Investigating the Feasibility of Near Real-Time Music Transcription on Mobile Devices

Barry van Oudtshoorn

This report is submitted as partial fulfilment
of the requirements for the Honours Programme of the
School of Computer Science and Software Engineering,
The University of Western Australia,
2008

Abstract

Converting music from an audio signal into its abstract (notated) form is an extremely complex task. With increased processing power comes the possibility of more complex, and accurate, analysis. The aim of this project is to investigate the feasibility of developing a system designed to run in near-real time on mobile devices, which have limited processing power. Two analysis techniques are described: a windowed Discrete Fourier Transform (DFT) technique, and a sliding window DFT, both of which are well-defined and computationally efficient. Computationally expensive techniques, such as wavelets, are not considered, as their processing requirements are prohibitively high for mobile devices. Ultimately in the transcription of audio signals, there is a trade-off between accuracy and processing requirements; the goal, then, is to maximise accuracy and minimise the computation needed. To attain this goal and determine the feasibility of developing a mobile system for transcription, a modular prototype system was developed for desktop computers, and is fully explained in this project. Through experiments conducted using this prototype system, it is shown that a system capable of automatically transcribing music within the context of a mobile device is certainly feasible.

Keywords: Honours, music, notation, signal processing, mobile devices, transcription
CR Categories: H.5.5, J.5

Acknowledgements

I would like to thank my project supervisor, Dr Du Huynh to whom I am indebted for her knowledge of signal processing and Matlab, and her ability to explain concepts clearly and understandably. My thanks also to my family, who have supported me throughout this project, and understood my need to spend hours in my room without human interaction. Finally, I must thank my fiancée, Ariel, who has been a source of constant encouragement and understanding.

Contents

List of Tables

List of Figures

Listings

CHAPTER 1

Introduction

Music is a uniquely human phenomenon. In 1871, Darwin [3] observed that man's musical abilities "must be ranked amongst the most mysterious with which he is endowed". In 1985, computer music became a market-centred enterprise, and, with the accompanying reduction in the price of producing music, there was an increase in the global availability and output of music [8]. Yet music still tends to exist in one of two forms: the abstract, or notated, form and as an audio signal. Moving from notation to audio is a relatively simple task, given a computer or a musician. However, moving from an audio signal to an abstract representation of the music, is a much more complex task. Systems exist which are designed to do just that, but they suffer from two major flaws: their inability to work on audio signals in real-time, and their reliance on a powerful computer on which to run.

Clearly, then, a system capable of transcribing music in near real-time on a mobile device would not only be a step forward technically, but also of great use to composers and other musicians. The focus of this project is to investigate the feasibility of developing such a system, by producing a computer-based prototype, written in Java. The prototype will use a modular approach, allowing different algorithms to be used in different parts of the system for comparative analysis. Using this prototype a fixed set of musical phrases will be performed by a variety of waveforms and instruments, running under a number of different algorithms and settings.

The prototype will produce results in XML, using a customised Document-Type Definition (see Appendix B). XML is well-suited to this task, as it is well-supported, human-readable, and can cleanly encapsulate all of the outputted data. Significantly, producing results in XML will allow the prototype to defer the actual presentation of the results to any one of a number of third-party applications. In other words, the prototype will not produce output which can be immediately used by a performer, but which will require conversion into a more conventional form of notation. This process is illustrated in Figure 1.1; the audio signal is read in by the prototype, XML is output, and this XML is then converted into conventional notation by an external application. As many

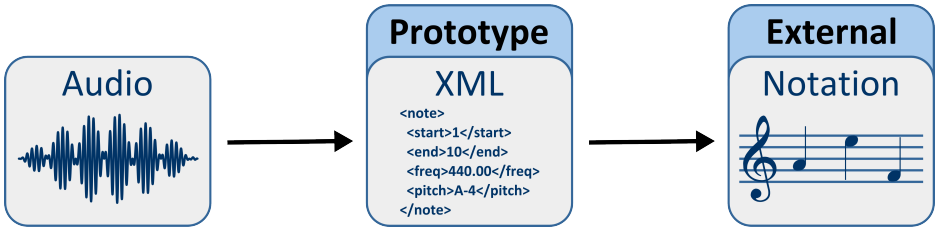


Figure 1.1: Converting audio into notation using the prototype

notation applications already support the MusicXML definition [12], they may easily be modified to accept the much simpler output of the prototype.

The prototype developed will analyse data using two distinct techniques: a windowed discrete Fourier transform, and a sliding window discrete Fourier transform. The reason these techniques will be used is that they are well-defined and relatively computationally efficient, making them suitable candidates for use in a mobile device context. These techniques are discussed in greater detail in Section 3.4.2. This report details the development of the prototype.

