

Figure 2.1: Output of FFT method using Matlab on a pure and a complex sound

### 2.3.2 Yule-Walker method

Autoregressive models (AR) are possible alternatives to FFT. They are already used in many domains such as statistics, nature, economics and signal processing. AR is a way to represent data on the assumption that the processed variable depends on its own previous variable. More particularly, it can be used in signal processing to compute the power spectral density estimate of a sound [18].

This method is good for diagnosing the signal and dealing with noise data. The latter is of great interest, given there is always a chance that output data can be subject to noise [17].

On the other hand De Hoon et al. [17] have run tests on and evaluations of the Yule Walker method
for a large range of signals. However, it has been proved that in some cases involving nearly periodic signals, the Yule-Walker approach may lead to incorrect parameter estimates. Given that the processed signal is considered to be periodic or nearly periodic, there is a high chance this method will not provide any accurate results.

The method has been tested using the corresponding Matlab function. The Figure 2.2 shows that the output results match the De Hoon evaluation.

The sound analysed is a 440 Hz pitch played on a violin. In this example the sample rate is 10,000.

**Remarks on the left graph of Figure 2.2 results:**

The default Matlab parameters have been chosen. According to the axis values:

- Maximum power/frequency read on the graph: $0.87 \times \pi$ rad/sample
- Corresponding frequency:
  $$\frac{(x \text{ axis value} \times 10000)}{(2 \times \pi \times \pi)} = \frac{(0.87 \times 10000)}{(2 \times \pi \times \pi)} = 441 \text{ Hz}$$

The fundamental frequency result is correct.

However, no other optimum has been found, hence there is only one frequency found by the algorithm. The harmonics were not found.

**Remarks on the right graph of Figure 2.2 results:**

In order to find the harmonics, some functional parameters have been changed. According to the axis values:

- Maximum power/frequency read on the graph: $0.98 \times \pi$ rad/sample
- Corresponding frequency:
  $$\frac{(x \text{ axis value} \times 10000)}{(2 \times \pi \times \pi)} = \frac{(0.98 \times 10000)}{(2 \times \pi \times \pi)} = 497 \text{ Hz}$$

The fundamental frequency result is not correct.
The second peak power/frequency has a lower frequency compared to the fundamental. This makes no sense for a harmonic frequency.

It seems that the Yule-Walker technique is good for finding the fundamental frequency but not suitable for tracking the harmonics. It is of no interest to the project scope. Hence it is not recommended to use this method in the scope of this project.

2.3.3 Short Time Fourier Transform (STFT) and windowing

2.3.3.1 Short Time Fourier Transform

To overcome the global analysis problem of the FFT method, the idea is to perform multiple FFT on multiple parts of the signal using a window function. The latter is a mathematical function which is described in Section 2.3.3.2. The result is that a windowing of the signal can give a more precise idea of pitch variation over time \[12\] [25].

Note that a problem may rise if the window contains a signal with very different frequencies. Huynh [14] has specified that a STFT analysis of a segment covering the boundary between two notes may detect an average frequency instead of the two individual frequencies. The tests below confirm Huynh’s affirmation.

In order to use this method, the window size has to be fixed. Seo [29] has performed some tests on a complex sound signal for different window sizes. His results and additional tests will be used to decide the window size in Section 3.3.4.

Figure 2.3 shows an example of STFT output using Matlab:

![Figure 2.3: Output of STFT method using Matlab on a pure and a complex sound](10)
Left graph: performance on a single sound, the note G at 98 Hz played on a double bass.

Right graph: performance of two sounds, the notes G at 98 Hz and D at 73 Hz played on a double bass.

Analysis of the left graph:

The colours represent the intensity of the signal. The greater the intensity of the colour red, the higher the amplitude of the corresponding frequency in the processed signal. The greater the intensity of the colour blue, the lower the amplitude in the corresponding frequency. In the graph on the left we see clearly one frequency over time with an approximate frequency of 100 Hz.

Analysis of the right graph:

The representation uses the same colour code. In this case it is hard to say how many different sounds are played and when. This weakness supports Huynh’s limitation argument.

However, the project scope is to develop a tuner and a timbre analyser. In both cases the frequency is expected to be sustained. As shown in the graph on the right, the small frequency variations due to the tuning of the note do not affect the STFT result. Hence the method is of interest.

2.3.3.2 Description and comparison of window types

A window function is a mathematical function. This window is composed of coefficients which aim to add value to some part of the processed signal [34]. When applied to STFT, this window goes across the signal and computes the frequency distribution.

This window may produce leakage depending on its type. Leakage, also called ‘temporal aliasing’, is a phenomenon which happens when FFT is performed on a sampled signal. FFT requires a complete periodic signal in order to find mains frequencies. But in the case of a sampled signal, it is not guaranteed that the processed signal is complete, i.e. there is no guarantee that all periods present in the window are complete. Windowing methods deal with leakage by reducing its effects. They cannot eliminate it entirely but less attention needs to be paid to the leakage while performing FFT.

There are quite a few types of window. They affect the signal in different ways as described in Figure 2.4 [34].

Note that the rectangular window, or Dirichlet window, is missing from the above listing of techniques. It does not affect the processed signal at all. An idea of its shape can be represented as shown in Figure 2.5.
Commentary on choice of window:

Weisstein [34] and Seo [29] both share the view that the Hamming and the Hanning Windows give a more precise frequency estimation than the rectangular window. Actually, these windows accord less importance to the beginning (the attack) and the end of the signal (the release) but focus on the sustain part (see Section 2.3.5). Looking at the table specification in Figure 2.6 the Hanning window may be one of greatest interest.

In the project context we want the window to give equal importance to all the signals. However aliasing may occur at the extremes because the periods may be truncated. According to Figure 2.4 and Figure 2.5 windows having flat shape are the Tukey and the Dirichlet. The comparison in Figure 2.6 shows that the Dirichlet is better that the Tukey for finding frequency.

Hence the Dirichlet seems the most appropriate window shape to choose.
2.3.4 Wavelets

Wavelets are one way of overcoming all difficulties faced in FFT and STFT [12][25]. Discrete Wavelet transform (DWT) is a method implementing Wavelets.

The DWT method matches a synthesised wavelet to the signal using two parameters: one describes the wave shape and the other describes its position along the signal, i.e. its position in time. It is a multi-resolution process. In other words, it is able to change its time and frequency range resolution as it goes along [36][25]. In this way, Wavelet is a very interesting method for analysing signals containing multiple pitches, as during the analysis it can focus on specific moments, i.e. when and how the frequencies vary while going from one note to another.

The DWT evaluation of Joe Yen in [37] on wavelet analysis proves that using the wavelet method delivers excellent results in both time and frequency domains.

Timbre is constituted of multiple frequency phases. Thanks to the DWT method, a deep analysis of the sound timbre becomes possible [25].

On the other hand, the main drawback of this technique is the absence of any standard configuration of the two parameters. Hence every time this technique is used, a search over a large range of data has to be performed in order to find the most appropriate values. During this stage, redundancy may appear.
Thus, this method is very time-consuming and it seems difficult to obtain a solution in an acceptable time when used in real-time [25][37].

This method is still of great interest to the project scope, as it is the only method which analyses the timbre of a musical sound at such depth over time [25][37].

![Wavelet Method Output](image)

**Figure 2.7: Output of Wavelet method using Matlab**

**Analysis:**

Figure 2.7 represents a visualisation of the Wavelets output on an A 440 Hz played on a violin. Grey levels represent the amplitude intensity of a particular frequency. Going up each layer along the frequency axis, the signal is split into smaller segments and a more detailed sound analysis is performed. Looking at the diagram, a user might have some difficulties in grasping the meaning of the displayed data, hence another form of results visualisation is needed. Furthermore, the analysis has been carried out in such detail that it does not seem appropriate to use this technique to extract the frequencies present in a signal.

The Wavelets method is usually utilized to reconstitute sound composed of multiple notes at a very detailed level [36] and this goes far beyond the project scope.

**2.3.5 Signal processing using the envelope of the signal**

The envelope is another way to analyse a sound. Instead of tracking pitches and amplitudes on the global signal, the envelope approach looks at the signals shape over time. Each instrument type has its own envelope shape defined by the gradient of the four phases described
in Figure 2.8. This shape defines the instrumental timbre. But the shape does not vary enough between two instruments of the same type to be tracked [1][12]. The analysis of sound using the envelope of the signal is mainly used by synthesizer to reproduce an instrumental sound [1].

![Envelope shape of a signal](image)

Figure 2.8: Envelope shape of a signal

However, the projects scope is limited to a comparison of sounds between two instruments of the same type. Hence according to the previous paragraph, signal processing using the envelope of sound is not adequate for use in this project.