It is hypothesised that an automated system for the near real-time transcription of music capable of running on a mobile device is a feasible proposition. It is further hypothesised that whilst the simplest analysis techniques may have the lowest processing requirements, their accuracy will not be sufficient within the given context.

In Chapter 2, an overview of previous research in the area of computer-based music transcription is provided, and several papers are reviewed in the context of this project. Chapter 3 provides a layman's introduction to the concepts of pitch, frequency, and musical waveforms. Also in Chapter 3, the structure of the prototype developed is described in greater detail, and the analysis techniques are outlined. The experiments conducted and their results are provided in Chapter 4. Finally, Chapter 5 draws together the results of the experiments, and shows that they do support the feasibility of a mobile implementation.

CHAPTER 2

Literature Review

2.1 Overview

The six papers considered in this review each include a wealth of information not found in other papers, despite being in the same area of automated music transcription. Alm et al. [1] deal primarily with the analysis of the frequency content of sound; their paper includes discussion on the Fourier series, spectrograms, and scalograms. Klapuri et al. [6] include a useful discussion on musical metre estimation, and also discuss pitch estimation. Lepain [7] provides a method for "fingerprinting" a piece of audio based on its harmonic content, whilst Paiva et al. [11] elucidate a method for extracting the melody from polyphonic music. The technical report by Gómez et al. [5] not only summarises some of the methods employed by other researchers, but also provides useful experimental evidence for the superiority of Fourier-based techniques over three more novel approaches. Finally, Steedman [14] provides a background on consonance (including tuning and equal temperament) and harmony.

2.2 Frequency Analysis

Underpinning any attempt at transcribing music from an audio stream is the analysis and decomposition of the stream to obtain its frequency information. That is to say, we must determine which pitches are being played before we can consider how we may make sense of these data. A variety of techniques exist for performing this analysis, including Fourier analysis, spectrogram-based analysis, and scalogram-based analysis. To understand these analyses, however, it is necessary to first understand pitch and frequency.

2.2.1 Pitch and Frequency

Alm et al. [1] provide a solid overview of the relationship between pitch and frequency. In their article, they demonstrate that a pure tone with a single frequency (such as, for example, the tone represented by the function $f(x) = 100\sin(2\pi 440x)$ will have a single related pitch (in our example, 440Hz, or A_4). They note, however, that the sounds produced by musical instruments may be viewed as being composed of a number of pure tones (which alter over time). There is typically a *fundamental* tone (such as 440Hz) which dominates, and which interacts with other tones that tend to be at integral multiples of this fundamental (such as 880Hz and 1320Hz); these weaker tones are termed *overtones*. It is these overtones which give each instrument its unique timbre.

In traditional Western music, pitches are grouped in *octaves*; each octave consists of twelve pitches. Mathematically, the octave above a note of frequency f is at a frequency of $2f$. There exists a plethora of different ways of dividing up the octave; the most widely-used model in Western music is *equal temperament*. Steedman [14] points out that in this scheme, the pitches are tuned to the ratio of $\sqrt[12]{2}$. It is interesting to note that there is little literature which concerns itself with any tuning other than Western; indeed, most authors go so far as to unquestioningly accept equal temperament as the only tuning worth considering. Although many of the techniques outlined may be applied to other systems of tuning, this is not made explicit.

2.2.2 Fourier-based techniques

Fourier analysis allows us to extract the frequency information from a signal. If we take the Fourier spectrum of a sine wave oscillating at 440Hz and plot it, we will be able to observe a "spike" at 440Hz and nothing else. Alm et al. [1] provide a very good graphical representation of this. It is clear that an understanding of Fourier analysis is important to an overall understanding of the field; indeed, the vast majority of papers assume familiarity with it, except for Alm et al. who go to any length in explaining it.

By itself, Fourier analysis is inadequate in analysing a series of consecutive notes, as it deals with the entire signal as a whole. To this end, *spectrograms* have been developed. A spectrogram makes use of "windowed" Fourier analyses. The spectrograms reproduced in Alm et al. [1] illustrate this technique clearly. *Scalograms* are an extension of spectrograms: they discretise the frequencies, making analysis simpler. Fourier analysis is, then, a well-understood and established technique for frequency analysis, and shows great promise for this project.

2.2.3 Other frequency analysis techniques

The technical report by Gómez et al. [5] summarise the techniques used in four other papers for audio transcription and other pitch-related operations. These include the use of Fourier transformations which we have already seen, the Cochlear model, multirate filterbanks, and frame-based autocorrelation combined with high-pass filtering. (These will not be considered here due to their inapplicability to this project; the interested reader should consult Gómez et al. for further detail.) This report also tests the accuracy of each of these techniques; in showing the superiority of Fourier transformations in this group, the authors enable researchers to then focus on the accuracy of other, novel methods of analysis.

In addition to experimental results, the report details the methods used for conducting the experiments. It thus allows these methods to be applied in new contexts, further developed, and themselves tested. In conducting research, such a stepping-stone should not be cast aside lightly.

2.3 Feature Computation

Having obtained the frequencies by whatever means, it is then necessary to derive information from these raw data. It is at this stage that many of the papers begin, including Paiva et al. [11]. In essence, the system proposed by Paiva et al. [11] attempts to strip "unimportant" pitches from the signal (such as ghost octaves, which arise from the overtones as outlined in Section 2.2.1). Whilst this methodology is intriguing, it is not directly applicable to my proposed project, which focuses primarily on monophonic signals.

Lepain [7] provides a list of "desired properties for pitch detection algorithms", where the majority of listed items are in the area of feature computation. These include tolerance of slight harmonic mistuning; tolerance of the absence of harmonics (overtones); the ability to follow frequency modulations (such as vibrati or glassandi); and tolerance for imperceptible erratic fluctuations. It is interesting to note that the system proposed by Lepain, PHISME (*Projection des Hauteurs par Itération de Soustraction de Modèles Exponentiels*), is not at all targeted towards the transcription of music; rather, it is designed for use as a "fingerprinting" mechanism which makes use of the music's harmonic content (irrespective of the key in which it is played). Despite this, the paper includes a wealth of information relevant to this project.

In deriving information from frequency data, it becomes necessary to employ some elements of tuning and musical theory. Steedman [14] provides useful infor-

mation on equal temperament, and notes that due to its general use in Western music, the "ideal" relations between pitches do not hold. For instance, a perfect fifth (two notes separated by seven semitones) *should* have a ratio of 3/2; however, under equal temperament, this ratio is skewed somewhat, as the notes in an octave are tuned to a ratio of $\sqrt[12]{2}$. It is necessary to use this knowledge when determining which note is playing; for example, does a frequency of 435Hz represent an A_4 or $A\flat_4$?

2.4 Transcription

Describing the features found (see Section 2.3) in an abstract manner such that a human or computer may use them for analysis, performance, or modification is known as transcription. (It is worth noting, however, that the term "transcription" is also used to describe the process of conversion as a whole.) Transcription itself is perhaps the most neglected area in all of the papers. Whilst some form of audio-independent notation is alluded to in both Paiva et al. [11] and Lepain [7], there is no formal specification. This suggests that the systems developed produced output which still required further analysis (or manual adjustment) to determine musical properties and events such as tempo, time signature (and changes thereof), key signature, modulations, and expressive marks. Indeed, it is only Klapuri et al.'s [6] paper which makes mention of such properties at all. The method proposed by Klapuri et al. is useful in that it is a "bottom-up" approach. First, the temporal position of each note is found; using these data the *tata* (which are the temporally shortest pulses) are estimated. These tata then allow the beat, or *tactus* (the so-called "foot-tapping" rate) to be estimated. Using these data in turn, the length of measures (or bars) may be determined. Combining all of these together, the time signature of a piece may be determined fairly accurately, as well as any changes in time signature that may occur. As well as traditional time signatures, such as 4/4 and 3/4, more "exotic" time signatures such as 13/8 may be found using this technique as well, as long as they have a steady pulse. Despite discussing metre estimation, however, Klapuri et al. [6] fail to take the next logical step: transcription.

2.5 Conclusion

Converting a piece of audio into an abstract form (such as notation) is a multi-stage process. Whilst there exist a variety of different techniques for dealing with the problem, they all tend to proceed in the same general stages:

- Extract frequency information from the audio stream;
- Derive information from these frequency data; and finally,
- Use this information to produce the abstract form.

Five of the six papers discussed in this literature fall into these categories. The exception is Steedman [14], which provides a general overview of musical theory as it applies to computation. Alm et al. [1] and Gómez et al. [5] cover frequency analysis in their papers, and both Paiva et al. [11] and Lepain [7] discuss feature computation. Finally, Klapuri et al. [6] provide the most in-depth discussion on the transcription of the features determined.

CHAPTER 3

Materials and Methods

3.1 Overview

In this chapter, Section 3.2 describes the relationship between pitch and frequency, and the concept of overtones in musical tones is outlined. The structure of the prototype developed is discussed in detail in Sections 3.3 and 3.4, with attention paid to each of its four relatively independent modules. The analysis techniques that were implemented are explained, as well as an additional analysis technique which is the subject of future work.

3.2 Pitch and Frequency

Pitch and frequency are very similar concepts. If one considers a note as a simple sine wave, the frequency of this note is the number of cycles it completes within a certain period of time; generally, this is shown in cycles per second, or Hertz (Hz). Frequency, then, is a continuous value. Pitch, by contrast, is discrete: the pitch of a note is its agreed frequency. The note A_4, for example, (as illustrated in Figure 3.1) is generally now considered to have a frequency of 440Hz. It should be stressed that this frequency is in no way a function of the note itself, but merely the arbitrary and widely-accepted value associated with that note.