2.3.6 Conclusion on signal processing techniques

Signal processing is still an open research area on both pitch tracking and timbre analysis. The Fourier series alternative attempts are not convincing and it seems that no recent methods have been able to overtake it. Since 1807 the Fourier series method has remained as the standard means to detect both the amplitudes and frequencies composing the sound and so to identify the uniqueness of timbre of every single instrument. Furthermore, its improved version (FFT) provides a way to make it computationally effective. Even if the Wavelets method seems to be the main FFT rival, it remains an unfitting method regarding the project scope.

The Fourier series, more specifically the STFT method, has been selected for the project software implementation.

2.4 Interface design

Fourier series implementation code found on-line does not provide a user interface. The code is run through the command line and the results are not displayed in a comprehensive way for any user.

Moreover the current tuner does not provide enough information to give a complete feedback on the analysis carried out.

It is therefore crucial to design and develop a new interface.

As a start, the design of the new interface needs an analysis of the advantages and limitations of the most current interface in order to extend that version to a more complete and user-friendly interface.

The generally known current interface is presented in Figure 2.9.
This interface comprises a tuner function only. The main advantage of this interface is that it is highly intuitive and easy use: it is a matter of common sense that at the moment the pointer hits zero and the green light turns on, the note played is now tuned. However, it is not easy to verify that the note is tuned properly. In fact, the experience of musicians shows that the tuning result is valid only if the played note is stable for 2 to 3 seconds. The current system does not show any history, so the situation cannot be verified. A history tracker is needed. Figure 2.10 illustrates the discussed situation.

Some ‘out of the box’ interfaces [9] use a history tracker on the tuner. The graph in Figure 2.11 gives an idea of this functionality. The latter will be present in the future interface. Note that Section 2 has addressed Step 2 of the methodology.
Figure 2.11: Alternative interface
3.1 Introduction

This chapter includes the design and development process undertaken in order to create the prototype.

The Design section discusses all the decisions taken in order to develop a program that meets the project objectives and requirements.

In the Development section we will see how the data gathered in the Design section will be used to create the software.

3.2 Design

3.2.1 Context

Given the minimum requirements and objectives, the software has had to deliver a new interface to support a musician playing an instrument. It results in a product which:

- analyses a pitch played on one instrument,
- compares this pitch with the same pitch played on another instrument of the same type.

3.2.2 Methodology

The user requirement and prototype specifications will be discussed to identify the final product’s functionalities. They will be visualised, thanks to a PAD architecture and a schema of the forthcoming interface.

Note that Section 3.2.3 and 3.2.4 are addressing Step 3 of the undertaken project methodology.

3.2.3 User requirements

In order to analyse pitch, a musician needs first for the system:
• to show the played note and to display how far the pitch is from the correct tuning;

• to show the reference frequency. This is important as it may not be the same one for each musician; some tune their note A to 440Hz, others to 442Hz [19];

• to keep the processed note in a history list. It results in an interesting didactic tool with which the musician can practise and get feedback on his playing in terms of tuning;

• to run in real-time with an adequate reactivity.

The above requirements define a tuner.

In order to analyse pitch, a musician also needs for the system:

• to show the components of sounds. This may be useful for a musician or an opera singer to practise sound or vocal quality;

• to run in real-time with an adequate reactivity.

The above requirements define a timbre analyser.

In order to compare two sounds, a musician needs the system:

• to enable the recording of two sounds;

• to show how close the two sounds are.

The above requirements define a timbre comparator.

In addition, for the musician’s comfort, the system needs:

• to be able to record music. This is useful because while playing the musician cannot focus on the musicality as he concentrates on the performance. It could be useful to enable a replay of the recorded sound. It becomes a teaching tool;

• to store the recording.

The above requirements define a recorder.

3.2.4 Specifications

In order to meet the user requirements described in the previous section, the system must:

• capture sound by using a microphone;

• find the fundamental frequency and the harmonics of the played note;

• compare the frequency value to a reference table composed of all the theoretical note frequencies;

• find the name, the octave number and the tuning error of the played note;

• record all the information in a file.
In addition:
The *tuner* must:

- display in real-time the name, the octave number and the tuning error of the played note;
- display in real-time the evolution of the tuning error using a graph.

The *tuner history* must:

- display the tuner history file sorted by date;
- compute and display a qualitative estimation of the tuning.

The *timbre analyser* must:

- set and display the graph of the harmonics distribution in real-time (seen in Section 2.2).

The *timbre comparator* must:

- check if the two sounds correspond to the same note;
- set and display a comparative graph by showing the harmonics distribution of the two sounds;
- compute and display comparative timbre figures (IRR and gravity centre Section 2.2).

The *recorder* must:

- set and display a graph to track audio playback;
- save the recording in an audio file.

A *Help file* will be provided for the user to understand how to use each functionality and how to interpret the result.

### 3.2.5 Three-tier architecture: Presentation Application Database diagram

Figure 3.1 presents the three-tier architecture of the overall system and address the user requirements and specifications discussed in Sections 3.2.3 and 3.2.4.

A more detailed process description of module communication and working is displayed in Section 3.3.6.

Note that every time a procedure is repeated, the same code is used.
Figure 3.1: PAD architecture of the system
3.2.6 User interface schema

The interface schema in Figure 3.2, 3.3, 3.4 has been drawn using the material discussed in Section 2.6 and also personal ideas. The Background Research Chapter and the Java Sound Demo Applet [22] have guided the work.

The different modules can be assessed by clicking the tab at the top of the window.

![Figure 3.2: Interface schema of welcome tab and menus](image)

Figure 3.2 shows the software home tab and menus.

![Figure 3.3: Interface schema of the tuner and the tuner history tabs](image)

The window on the left of Figure 3.3 presents the tuner module. It is initiated by clicking the ‘Start’ button. If a sound is detected, the corresponding characteristics of this sound are displayed on the right hand side. A line representing the error in cents is displayed on the graph.

While the module is running the ‘Start’ button is labelled ‘Stop’. Clicking the ‘Stop’ button will freeze the window display and stop the tuner module. Note that the tuner module is automatically stopped if the user clicks on another tab.

The window on the right of Figure 3.3 presents the tuner history module. The user selects a date and clicks the button ‘Get’ to display up from this date the tuner history in the left part of the window. The button ‘Clear’ erases all the history.
The window on the left of Figure 3.4 shows the timbre analyser interface. The user must record two sounds of same pitch in order to compare them. The ‘Plot Comparison’ button displays the harmonic distribution of the two sounds in the graph and the timbre percentage figures.

The user can play back the recorded sounds, thanks to the ‘Play’ buttons.

The ‘Plot in Real Time’ button only displays the harmonic distribution graph while the user is playing.

The window on the right of Figure 3.4 shows the recorder module interface. The user can record a sound. By clicking on the ‘Play’ button the recorded sound will be played back. An audio playback tracker graph displays the window below the button.

The user can choose the audio format and save the recorded file.

### 3.3 Development

#### 3.3.1 Introduction

In this section we will summarize the methodology and discuss the choice of a programming language. Secondly, we will figure out appropriate values for the main parameters to be determined for the programme to work properly. Thirdly, we will go through the integration stages and show some results. Finally, we will see the changes that have been made to produce an implemented version of the prototype thanks to a whole evaluation of the system (in Section 4).

#### 3.3.2 Methodology

As mentioned in the Introduction, the Iterative Design Methodology approach has been used to handle this project. Two prototypes have been made. The second prototype is an improvement of the first one taking in account an evaluation of functionality and usability.
3.3.3 Programming language selection

Multiple programming languages could be used for developing the software; the main ones being Matlab, Python, C/C++ and Java.

Matlab is a well-supported dedicated environment programming language. A lot of documentation is accessible online and many methods are already implemented as functions in this language. But running a Matlab script is usually far less efficient than using a programming language which has not a dedicated problem-solving environment. Furthermore, Matlab does not provide the possibility of creating a user interface. This language is a good tool for prototyping only.

At an early stage, Matlab has been used for prototyping. This approach has been undertaken to understand in a practical way how the methods were working and to check if the output was manageable. Some examples of results have been presented in Section 2.

Python is also a high level language. Some libraries like PyAudio are available to process sound in real-time. But like Matlab it is not as efficient as low level languages. As the software would have to respond to the user with an adequate reactivity, it would be more strategic to choose a tool with a better running performance.

C/C++ is a low-level language and is very efficient. Also, several external audio libraries are available online, (for example, Portaudio). However, this programming language needed to be mastered at a higher level to implement the software properly. This learning process would have been very-time consuming at the expense of the time set aside to carry out the product development.