In music there are usually many pitches available, generally with recurring

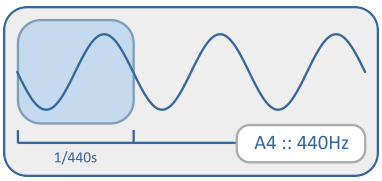


Figure 3.1: The note A_4, with a frequency of 440Hz

patterns. In Western music, for example, pitches are grouped into lots of twelve, with each grouping termed an *octave*. The final note in one octave is the starting note of the next octave, and, broadly speaking, is twice the frequency of the initial note. That is to say, if a note A exists in an octave, the note A' in the following octave will have twice its frequency. Within an octave, the relationship between two notes is less clearly defined. The distance between two notes is known as an interval; these sizes of these intervals are determined using a variety of tuning mechanisms. Nowadays, the most widely-used, generally applicable, and mathematically simple tuning is known as Equal Temperament.

Using an Equal Temperament tuning, it is assumed that the frequency of a note is twice that of the note an octave below. The notes between these are tuned in the ratio of $\sqrt[d]{2}$, where d is the number of different notes in the octave (also termed the *degree* of the scale). In other words, the frequency f of a note n degrees above a given fundamental frequency g is given by

$$f = g2^{\frac{n}{d}} \tag{3.1}$$

where the fundamental frequency normally assumed is that of the note A_4; 440Hz. Equal Temperament is by no means the only tuning scheme both historically and in current use; the interested reader will find a great deal of information on tunings by consulting Wolf [16], Nolan [10], and Bosanquet [2].

In music, it is rare to have clear boundaries between notes. Notes tend to blur and overlap each other, both because of playing style (not releasing a note until the next note has already started sounding), and because of environmental factors, such as echo and reverberation. In addition to this, most music is (to varying degrees) polyphonic; that is, more than one note may be played simultaneously. When two notes are played together, the resultant waveform is altered dramatically, as illustrated in Figure 3.2. In this figure, the combination of two notes with different frequencies produces a waveform with a shape bearing little resemblance to either of its component parts. This phenomenon occurs because the waveforms periodically interfere with each other.

As outlined by Nikolova et al. [9], musical notes generally exhibit *overtones*. In stringed instruments, these arise because the string tends to vibrate in its entire length (producing the fundamental frequency), and in equal sections of the string. A similar effect occurs in wind instruments. Figure 3.3 shows a note with a fundamental frequency of 440Hz, as indicated by the strong peak at this frequency. Note, however, the peak at 880Hz; this is an overtone.

The goal of this project, then, is to determine the constituent parts of a resultant waveform, in order that the various notes that make up a musical signal may be determined, including their starting and ending times. However,

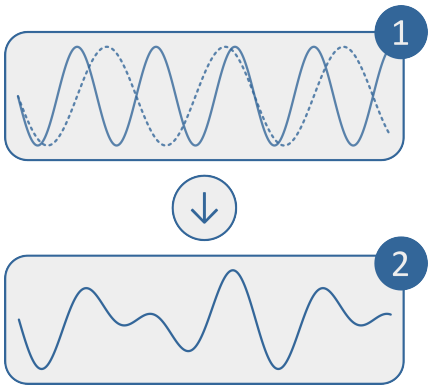


Figure 3.2: Two notes played simultaneously, and the resultant waveform

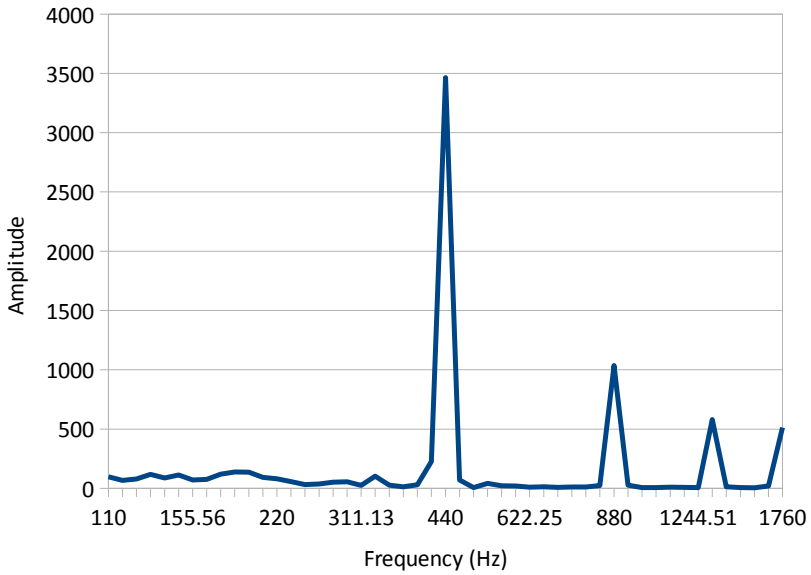


Figure 3.3: Relative amplitudes of frequencies present in a musical tone

it is necessary to distinguish between two notes of differing frequencies and a single note with overtones. The ultimate goal is to investigate the feasibility of developing a music transcription system which is capable of running on mobile devices in near real-time, based on the results achieved using a computer-based prototype system (described in Section 3.3).

3.3 The Prototype

In order to determine whether or not it is feasible to develop a mobile device-based music transcription system, a prototype was developed for this Honours project. This prototype was designed to facilitate the determination of the optimal direction to take in developing a mobile-based system. The prototype developed was a wholly computer-based system, in order to make testing, development, and experimentation as simple as possible. The prototype's architecture is outlined in Section 3.4.

The prototype reads in an audio signal, applies one of a variety of analysis techniques to it, and outputs an XML file containing the extracted information. It was developed entirely in Java, using only standard Java libraries; by doing this, the system is easily portable to different Java environments, such as those available on mobile devices (for example, the J2ME). It is important to note that the prototype was built as a means to determine the optimal way in which to put together the final system to maximise accuracy and minimise processing. As such, it is not strictly real-time, in that it does not have microphone support. Internally, however, real-time processing is simulated by the audio streamer.

The prototype includes a variety of utilities designed to make experimentation simple. These include a basic system for handling multiple files sequentially and the ability to produce output in a variety of formats, including:

- Graphically, as a plot of pitch against window number (time);
- In XML, as outlined in Section 3.4.4; and
- General statistics on the processing required for the analysis of each file in each module of the prototype.

These may then be used as the basis for experimentation and statistical analysis.

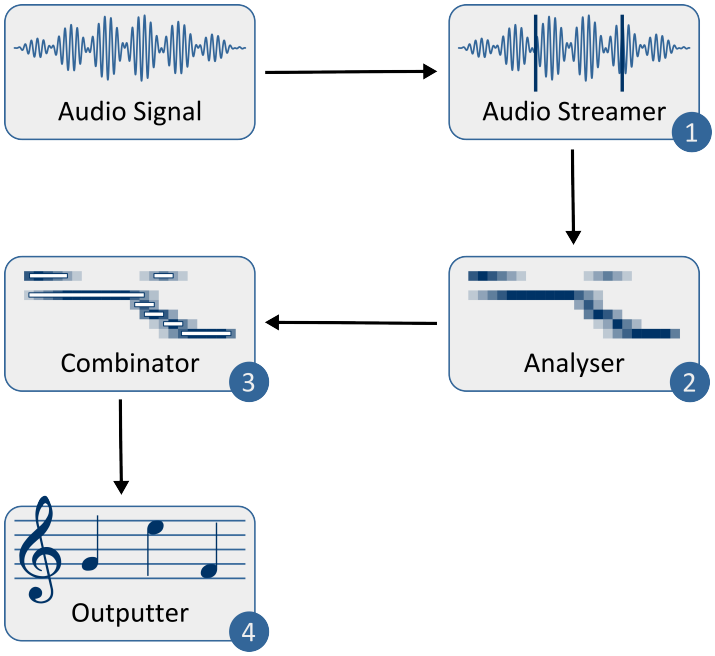


Figure 3.4: System Structure

3.4 System Structure

The system was structured in a very modular fashion, with four main components, as outlined in Figure 3.4:

1. **The audio streamer.**

   The streamer is responsible for delivering the audio signal in discrete windows of a pre-determined size. (See Section 3.4.1)

2. **The analyser.**

   The analyser converts the windowed audio signal into a series of frequency/amplitude pairs for each window. (See Section 3.4.2)

3. **The combinator.**

   The combinator examines the analyser's results, and returns a series of objects representing notes. In other words, it determines when and where notes start and end. (See Section 3.4.3)

4. **The output formatter.**

   This module converts internally-represented notes into formats usable by external applications, such as XML. (See Section 3.4.4)

The prototype follows the system structure outlined in Figure 3.4. Communication between the components is managed by a controller layer which sits above these four components. It is important to note that each module in the chain may be implemented in a variety of different ways, but that all four are necessary for the system to function. Each of these components is discussed in greater detail in Sections 3.4.1 – 3.4.4.