Java was learned easily and it is more comfortable to use than C++. Java offers large and powerful libraries like Swing to implement interfaces and Awt to plot graphs. Also, it has a reasonably good performance. Hence Java was chosen as the programming language.

3.3.4 System configuration

This section presents all the arguments for the arbitrary decisions on parameters to develop the system:

Sound analyser method

Regarding the background survey on the existing techniques and the results given by the Matlab prototypes, the Fast Fourier Transform method has been chosen to analyse all sound signals.

As mentioned in Section 2.3.1, this method owes its running time of $O(n \times \log(n))$ to a data length restriction of power of two. There is no guarantee that the signal always has a convenient length. The solution is to add zero values at the end of the list to reach the desired length. This change does not prevent FFT from working well [13]. The corresponding code is available in the Powerof2.java file.

Window size

Both tuner and timbre analyser need to run in real-time. This implies an amount of the input audio data to be buffered and processed while the audio capture is running. Furthermore the window size affects the programme running time.

In the tuner module, a reaction time of half a second is chosen. This elapsed time has been approved by the users (see Section 4.2.2.3). The System.currentTimeMillis() tool is used to get the code running time. The system states that the sound processing takes on average 130 ms. This leaves on average 370
ms (500 – 130) for the audio capture. After some tests, the window bytes length is fixed at on average 37500 bytes. Note that this value may depend on the microphone reactivity as well as the computer used to run the code. All tests have been done using the laptop described in Appendix B.

In the timbre analyser module, a reaction time of a second is chosen. This elapsed time is required to read and interpret the graph shape. This time has been approved by the users (see Section 4.2.2.3). The timbre analyser processing running time is around the same as the tuner processing running time. The sound processing takes now on average 300 ms, hence a 700 ms (1000 – 300) is left for the data capture. To achieve that, the buffer length should be fixed at 125000 bytes.

Note that the 130 ms audio omission for the tuner and the 300 ms audio omission for the timbre analyser do not prevent the result correctness as the test demonstrates it in Section 4.2.1.3.

**Window shape**

As discussed in Section 2.3.3.2, the window shape chosen is the Dirichlet window.

**Note finder**

Usually, the reference A4 is at 440Hz.

The piano frequency range is the largest of all those produced by musical instruments. Therefore, it was chosen as a basis to fix the database frequency range.

The piano range is defined from A0 at 27.5 Hz to C8 at 4186 Hz. The low and the high notes are rarely used. So the database will contain 7 octaves from C1 to B7. This entails that the prototype is not programmed to recognise a frequency lower than 32.7 Hz or higher than 3951.04 Hz.

**Timbre analyser and comparator parameters**

As seen in Section 2, no general method exists to get all the timbre characteristics. Hence a set of complementary methods has to be chosen. The three chosen methods are: a graph displaying the harmonics distribution, the value of the gravity centre, and the IRR value.

* **Harmonics distribution:**

  We do not expect two same notes to have the same amplitude and exactly the same frequency. We have to use ratio in order to compare their timbre characteristics.

The method to compute the harmonics distribution can be done by the following steps. For each note:

- knowing the fundamental frequency value, estimate the harmonics frequencies,
- calculate the ratio of the harmonics frequency to the fundamental frequency,
- store that list,
- calculate the ratio of the harmonics frequency amplitude to the fundamental frequency amplitude,
- store that second list,
• plot the graph of the amplitude ratio over the frequency ratio.

In the Section 4.2.1.3 we can see that the tuner recognises perfectly the fundamental frequency $f_0$ with an error not larger than 0.1%. But in practice we cannot expect the harmonics to be exactly at $f_0 \times 2.0$, $f_0 \times 3.0$, $f_0 \times 4.0$ Hz etc. It has been observed that the error range is not greater than +- 5 Hz. So we have to search the highest amplitude in this interval.

We also have to check that the chosen range does not overlap multiple harmonics. Due to the logarithmic scale of musical note frequencies, the lowest detected frequency by the system is the one asking for the highest constraints.

Considering the lowest note:
- Fundamental: 32.7 Hz
- First harmonic: $32.7 \times 2.0 = 65.4$ Hz
- Gap: $65.4 - 32.7 = 32.7$ Hz

In the worst case we can get a range of maximum +- 32.7 Hz. So a +- 5 Hz range is a good one.

Graph design:

The harmonic distribution could have been plotted using a bar chart. However, the harmonic distribution of a sound should been considered as a general shape which describes the timbre. Using a bar chart would have caused this overall view to disappear and guided the user to compare harmonics only pair by pair. To avoid this, the design is based on a continuous representation. So, the harmonics are related by line to each other.

Typically, the amplitude of the harmonics is decreasing; this could result in a decreasing line. This is not good visually because the graph loses the idea of a harmonic discrete distribution: the number of analysed harmonics is not clearly shown. So, to overcome this problem, it has been decided that the line joining points will go back to 0 between each harmonic. Visually, the graph shape is composed of a determined number of isosceles triangles of varying height depending on the harmonics amplitude.

An example of a graphic result is presented in Figure 3.5.
Figure 3.5: Schema of the harmonic distribution graph

```java
// normalize the harm/ampl array to a ratio array
public double[][] GetRatio (double[][] harmampl){
    double[][] r = new double[5][2];
    for (int t = 0; t<5; t++){
        r[t][0] = harmampl[t][0]/ harmampl[0][0];
        r[t][1] = harmampl[t][1]/ harmampl[0][1];
    } 

    // add extra points
    double[][] ratio = new double[10][2];
    for (int t = 0; t<10; t++){
        if (t==0){
            ratio[t][0] = r[t][0];
            ratio[t][1] = r[t][1];
        }else{
            ratio[t][0] = (1/2)*1.5;
            ratio[t][1] = 0;
        }
    }

    return ratio;
}
```
The code version of this method is available in Figure 3.6. The code comes from src/TimberAnalysis.java. It includes both the GetRatio and the GetHarmAmpl functions. The graph generator can be found in the PlotSpectrum.java file.

* Gravity center and IRR values

Krimphof *et al.* [15] propose a formula to compute the gravity centre using the FFT spectrum (3.1):

\[
Gravity = \frac{\sum_{i=\text{lowest}}^{\text{highest}} f_i \times a_i}{\sum_{i=\text{lowest}}^{\text{highest}} a_i}
\]  

(3.1)

\(f\) : frequency in Hertz
\(a\) : amplitude

Krimphof *et al.* also propose a formula to calculate the IRR parameter. The formula is (3.2):
\[
IRR = \log_{10} \left( \sum_{k=2}^{n-1} \left| 20 \times \log_{10}(A_k) - \frac{20\log_{10}(A_{k+1}) + 20\log_{10}(A_k) + 20\log_{10}(A_{k-1})}{3} \right| \right)
\] (3.2)

\(A_k\) is the amplitude of the \(k^{th}\) harmonic.

These two formulae are used as defined in order to compute the gravity centre and the IRR parameter.

Regarding the comparison of two timbres, the system gives two values but no reference. The user cannot appreciate the difference signification. In order to help him/her, the system gives the percentage difference using one of the two timbres as a reference.

The percentage difference can be found by applying the following formula (3.3):

\[
\text{Percentage} = 100 - \left( \frac{|\text{model} - \text{data}|}{\text{model}} \times 100 \right)
\] (3.3)

**Audio capture format**

The prototype is dealing with audio input. In order to process the stream adequately, some parameters have to be determined.

The sampled rate is important as it can limit the detected range of sounds. According to the Nyquist theorem, the sampled rate has to be set at twice the highest frequency to avoid ear detecting aliasing [24][4].

Aliasing may occur in sound reproduction. In order to transform the sound from bytes to audible noise, the sound wave goes through several processes, electrical, magnetic and mechanical. The latter produces aliasing [20].

The human cannot hear a sound higher than approximately 20,000 Hz [20], so a rate of 40,000 Hz is good to use.

However, the higher the sampling rate the less the ear may notice aliasing. As a standard, the digital audio is typically sampled at 44.1 kHz for CD recordings or 48 kHz for professional audio applications [20].

Given the project scope, a sampled rate of 44100 Hz is sufficient.

The prototype also includes a recorder. In that part, audio encoding is needed to save the sound in a suitable format, convenient for users. It has been set as follows:

- 16 Bytes, which ensures a better capture precision than 4 Bytes. A good capture is crucial in this project.
- Signed values. Java handles values as signed by default.
- Little Endian. Java is written in Big Endian but Windows is in Little Endian [21]. It is more likely that the product will run on Windows and that the users will open audio files using a Windows application.
- WAV format. Here again, we assume the system to run on Windows, so a Windows audio format has been chosen. Furthermore WAV does not deal with compression, unlike MP3. Hence formatting problem is low.

- Mono/Stereo. This choice can be left to the user to be determined because we expect the user to understand the difference between the two formats as they are widely used. Note that the other parameters cannot be chosen. We do not expect the user to have background in computing and so to understand in what extend the parameters may affect the signal.

Microphone amplitude sensibility

The microphone amplitude sensibility has to be kept in mind in order to limit the sound process on the targeted sound. If no threshold is set, the programme will process every sound it captures, including undesired noise. To overcome this issue, it is needed to set a sensibility threshold, which may depend on the microphone being used. The aim is to state the sensibility where the sound detected comes either from an instrument or a singing voice. No lower sounds need to be processed.

Figure D.1 in Appendix D shows the results found after the production of different sounds at one metre from the microphone using both the computer and a sonometer. The sonometer is used to calibrate the system.

The sensitivity limit is fixed at an amplitude of 70 dB. It corresponds to 250.

Frequency data range and normalization of FFT

The data range should go from 1 Hz to the highest computed harmonic frequency of the highest detected note.

Highest detected note: B7 at 3951.04 Hz

Highest computed harmonics frequency: $3951.04 \times 5 = 19755.2$ Hz.

Error range: $\pm 5$ Hz.

Data range: $19755.2 + 5 = 19760.2$ Hz.

Chosen data range: 1 Hz to 19770 Hz.

In order to compute efficiently timbre features, the data are computed with a frequency precision at the unit. The FFT process does not return a normalized array of data. The method used to normalize the data array can be found in the Norm function in the file src/Normalize.java.

Other parameters

Several other, less crucial, parameters have been taken in account. This paragraph examines them.

The tuner windowing shows no more than 15 points of capture. This is a good visual balance. If 15 points have been reached, the window is moving.

The History list groups notes on a 10 minutes period for a good visualization.

The number of harmonics displayed in the Timbre comparator graph needs to be determined. As explained in [26], in practice the number of harmonics is finite and the more regular the signal the faster the decrease in harmonic amplitudes. Some tests confirm that property. They have shown that only the four first harmonics have significant amplitudes. The number of displayed harmonics has been stated at 4. The tuner module must give an appreciation of the tuning. Therefore it needs to show the limits
between sounds that are too flat, too sharp and those that are correctly tuned. In the current tuner system, the frequency error is displayed in cents [6]. The formula is (3.4):

\[ frequency\ error = 1200 \times \log_2 \left( \frac{f_1}{f_2} \right) \] (3.4)

The note is marked as tuned if the frequency error is estimated between -5 cents and 5 cents. A pitch with a frequency error lower than -5 cents is considered as flat, and a pitch with a frequency error higher than +5 cents is considered as sharp.

Furthermore current tuners show a pitch error range between 50 cents and +50 cents. The same tuning system has been used for the software.

### 3.3.5 Prototype development phases

Applying a step by step methodology, a planning has been followed to ensure the successful completion of the software.

The first phase focuses on the implementation of the FFT. It establishes the answer to the first minimum requirement. Over this step, code sources are collected to support the reading of a WAV audio file [10] and the FFT processing [28][27]. The output gives basic information on the pitch and the tuning.

Over the second phase, a first implementation of the STFT method is developed still using an audio file. In addition to the output obtained from the first stage, a graph draws the variation of the pitch over time. Graph drawing is supported by the source described in [2]. This stage fulfils the second minimum requirement.

The third phase involves the interface integration to the system. Some sources support this phase [22]. Microphone capture is included. The prototype still does not run in real-time. Inputs and outputs are shown through the interface. The last minimum requirement is addressed.

In the fourth phase, the timbre aspect is on concern. This phase includes the handling of sound inputs to output comparative figures and graphs. The first extension objective is implemented.

The fifth phase leads up the interface to run in real-time. To achieve this, the code processing the sound is included in the sound capturing part of the code. The last extension is addressed.

Over the last phase some part of the code are reviewed and restructured to ensure a good integration of all functionalities. In addition, files menus are built in.

This section has addressed Step 4 of the methodology.

### 3.3.6 Integration

**Threads usage**

The application has to support real-time processing. To avoid latency, it seems useful to think about implementing some parts of the code in threads.

By convention, the main interface has to run in a thread, as this is the way Java Swing implements graphical interfaces. Some modules such as the sound capture and the graph outputs have to be run in separate threads because they need to appear simultaneously to the user.

For good interface usability, all audio playbacks are also carried out in a separate thread.

Figure 3.7 summarizes the thread distribution.
Modules

The main parts of the system are: the interface, the sound capture, the FFT processing, the graph drawing, the data processing (such as a note finder, a tune finder, a timbre characteristics computation). This section presents the way each part has been implemented. Note that all sources have been used including significant changes in the program in order to meet the requirements. If no source is cited, no source has been used to develop the methods. Refer to the src/ folder on disc to access all program code.

The graphical interface structure implementation has been based on the Java Demo Applet [22]. It regroups the Java files:

- GUIHistory.java
- GUITuner.java
- GUIJavaSound.java
- GUIWelcome.java
- GUIRecorder.java
- JavaSoundApplet.java
- GUITimberAnalyser.java
- ControlContex.java

The files starting with ‘GUI’ implement the audio processing is composed of an audio capture based on the Java Demo Applet [22] and a post processing to convert the audio bytes into a convenient format for FFT supported by [10]. Related codes can be found in GUIRecorder.java, GUITimberAnalyser.java, GUITuner.java, MP2.java

The FFT processing consists of three parts: a pre-processing phase [27], a processing phase [28] and a post processing phase. It regroups the following files: Complex.java, FFT.java, FFTInterp.java.

The graph drawing design has been supported by [2]. Two graph types have been implemented and can be found on Plotspectrum.java and PlotStft.java. The first one is used for the timbre analysis graph and the second one for the tuner graph.
The processing of all other information regroups the files ButtonTimberAnalyser.java, ButtonTunerAna.java, TimberAnalysis.java and ToneFinder.java. Both ButtonTimberAnalyser.java and ButtonTunerAna.java handle the post processing of the interface buttons. TimberAnalysis.java implements the computation to get all features used in the timbre process. ToneFinder.java finds the tuner process features.

**Modules communication**

Communication between modules has been done by implementing getters and setters functions.

Figures F.1 to F.11 in Appendix F present the data flow diagrams showing the functionalities of the modules and how they communicate with each other.

In Figure F.6 the history array is a temporary array which stores the tuning variation of one note. This array is used to display the pitch tracking graph. When the detected note name changes, the previous note name together with the maximum tuning error contained in the history array is stored in the history.txt file. The history array is then cleared and refilled with the tuning error of the next played note.

Note that quite a few details have been excluded in the data flow figures and are taken for granted. I.e. the ‘try and catch’ statements of some processes such as reading and writing through a .txt file; the capture of audio bytes from a data line, etc.

**3.3.7 Results of prototype 1**

Some screen-shots of prototype 1 are given in Figures E.1 of Appendix E. It presents the help documentation for the programme and an example of feedback given by all the modules.

**3.3.8 Revision of prototype 1**

**Changes performed**

After the discussion on evaluations feedback in Section 4.2.2.3, it has been decided that the changes between the two prototypes should aim:

- to give the possibility to the user to choose the reference frequency;
- to restrict buttons usage in the Timbre Analyser tab;
- to remove some timbre comparison figures;
- to review the help documentation and its accessibility.

**Reference frequency choice**

As a standard, A4 is the reference. The user inputs the A4 frequency he wants as the reference. The tuner has to change the frequency data table regarding this reference frequency. The table has to be generated. For that, two formulae are applied to find out all the other note frequencies.
from the given one. The formulae are [33]:

\[ F_n : \text{known frequency} \]

For the next \( x^{th} \) note: \( F_{n+x} = F_n \times \left(2^{\frac{1}{12}}\right)^x \)

For the previous \( x^{th} \) note: \( F_{n-x} = F_n / \left(2^{\frac{1}{12}}\right)^x \)

The algorithm which generates the table can be found in the function called Database in src/ToneFinder.java.

**Prototype 2 results**

Some screen-shots of Prototype 2 are given in Figures E.2 on Appendix E.

Please see also the Visusound.jar application for a prototype running demonstration and the src/ folder of the whole code.

**3.3.9 Conclusion**

The methodology has been followed closely. This means that the system has met all the minimum requirements as well as the extensions.
Chapter 4

Evaluation

4.1 Introduction

After development, the first prototype needed to be tested. This chapter describes the whole evaluation process for the prototype described in the Design and Development chapter. To this end, we shall evaluate the system for both its functionality and for user experience.