3.4.1 Audio Streamer

The Audio Streamer takes an audio signal, and presents it to the following component as a series of windowed data. In the prototype system, it makes use of the Java Sound API, which presents audio signals as a simple byte-stream. The streamer must convert the incoming data into a more useful format, regardless of its source. Leveraging the power of Java's Sound API, the streamer is capable of handling data in a variety of formats, both compressed and non-compressed.

The size of the windows output by the streamer has a direct impact on both the accuracy and processing requirements of the system: too large a window, and each window requires a lot of processing, and short notes cannot be detected; too small a window, and low frequencies cannot be detected. The window size can also cause quantisation problems. If, for example, a track is written at a tempo such that each "beat" is 4000 samples in length, a window size of 6000 samples will cause a loss of timing accuracy, as illustrated in Figure 3.5. In this figure, there are three notes, each 4000 samples in length; these notes are labelled X, Y, and Z. When they are analysed using a window size of 6000, they do not fit well. Indeed, notes X and Z both play for only two thirds of the window length, and Y overlaps both windows. Because the amplitude ("loudness") of the note is determined across the entire window, the note Y will be detected in both analysis windows, but only at a third of its true amplitude.

To alleviate this difficulty, the user may provide the tempo of the audio signal and the desired shortest detectable duration. From these data, the system is able to determine an appropriate window size. As always, however, there is a trade-off between the lowest detectable frequencies and the shortest detectable notes. It should also be noted that the prototype has the ability to reject notes which have a duration less than a specified number of windows. This can reduce the effect

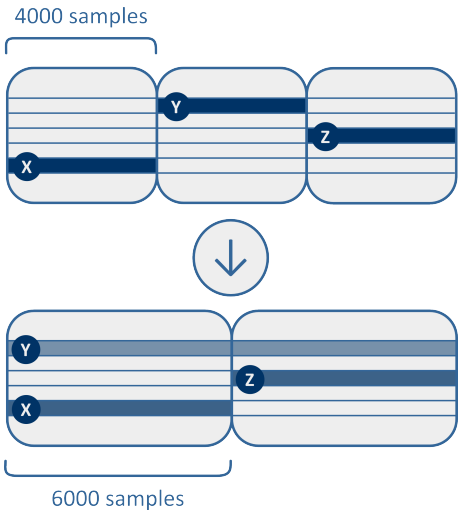


Figure 3.5: The effect of non-matching window sizes

of noise in the system, but can also lead to problems when notes are of a short duration, or are played staccato.

3.4.2 Analysis Techniques

In the prototype, two analysis techniques were fully implemented - a Short Time Discrete Fourier Transform (STDFT, or DFT) analysis and a simple sliding window DFT analysis. The sliding window analysis builds upon the DFT analysis. The underlying principles of these techniques and their implementation in the prototype are discussed in this section. Additionally, a pyramid-like technique which also extends the STFT analysis is discussed; this technique holds promise for overcoming some of the shortcomings of the other two techniques; its implementation is the subject of future work. Other techniques, such as wavelets, were not implemented, as their computational requirements are prohibitively high for mobile devices.

Discrete Fourier Transform Analysis

If we consider a signal to be the summation of many sine waves with different frequencies and amplitudes (see Figure 3.6), we can use the Fourier transform to determine the frequencies and amplitudes of these component signals. In the

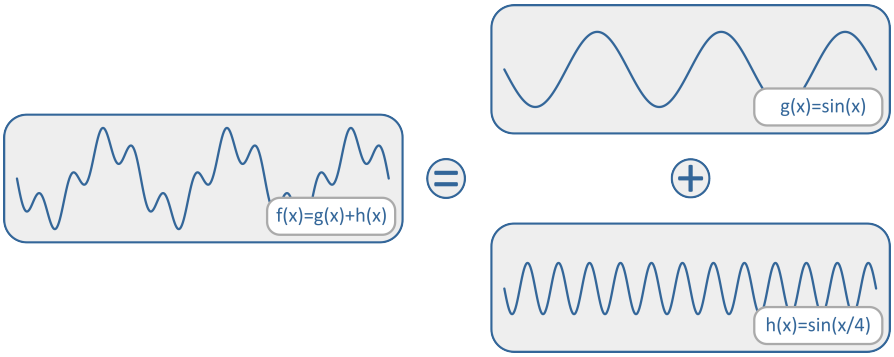


Figure 3.6: Signal composed of two sine waves

discrete case, the Fourier transform may be expressed as

$$F(\omega) = \sum_{n=0}^{N-1} x_n e^{-2i\pi\omega t_n} \tag{3.2}$$

where x is the signal. The Fourier transform returns the amplitudes of all frequencies up to the Nyquist frequency, often expressed as $f_{\text{Nyquist}} = \frac{1}{2}v$, where v is the sampling rate of the signal. This means that for an audio signal at a sampling rate of 44.1kHz (the sampling rate of Compact Discs), 22050 values will be returned. Performing a Fourier transform is very computationally expensive, however. For this reason, there exist various Fast Fourier Transform algorithms. Most of these use an approach similar to a binary search - they recursively break the problem into smaller and smaller parts.

When working with music, the ability to detect tens of thousands of distinct frequencies is superfluous; consider that most modern pianos have only eighty-eight keys, with a range extending beyond both the upper and lower registers of most other instruments. It is thus logical to reduce the number of detected frequencies for musical signals, and consequently reduce the computational requirements. This is known as a Discrete Fourier Transform (DFT). The DFT algorithm employed by the prototype minimises the number of calculations by only finding a small set of frequencies:

$$440 \times 2^{\frac{n}{12}} \quad \text{for } n = -24, \cdots, 24$$

and by using pre-calculated lookup tables for the real and imaginary parts of $\exp(-2i\pi\omega t_n)$. Given N samples in a sound signal with a sampling rate of s, these samples would occur at the time instants $t = 0, \frac{1}{s}, \frac{2}{s}, \cdots, \frac{N-1}{s}$. As all of the windows analysed will be of the same length (N is constant) and only x_n

```
// Table of lookup frequencies
private double[] freqs;
// Lookup table for real part of (exp(-i*2*pi*omega*t))
private double[] lookupReal;
// Lookup table for imaginary part
private double[] lookupImag;

public Complex[] analyseWindow(double[] windowData) {
  Complex[] F = new Complex[freqs.length];
  for (int i = 0; i < F.length; i++) {
    double[] Fr = windowData .* lookupReal[i];
    double[] Fi = windowData .* lookupImag[i];
    F[i] = new Complex(sum(Fr), sum(Fi));
  }

  // In the prototype, Complex numbers are also used as
  // frequency/amplitude pairs.
  Complex[] result = new Complex[freqs.length];
  double amplitude;
  for (int i = 0; i < F.length; i++) {
    amplitude = (sqrt(sqr(F[i].real) + sqr(F[i].imag)));
    toReturn[i] = new Complex(freqs[i], amplitude);
  }
  return toReturn;
}
```

Listing 3.1: Analysing a window using a modified DFT

changes, the expression $\exp(-2i\pi\omega t_n)$ need only be calculated once and stored in a lookup table. Calculating the transform is then simply a case of multiplying each value from the source data by the appropriate value from the lookup table, and summing the result, as outlined in Equation (3.2). Listing 3.1 outlines this technique in pseudo-code.

Significantly, using this technique allows one to calculate the Fourier transform for windows with a length not equal to a power of 2, a limitation of standard FFT implementations. This allows one to use window sizes which are more immediately related to the data being analysed. For example, a semiquaver (sixteenth note) played at 120BPM and recorded at a sampling rate of 44.1kHz lasts 0.125s, or approximately 5512 samples. The closest power of 2 is 2^{12}, i.e., 4096 samples. By using a window size which corresponds more closely with the tempo of the music being analysed, the temporal accuracy of the analysis is increased, as there is less overlap between analysed notes.

However, it should be noted that this algorithm is very limited. If the frequency lookup table does not match the input data, the results can be very in-

accurate. This occurs when the signal is out of tune, or when the tuning method used for the lookup table differs from that used by the performer of the music. The default tuning of the prototype system is known as Equal Temperament (see Section 3.2 for details on this tuning), which is used by most modern instruments. Vocalists, by contrast, tend to use Just Temperament, particularly when performing without instrumental accompaniment. Untuned percussion instruments, such as cymbals and bass drums, also cause problems for this algorithm.

When using a simple windowed DFT analysis, there is necessarily a trade-off between temporal precision and the lowest detectable frequency. This is because at least one full cycle of that frequency must be present for a frequency to be detected. In other words, if we wish to detect, for example, the note A three octaves below centre C, which has a frequency of 55Hz, we need at least 0.0182 seconds of sound. In practice, this period is generally doubled to assure accuracy, which means that 0.0364 seconds are required. At a tempo of 120BPM (Beats Per Minute), in 4/4 time, each crotchet (beat) is 0.0083 seconds in length, which is significantly less time than that required to detect the A. Clearly, this presents a problem, which may be partially offset by implementing a sliding window analysis.

Simple Sliding Window DFT Analysis

Many of the shortcomings of the simple windowed DFT analysis may be addressed by using a "Sliding Window" approach. The underlying precept of the sliding window is that by overlaying analyses, we improve temporal precision whilst maintaining the ability to detect low frequencies accurately. The drawback is that the processing requirements increase quite significantly. For this reason, the prototype's implementation slides the window by half of the window size. This effectively doubles the precision of the simple windowed DFT analysis. Figure 3.7 illustrates this visually. The first window overlaps the second, which in turn overlaps the third, and so on.