In order to evaluate the functionalities (in Section 4.2.1) we define the evaluation criteria, and describe the methodology used to meet the criteria. Finally, we interpret and discuss the test results. This leads to our findings on the system’s limits and suggestions for improving the functionalities. This section is addressing Step 5 of the methodology.

In Section 4.2.2 we concentrate on the remarks made by the users. We define a test procedure, and then we interpret the user feedback. Thanks to this information, we will be able to improve the interface and the system functionalities.

After implementation of a new prototype, we perform a second quick evaluation of the functionalities in order to describe (in Section 4.2.3) how far the software has been improved.

4.2 Product evaluation

4.2.1 Evaluation of functionalities of Prototype 1

4.2.1.1 Criteria

The whole software functionalities are dependent on the FFT process and its use.

Evaluation of functionalities includes the testing of its crucial components:

- the precision of FFT and pitch finder,
- the software running time,
• the software integration and the interface functionalities.

4.2.1.2 Methodology

The FFT and pitch finder precision

This evaluation is about checking the value of the fundamental frequency of a played note, the correct assignment of the note name and the octave number.

The tests are done in real time using the tuner module. In addition, ‘print statements’ are included in the code to track the algorithm results at all times.

A set of ground truth sound samples has to be used to achieve an accurate evaluation. The electronic piano is expected to be tuned. Nevertheless, a commercial tuner is used to double-check the pitch. The sound capture is performed by the computer’s microphone.

The software running time

Running time may be an issue on three modules: the Tuner, the Timbre analyser and the Timbre comparator. The running time of the two first modules was already determined in Section 3.3.4, so does not need to be discussed here.

The Timbre comparator is the only functionality where the user can choose the input file length. So a running time evaluation has to be performed by varying the input audio length. Results show the algorithm limits and give an accurate reading to the user on the sound length he should record to get valid outputs.

The software integration and the interface functionalities

This evaluation includes the check of all displayed results and the possible communication between modules. No particular methodology is required for this evaluation.

4.2.1.3 Evaluation results

The FFT and pitch finder precision

Figure G.2 in Appendix G shows the note tracking precision of the programme.

Assuming an absolute error of range of ± 5 cents between the results found by the software and the commercial tuner, the table shows that:

• for the notes lower than A1, the detected note and the octave number are incorrect,
• between A1 and G#2, the note name is correct but the octave number is incorrect,
• between A2 and B6, the results are very good,
• higher than B6, the note is not detected by the system.

As all sounds are found using a unique and valid method, it was not expected that some notes would be correctly detected while others not.

To understand the source of the problem, all incorrectly detected sounds and some others have been recreated using Audacity. Each sound file has the same length as the stream processed by the
microphone. They last 400 ms. (The programme has been changed so that the system reads a WAV file. This version is very similar to the one previously made during phase 1 (see Section 3.3.5).

The results of these tests can be found in Figure G.1 in Appendix G. We can see that all Audacity sounds have been successfully detected and processed with a very low error range (less than or equal to 1% of error).

This leads to the conclusion that the problem is not to be found in the FFT procedure. It seems that the undetected sounds and the faulty processing come from a weakness in the microphone. In fact, a microphone has a sensitivity range which is usually shorter than that of human hearing. Hence not all frequencies have the same importance. It affects the frequency amplitude. The sensibility range of the used microphone has not been found. However, as discussed in [30], the usual sensibility of a laptop microphone may range from around 100 Hz to 14000 Hz. An example of microphone sensitivity graph can be found in Figure 4.1. This range matches with the accuracy limits seen in Figure G.2. Significantly changing the amplitude of low frequencies (less than 100 Hz) and high frequencies (more than 14000 Hz) is problematic in the project scope because the FFT process relies on amplitude distribution.

![Figure 4.1: Response frequency curve of a Shure SM58 dynamic microphone](image)

One way to address this problem is to obtain the microphone sensitivity range and to change the frequency amplitude accordingly. This can be done by multiplying every frequency amplitude by its corresponding sensitivity values. The resulting list is then closer to the real input sound.

However the microphone sensitivity cannot be found easily and is specific for each frequency. We cannot afford to have a vague sensitivity range because the amplitude level is crucial for detecting notes.

Up till now, the only solution has been to use a high quality microphone.

Besides, it is not expected that users will test instrument timbre on very high or very low notes as it is very difficult for the ear to appreciate the timbre quality in such extreme cases.

**Timbre comparator running time**

Figure 4.2 shows the timbre comparator running times. The graph shows that the running time is proportional to the file length, so there is no particular method to adopt for choosing the recording length.
We assume that 1 to 3 seconds is the acceptable limit for the user to wait, so the help section will indicate that 5 to 10 seconds recording time is an appropriate recording length.

Some other sounds at different pitches have been processed to see if there would be any difference in running time between low and high sounds. Looking at the table, no difference is of sufficient relevance (17 ms for a recording of 10 seconds) for this factor to be taken into account.

The table in Figure 4.2 demonstrates that the system is able successfully to process a very short file (200 ms). We could perform further evaluations to find the limit where the system is unable to process the input file due to its brevity, but it seems unnecessary for the scope of the timbre comparator module functionality. In fact, users are musicians; they instinctively sustain the sound for a couple of seconds, a time longer than 200 ms.

**The software integration and the interface functionalities.**

Tests show a good working of the interface. User feedbacks in Section 4.2.2 will complete this evaluation.
4.2.2 Evaluation of Prototype 1 by users

4.2.2.1 Criteria

In order to evaluate the prototype by users, two approaches were used.

The first one considers the evaluator as a simple user. This evaluation includes the user-friendliness of the interface, comprehension of the displayed results, the running time of each tool and the reliability of the system i.e. if it is bug-free.

The second approach considers the evaluator as a musician. It includes the usefulness of the modules, the accuracy of the results, raises a potential question on reuse of the software and offers potential improvements to the system.

Note that no particular knowledge of the physics of sound is required to test the tool.

4.2.2.2 Methodology

The tests were taken by 22 musicians playing in a non-professional symphony orchestra. All families of instruments except Percussion were represented.

Before the test, the participants were each asked to read and sign the questionnaire form, a copy of which can be found in Figure H.1 in Appendix H.

The participants tested the software separately except for the Timbre Comparator module. For this part, the testers were in pairs to enable comparison of instrumental timbres of the same instrumental type.

During the test, comments and observations were recorded.

After the test, the users were asked to complete a questionnaire, a copy of which can be found in Figure H.2.

Finally, all feedback and audio recordings were compiled and organized to simplify the interpretation.

4.2.2.3 Results

Presentation of a summary of users feedback

Figure H.3 in Appendix H shows a summary of the data collected from the questionnaire.

Analysis of feedback

Looking at the feedback from the first question of the questionnaire and using the audio recorder, it seems that the tools, and more specifically the Timbre tab, are subject to bugs.

However, the functionality tests did not raise that problem. We have seen in Section 4.2.1.3 that the running time of this section of the programme is highly dependent on the length of the output sound.

Bugs may be the result of user impatience. Providing more details and restricting the use of the button are considered adequate measures for overcoming this problem.

The interface-related question has only 2 negative feedbacks.

The interface seems intuitive enough.
It does not have to be changed.

Concerning the help section, more than half of the participants either did not find it or did not read it. Comments recorded in audio reveal that some participants experienced difficulties in finding the programmes help section, while others found the documentation but did not want to spend time reading it.

This may explain also the negative feedback comments linked to misunderstanding of unusual modules like the Timbre and the Tuner History modules.

The proposal to put the help section on a specific tab cannot be followed as the help section is not a module in itself. A solution to encourage people to read the help documentation consists of giving a better indication of where it can be found and displaying its content in a more user-friendly way.

Concerning the Tuner, more than three quarters of the testers found the interface easy to understand, accurate and running in an acceptable amount of time. However a quarter of the testers asked that the choice of reference frequency choice should be left to the user.

User wishes have been taken into account; the possibility of changing the reference frequency will be implemented.

Users did not detect any problem concerning inaccurate tuner results when processing low pitches. This was quite unexpected. There could be two explanations:

- the user did not play a very low note;
- the user paid little attention to the displayed octave note number. Musicians are not interested in learning the octave number of a note as they never need this information while playing.

The octave number will now appear as a point of information in a smaller font in the Tuner tab and will remain on the Tuner History tab. It may help the musician for easy identification of the recorded notes.

The Tuner History was generally welcomed. However, a quarter of the testers stated that they didn’t understand what it was for, and another quarter declared than it was not working properly.

Listening to the audio, some of them were expecting the Tuner History to be independent of the Tuner tab and tried to record sounds in the Tuner History module itself. This may explain the user’s confusion.

The confusion could have been avoided if more testers had read the help documentation.

Most testers found the Timbre Analyser presentation understandable but more than a quarter did not. Furthermore, the recorded comments of the testers reveal that they looked mainly at the graph, and only a couple of them commented on the percentage figures. These players found the figures hard to interpret and to use. The number of understanding/misunderstandings of the graph interpretation is balanced, 12/10. The lack of answers is interpreted as a misunderstanding. In addition, the audio recording revealed that some musicians didn’t know that a sound was composed of a fundamental and harmonics.

Problems connected with misunderstanding the graph may arise partially due to failure to read the help documentation and partially due to the vagueness of the explanation provided.
The interest for providing the percentage figures as well as a better explanation of how to interpret
the graphs will be added to the documentation. Only the Gravity Similarity will remain unchanged,
as this parameter is really correlated to brightness. The brightness of a sound is assumed to be under-
standable by any user. The other parameters will be removed from the interface. However, this part
is commented in the code so it is still possible to restore it. The relevant scripts are to be found in
src/GUITimbreAnalyser.java and src/TimbreAnalysis.java.

The Timbre Analyser playback audio is subject to bugs. Sometimes, immediately after the recording,
the playback sound was cut.

The audio problem may be due to the processing of two sound inputs at the same time. The two
threads are perhaps influencing the performance of each other.

This problem could be overcome by restricting the number of enabled buttons while the sound anal-
ysis is processing.

The Timbre Analyser running time received very good feedback. It seems to run in a reasonable pe-
riod of time.

No changes seem to be necessary for speeding up the system.

The Recorder had very positive feedback comments but one was negative. A quarter of the participants
asked that the place where the WAV file was saved should be indicated.

That will be indicated in the new prototype.

There is a very balanced ratio of Yes/No to the questions concerning the reuse of the software. It is
hard to say if, globally, the system was of interest to the participants or not. However, looking at the
answers to the question ‘Presented results are useful, why? ‘ three quarters of the testers wrote positive
comments. The user interest and curiosity regarding the prototype are certainly apparent.

Among the suggestions for improving the software, a quarter asked for the Tuner to include a way to
change the reference frequency and two comments ask explicitly for a better user guidance and provision
of an explanation of the purpose of each module.

A complete review of the help documentation has been already decided.

Conclusion

Globally, the main problem revealed by the user feedback is the lack of user knowledge to appreciate
the software purposes. This may be due to unwillingness to read the help documentation or to the short-
comings in the explanation provided. Hence familiar modules as the Tuner and the Recorder received
very good feedback while modules on more technical aspects received more negative feedback.

4.2.3 Evaluation of Prototype 2

4.2.3.1 Evaluation and results

No other evaluable functionalities have been added so no further evaluation is required.
After implementation, it was verified that the software is working correctly.
4.2.3.2 Prototype limitations

No limitations were faced while using the prototype. However we have to keep in mind that two limitations are present only if the software faces exceptional situations, namely:

- the detection of extreme pitches due to the microphone sensitivity,
- a blank period of 100 ms in the sound capture length used to process the audio stream.

4.3 Future work

Possibilities for software extensions include:

- making the software run on small devices like smart-phones,
- finding another way to visualize the features. The user evaluation has outlined that testers did not like the timbre module graph. It would be good to find a friendlier visualisation which relies less on physics or maths for way of presentation.

4.4 Conclusion

Thanks to user questionnaire feedback, the evaluation confirms that the software addresses a genuine musician need.

The evaluation also shows that the software is not subject to bugs in functionalities if it is used in an appropriate manner. This means that few changes were made to the first prototype. It could be explained by the need to provide the testers with a reliable tool so that they exclusively focus on the interface design and friendliness. It is then understandable that the changes on prototype 1 mainly concern the interface.

Besides, some software limitations became apparent, due mainly to external parameters like hardware or technological problems.

The learning aspect of the tool, including the practice of timbre and of note-tuning, is confirmed by the users.

On the other hand, the evaluation has revealed some unexpected facts.

The graph-based design is not well appreciated by the users. They did not like the scientific representation, so another visualisation disguising the mathematical aspect would be better. Visualisations have to be friendly and intuitive enough for the user to understand the contents easily.

All the tests show that users did not want to wait. They expected the programme to deliver results immediately, and it was not clear to them that processing the data could take some time. For users, the response time is very good only if it is nearly equal to zero. But this is impossible in practice.

The evaluation does not provide any estimate of correctness of the timbre analysis as the perception of the human ear is highly complex and difficult to compare. However, users show great interest in this module. They also understood that the timbre analyser software has to be used together with their own sensitivity as musicians.
Widening the range of the programme user testers to include sound specialists and musical instrument manufacturers would probably have helped provide better appreciation of the timbre analyser scope. But, due to time constraints, tests with such users were not undertaken.

As far as the project scope is concerned, the software can be deemed to be a success, as it has fulfilled its objective of providing multiple indicators for the evaluation and comparison of musical sounds.

Possible extensions of this software would be not only to address its limitations, but also to change the design to target a larger range of users, not only friends who are accustomed to figures with maths-based graphs.
Chapter 5

Conclusion

The project aim was to develop a tool to show the characteristics of musical sounds using visualization.

Course module materials have helped during the development of this tool. In particular, the techniques used for managing a large amount of data taught in the Computer Vision module have been adopted. The Software System Engineering module has contributed knowledge of the Java coding, the thread method and the interface creation. Teaching materials of the HCI module regarding testers experience has been useful while communicating with them.

The system could have implemented complex analysing tools (Wavelets) to deliver very detailed characteristics of sounds. But knowing that the users are not specialists in the domain of physics, the project focus was restricted to the teaching aspect and the easy understanding of the feedback delivered. In this context, common analyser tools have been chosen.

To address the minimum requirements, a sound analyser has been included into the software. The extension requirements have improved this software so that it has become a functional tool, adapted to the usage of musicians while practising their instruments.

The background research has gathered relevant data on both the physics and the computing aspects to address the project aim. Due to the large volume of documentation available, some selection had to be made in order to separate the feasible and understandable methods from the very complex and inoperable ones.

Java is less efficient than C++. However, Java suited the project scope and the objectives. C++ could have been used in order to perform a more detailed analyse of the sounds.

The choices of parameters and methods have been based on the results of the background research and on personally argued discussions. Personal tests have been helpful to face come through dilemmas and complex situations.

The integration stage was very important as it determined the software running time as well as the possibility of make changes between modules easy.

The first prototype already provided solutions while the second prototype resulted in a product that is ready to use.
The success of the tuner part of the product is well ensured as its implemented parts are based on quantitative and identifiable features.

The timbre analyser function is based on the processing of only two characteristics. These parameters are not exhaustive as timbre is defined by many specific characteristics which require high yield computation.

Computer science involving modelling, processing and visualisation is well adapted for leading this computation.
Bibliography


Appendix A

Personal Reflection

My interest in this project was firstly due to my background as a musician. Thanks to that, I have been able to identify the needs of a musician as I have the same while practising my instrument.

This project has given me my first opportunity to focus with autonomy on a project of long duration. I am now confident I am able to concentrate on a task over a three-months period, maintaining interest and staying on track. Furthermore, I enjoyed being involved in an investigative process in order to find solutions and achieve my goal.

I work with greater efficiency when I have a good idea of my workload. As is my practice, I paid attention to carrying out and following a carefully thought-through plan. So this allowed me to meet all planned milestones on time without stress.

During this project, I faced problems that necessitated taking important decisions regarding the project orientation. I had to make the right choices. This experience has given me an idea of how a project manager should think when confronted with dilemmas. I have understood that selecting one option always implies abandoning another, but the relative importance of this loss has to be compared with the option picked. In my project, the first critical point appeared with the selection of the FFT method against the wavelet as sound analysis technique. The second one was about choosing between Java and C++ as a programming language.

This project allowed me to improve my coding, more especially in the Java field. For example, I now understand how threads have to be implemented and used. I have also learned new coding techniques like how to capture and process sounds via a microphone.

Working on a concrete and complex subject clarified for me that collecting data, understanding and using sources is crucial. Furthermore, I became convinced of the need to understand the stakeholders profiles and needs (the musicians as users in this case) in order to address the real problem and find adequate solutions. The testers questionnaire appeared to be a good way of reaching the users. I found out what their interests were. Also I became aware of their expectations of a computer application: it has to run as fast as possible and the less there is to read, the better. The participants also gave me ideas
I would not have thought of. Concerning this aspect, it would be useful to improve the interaction with the users, thanks to a face to face meeting.

While working on my project I experienced several difficulties, mainly at the beginning.

The resources found on signal processing were massive and always maths-oriented. Very few research was devoted to investigating practical aspect of methods so I had to do these tests myself to understand each methods purpose and use. Research into all these methods was very time-consuming and required a lot of physics knowledge. I was obliged to categorise them in order to focus only on the most affordable ones.

The other difficulty I faced was the ‘blank page syndrome’ when processing audio file. I spent around three days to come out with the first class. Once the first minimum requirement (audio + FFT) had been addressed, coding became easier and then I was able to catch up with my schedule.

It was possible to overcome the above mentioned difficulties by managing well my work.

Firstly, even if the subject was complex, I decided to limit my project to accessible objectives. I preferred to extend the scope step by step rather than cut it back on discovery that I might not be able to address it fully. I decided to focus the sound analysis only on the timbre and the pitch of one note.

Then, thanks to my background in physics (from high school) and music (playing an instrument), I knew from the beginning that sound analysis was a very complex field not entirely mastered by scientists. This is why I decided to base my work on methods that had already proven effective (FFT, gravity centre - brightness).

Finally, a large part of my project required programming. I preferred to use Java because I knew what this language offers and that it was able to fulfil my project programming objectives. Furthermore, knowing the small amount of time available to complete the product, I could not afford to spend time in mastering another programming language as much as Java.

I have the satisfaction of being able to manage my project right to the end and to have fulfilled my objectives.

The final product may still be enriched but it is already a new tool meeting some musicians needs.

While doing my background research, it has been confirmed that in addition to my computing knowledge, it was also necessary to concentrate on physics, mathematics, electronics and even biology. I would have appreciated to have the time and the opportunity to communicate with students from relevant schools of the University of Leeds. In any case, this project has helped me understand what skill are necessary to succeed in such a project, in particular technical expertise, level of detail management, schedule management, communication and team work.
Appendix B

External Materials

As mentioned previously in the report, five external code materials were utilized in the software development, comprising:

- implementation in Java of the FFT method [28] with its complex number handler class [27]. This code has been changed in the final product so that it could also perform STFT on a data array.

- a class handling the reading of a WAV file [10]. In the final product, this source has been modified in order to handle an AudioInputStream.

- a class to display graphs [2]. Only the code providing the drawing of the graph has been used in the software conception. The code has been subject to a number of modifications as the graph has to be displayed in the main interface and has to communicate with other code parts.

- the Java Demo Applet [22]. This resource has been used to provide the structure of the software interface. All the Java Demo Applet modules have been greatly changed except for the recorder module, subject to fewer changes to fulfil the product specifications. This resource has been of great help to capture audio via a microphone.

The laptop used to run all tests is a:

- HP laptop, Ubuntu 12.04 64-bit

- Processor : Intel Core i5 CPU 2.60Ghz

- Total RAM : 3.9 GB
Appendix C

Ethical Issues

The evaluation based on users is the only part of this project that is subject to ethical issues. This issue was addressed by presenting the Project Information Sheet form to all participants and getting it back signed.

An example of this form can be found in Figure H.1 in Appendix H.
Appendix D

Evaluation Tables

<table>
<thead>
<tr>
<th>Sound description</th>
<th>Amplitude found by the sound meter (dB)</th>
<th>Amplitude found by the program (unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no sound</td>
<td>39</td>
<td>20</td>
</tr>
<tr>
<td>talking</td>
<td>53</td>
<td>100</td>
</tr>
<tr>
<td>talking high</td>
<td>70</td>
<td>250</td>
</tr>
<tr>
<td>Wind instrument playing (trumpet)</td>
<td>99</td>
<td>1675</td>
</tr>
<tr>
<td>Reed instrument playing (clarinet)</td>
<td>93</td>
<td>1144</td>
</tr>
<tr>
<td>String instrument playing (violin)</td>
<td>88</td>
<td>812</td>
</tr>
</tbody>
</table>

Figure D.1: Amplitude of a sound produced at 1 m from the microphone
Appendix E

Prototypes Screenshots

Figure E.1 presents 7 screen-shots from prototype 1.
Figure E.2 presents 10 screen-shots from prototype 2.
Welcome to Visu-Sound

This software is a prototype of a sound analyser tool developed in the scope of a final year project.

How to run it?
On Windows:
Double click the VisuSound.jar file

On Ubuntu:
wget http://example.com/VisuSound.jar

What is it?
The program is constituted of a tuner, a timbre analyser and a recorder.
Note that the reference frequency is the A at 440 Hz.
The modules are accessed by clicking on the tabs at the top of the window.

How does it work?
Welcome tab:
A presentation window

Tuner tab:
A real-time pitch tracker recording history. Press Start to initiate it. While playing a graph, outputs the evolution of sound and the frequency played as well as the note name and scale, the tuning and the interval to the tuned note are displayed on the right (Result section) of the window.

Tuner history tab:
A tab where the recorded tuner history is displayed. Select a date and press Get to show the history. From this date, tuned note appears in black, sharpened in red and flattened in blue. Press Clear to erase all history.

---

**Result**

Ref: A 440

**Frequency:** 443.0 Hz

**Note:** A1

**Tuning:** A bit sharp.

**Interval:** 11.76 Cents
Figure E.1: Screenshots of the interface of Prototype 1
Welcome to Visu-Sound

What is it?
The program is constituted of:
- A Tuner, a real time pitch tracker recording history.
- A Tuner History, the recorded tuner history is displayed.
- A Timbre Analyser, composed of a timbre analyser in real time and a timbre comparator on 2 sounds.
- A Recorder, to record audio in mono or stereo.

How to run it?
On Windows:
Double click the VisuSound.jar folder
On Linux:
write: java -jar VisuSound.jar

How does it work?
The modules are accessed by clicking on the tabs at the top of the window.

Welcome tab:
- A presentation window
- Press Start to initiate it.

Tuner tab:
- Choose the reference frequency (by default at 440 Hz)
- Frequencies: 735.0 Hz
- Note: F# 5
- Tuning: A bit flat.
- Interval: -11.71 Cents
Figure E.2: Screenshots of the interface of Prototype 2
Appendix F

Data Flow Diagrams Describing the System

Figure F.1: FFT PROCESS
Figure F.2: ANALYSIS ON TIMBRE

Figure F.3: GRAPH
Figure F.4: CAPTURE SOUND

Figure F.5: PLAYBACK SOUND
Figure F.6: Tuner Module

Figure F.7: Tuner Tab
Figure F.8: Tuner History Module Tab

Figure F.9: Timbre Analyser Tab
Figure F.10: Recorder Tab

User clicks "Record"

CAPTURE SOUND

User clicks "Stop"

Stop sound capture

User clicks "Pause"

Freeze window

User clicks "Play"

PLAYBACK SOUND

Set and display graph

Output sound playback and graph

User clicks "Save"

Get user Audio Input Stream format preference (default "Mono")

Format to WAV

Get desired file name from user (default "Untitled")

Save WAV on disc

Refer to appropriate tab diagram

Figure F.11: Overall interface and Welcome Tab
Appendix G

Evaluation on Pitch Finder

<table>
<thead>
<tr>
<th>Octave number</th>
<th>Note (Pitch)</th>
<th>Frequency [Hz]</th>
<th>Octave found</th>
<th>Note found</th>
<th>Frequency found</th>
<th>Relative Error of frequencies in Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>32.70</td>
<td>1</td>
<td>C</td>
<td>32.97</td>
<td>0.83</td>
</tr>
<tr>
<td>1</td>
<td>CH - Db</td>
<td>24.42</td>
<td>1</td>
<td>CH - Db</td>
<td>24.21</td>
<td>0.98</td>
</tr>
<tr>
<td>1</td>
<td>D</td>
<td>36.71</td>
<td>1</td>
<td>D</td>
<td>37.00</td>
<td>0.79</td>
</tr>
<tr>
<td>1</td>
<td>DM - Eb</td>
<td>38.89</td>
<td>1</td>
<td>DM - Eb</td>
<td>39.02</td>
<td>0.33</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
<td>41.20</td>
<td>1</td>
<td>E</td>
<td>41.04</td>
<td>0.39</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>43.65</td>
<td>1</td>
<td>F</td>
<td>43.74</td>
<td>0.21</td>
</tr>
<tr>
<td>1</td>
<td>FM - GB</td>
<td>46.25</td>
<td>1</td>
<td>FM - GB</td>
<td>46.43</td>
<td>0.39</td>
</tr>
<tr>
<td>1</td>
<td>G</td>
<td>49.00</td>
<td>1</td>
<td>G</td>
<td>49.12</td>
<td>0.24</td>
</tr>
<tr>
<td>1</td>
<td>G# - Ab</td>
<td>51.91</td>
<td>1</td>
<td>G# - Ab</td>
<td>51.81</td>
<td>0.19</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>55.00</td>
<td>1</td>
<td>A</td>
<td>55.17</td>
<td>0.31</td>
</tr>
<tr>
<td>1</td>
<td>AA - Bb</td>
<td>58.27</td>
<td>1</td>
<td>AA - Bb</td>
<td>58.54</td>
<td>0.36</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>61.74</td>
<td>1</td>
<td>B</td>
<td>61.90</td>
<td>0.26</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>65.41</td>
<td>2</td>
<td>C</td>
<td>65.27</td>
<td>0.21</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>82.41</td>
<td>2</td>
<td>E</td>
<td>82.09</td>
<td>0.39</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>110.00</td>
<td>2</td>
<td>A</td>
<td>109.66</td>
<td>0.29</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>146.83</td>
<td>3</td>
<td>D</td>
<td>146.69</td>
<td>0.10</td>
</tr>
<tr>
<td>3</td>
<td>G</td>
<td>195.00</td>
<td>3</td>
<td>G</td>
<td>195.8</td>
<td>0.16</td>
</tr>
<tr>
<td>4</td>
<td>CH - Db</td>
<td>277.18</td>
<td>4</td>
<td>CH - Db</td>
<td>277.24</td>
<td>0.02</td>
</tr>
<tr>
<td>4</td>
<td>FM - GB</td>
<td>369.99</td>
<td>4</td>
<td>FM - GB</td>
<td>370.1</td>
<td>0.03</td>
</tr>
<tr>
<td>4</td>
<td>AA - Bb</td>
<td>466.16</td>
<td>4</td>
<td>AA - Bb</td>
<td>466.32</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>CH - Db</td>
<td>554.37</td>
<td>5</td>
<td>CH - Db</td>
<td>554.48</td>
<td>0.02</td>
</tr>
<tr>
<td>5</td>
<td>FM - GB</td>
<td>739.99</td>
<td>5</td>
<td>FM - GB</td>
<td>740.2</td>
<td>0.03</td>
</tr>
<tr>
<td>6</td>
<td>DM - Eb</td>
<td>1224.51</td>
<td>6</td>
<td>DM - Eb</td>
<td>1224.21</td>
<td>0.02</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>1975.55</td>
<td>6</td>
<td>B</td>
<td>1975.67</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>2093.00</td>
<td>7</td>
<td>C</td>
<td>2092.76</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>CH - Db</td>
<td>2217.24</td>
<td>7</td>
<td>CH - Db</td>
<td>2217.24</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>E</td>
<td>2793.83</td>
<td>7</td>
<td>E</td>
<td>2793.93</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>AA - Bb</td>
<td>3879.81</td>
<td>7</td>
<td>AA - Bb</td>
<td>3879.38</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>3951.07</td>
<td>7</td>
<td>B</td>
<td>3951.34</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Figure G.1: Evaluation on pitch finder using artificial sound created by Audacity
| # | J | C | G | F | E | D | C | G | F | E | D | C | G | F | E | D | C | G | F | E | D | C |
| 1 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

This table shows the musical notation for various notes.
Figure G.2: Evaluation on pitch finder using an electronic piano and a commercial tuner
Appendix H

Questionnaire Cover Sheet and Questionnaire Form

Please see next page.
Project Information Sheet

Project Title: Frequency and spectrum analysis to compare sounds.

You are being invited to take part in a student project. Before you decide, it is important for you to understand the aim of the project and what participation will involve. Please take time to read the following information carefully and discuss it with others of your wish. Ask if there is anything that is not clear or if you would like more information. Take time to decide whether or not you wish to take part. Thank you for reading this.

Project Aim:
The project involves a research in signal processing. It aims to develop a new software and its user interface of a tuner and a sound analyser which presents a new way to visualise sounds. This project lasts 4 months.

Why have I been chosen?
Any participant can take part of this survey. You have been chosen randomly. The participation of about 20 persons is required.

Do I have to take part?
It is up to you to decide whether or not to take part. If you do decide to take part you will be given this information sheet to keep and you can still withdraw at any time. You do not have to give a reason.

What will happen to me if I take part?
You will be asked to test the software by completing tasks for 7 to 14 minutes. While testing, please “think out loud”, your says will be recorded. After that, you will be asked to answer a quick questionnaire of 10 questions about your user experience.

Will my participation in this project be kept confidential?
All personal details as name, age etc... will be kept confidential. Questionnaire answers and a transcript of what you said related to the software evaluation are the only data which are going to be used and published.

What type of information will be sought from me and why is the collection of this information relevant to achieve the project’s objectives?
Questionnaire answers related to the software evaluation are going to be used and published in order to improve the software user interface thanks to your feedback. Only the transcript of the recording will be published, the audio will not. The audio recording will be of great help as it will guide the student to understand how users act using the software and improve their experience.

What will happen to the results of the project?
The results of this project will be published in a report to be submitted for assessment at the end of the undergraduate module COMP3860 Research Project in the School of Computing at the University of Leeds.

Contact for further information:
Please contact: sc1levt@leeds.ac.uk

If you decide to participate in this project, you will be given a copy of this information sheet. Thank you very much for taking the time to read this information sheet.
**Questionnaire form**

**Introduction:**

This prototype presents a new interface for a tuner and a sound analyst.

**Task:**

Please try each tab functionality of this software for 10-15 minutes and answer the questionnaire below.

**Guidelines:**

- **Welcome tab:** 15 seconds
- **Tuner tab:** 4 minutes
- **Tuner History:** 3 minutes
- **Tuner Analyser:** 4 minutes
- **Quick Recorder:** 2 minutes

**Tick the box as appropriate:**

<table>
<thead>
<tr>
<th>Question:</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>The tool is bug free.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This interface is easy to use.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>While using the “help” section, it was informative.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tuner Tab:</strong> The result presentation is easy to understand.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tuner History Tab:</strong> The result presentation is easy to understand.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tuner Analyser Tab:</strong> The result presentation is easy to understand.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presented results are useful, why?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Question:</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tuner Tab:</strong> The tool is accurate.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tuner Analyser Tab:</strong> The notional timber differences make sense.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tuner Tab:</strong> The system is able to present results quickly enough to be useful</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tuner Analyser Tab:</strong> The system is able to present results quickly enough to be useful</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Circle the appropriate word:**

I would use this software once it is released: **Yes** or **No**

**General question:**

Any suggestion to improve the system?

---

**Figure H.2: Questionnaire**
<table>
<thead>
<tr>
<th>Question:</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>The tool is bug free.</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>1</td>
<td>2: one bug 3: N/A 1: many bugs 1: good if user waits while analysing is long</td>
</tr>
<tr>
<td>This interface is easy to use.</td>
<td>0</td>
<td>0</td>
<td>18</td>
<td>4</td>
<td>1: Timbre tab is confusing 1: Good if the “Help” is read</td>
</tr>
<tr>
<td>While using the “help” section, it was informative.</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>2</td>
<td>12: N/A 2: better explanation is needed 2: better design 1: should be more visible</td>
</tr>
<tr>
<td>Tuner Tab: The result presentation is easy to understand.</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>7</td>
<td>1: explanation is needed 1: larger test 1: N/A</td>
</tr>
<tr>
<td>Tuner History Tab: The result presentation is easy to understand.</td>
<td>1</td>
<td>3</td>
<td>10</td>
<td>4</td>
<td>3: Didn’t work 4: N/A</td>
</tr>
<tr>
<td>Timber analyser Tab: The result presentation is easy to understand.</td>
<td>0</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>7: explanation is needed 1: N/A</td>
</tr>
<tr>
<td>Presented results are useful, why?</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>8: good to see harmonics of a sound 3: N/A 2: music is intuitive, not scientific 2: good to compare timelines 2: good as “like a teacher” 1: good for specialist (not me) 1: good for testing others 1: good to promote instrument quality 1: used standards for timber analyser</td>
</tr>
<tr>
<td>Tuner Tab: The tool is accurate.</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>4</td>
<td>4: N/A 1: multiple results for 1 note 1: fundamental note not found</td>
</tr>
<tr>
<td>Timber Analyser Tab: The noticed timbre differences make sense.</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>5: N/A 1: not working 1: yes, if the “help” section is read</td>
</tr>
<tr>
<td>Tuner Tab: The system is able to present results quickly enough to be useful</td>
<td>0</td>
<td>2</td>
<td>12</td>
<td>8</td>
<td>2: lagging</td>
</tr>
<tr>
<td>Timber analyser Tab: The system is able to present results quickly enough to be useful</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>9</td>
<td>2: N/A 1: should run faster 1: it is fine, but need to be aware of it</td>
</tr>
<tr>
<td>Quick Recorder Tab: The system is able to present results quickly enough to be useful</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>10</td>
<td>7: Where is the WAV file saved?</td>
</tr>
</tbody>
</table>