As can be seen in the pseudo-code of Listing 3.2, the sliding window analysis makes use of the simple DFT analysis. It is also necessary to rearrange data within arrays for this technique to work correctly. On devices with a significant amount of processing power, it is possible, and indeed desirable, to using a sliding window which moves only a few frames at a time. This results in both good frequency detection and very precise timing. In the context of a mobile device, however, such a fine-grained analysis is not suitable, as the computational requirements are excessive. It is for this reason, then, that the sliding window analysis implemented in the prototype slides by half a window length at a time;

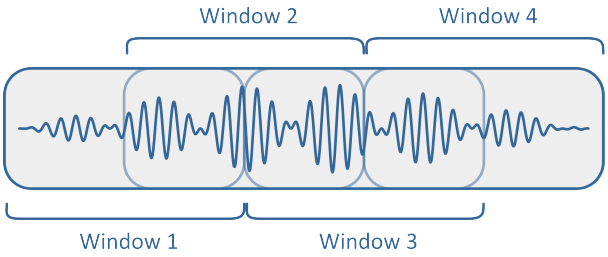


Figure 3.7: Sliding Window Analysis

```
public Complex[][] analyseWindow(double[] windowData) {
  Complex[][] result = new Complex[2][];

  // Calculate result from the second half of the previous
  // window and the first half of the current window
  double[] temp = new double[windowSize];
  for (int i = 0; i < previousWindow.length; i++) {
    temp[i] = previousWindow[i];
    temp[i + previousWindow.length] = windowData[i];
    // Update previousWindow with new data at the same time
    previousWindow[i] = windowData[i + previousWindow.length];
  }
  result[0] = fftAnalyser.analyseWindow(temp);

  // Calculate the result from the current window
  result[1] = fftAnalyser.analyseWindow(windowData);

  return result;
}
```

Listing 3.2: Sliding Window DFT Analysis

it should be noted that this then imposes the fairly trivial condition that the window size be an even number.

Pyramid DFT Analysis

A "pyramid" analysis attempts to address some of the problems of a standard windowed DFT analysis. Instead of using a single window, it uses multiple windows of different sizes to analyse the signal. Longer windows are used to detect lower frequencies, at the expense of timing precision, whilst shorter windows are used to detect higher frequencies at a high degree of temporal precision. The results are then combined to determine which frequencies are present at what times.

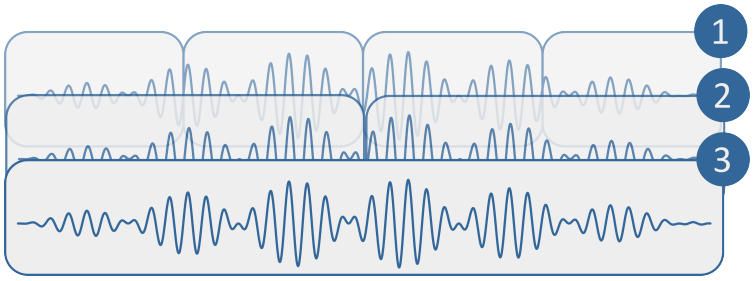


Figure 3.8: A pyramid analysis of a signal

In Figure 3.8, this concept is illustrated in a three-layered approach: the windows labelled "1" detect high frequencies, those labelled "2" detect mid-range frequencies, and the window labelled "3" detects low frequencies. Essentially, this results in a high degree of precision in detecting the timing of high-frequency notes and a more accurate detection of lower frequencies. When implementing a pyramid analysis system, it is possible to pre- or post-filter the data. Pre-filtering the data ensures that windows only receive data suitable to their length, whilst post-filtering simply ignores frequencies detected by one window that may be detected better by a shorter window. Both post- and pre-filtering require some additional processing overhead.

Pyramid analysis has not been implemented in the prototype, primarily due to time constraints. However, given the flexible nature of the system, developing an analysis model which utilises this technique is certainly feasible. It is also possible to develop a sliding pyramid analysis, similar in style to the sliding analysis outlined in section 3.4.2.

3.4.3 Combinator Methodology

The combinator is responsible for taking the raw frequency/amplitude data from the analysis, thresholding it, and determining when and where notes start and stop. It does so in three phases:

- Weighting the overtones,
- Managing notes which are still sounding ("incomplete" notes), and
- Managing notes which have finished ("complete" notes).

. The pseudo-code for this operation is provided in Listing 3.3, and provides a greater level of detail on the algorithm's implementation.

Overtone Management

Most instruments, including the human voice, do not produce pure sine waves. The fundamental note has a very strong presence, but in the entire sound, there are also "overtones". An overtone is a secondary frequency at a lower amplitude which alters the sound. Overtones tend to exist at integral multiples above the fundamental (that is, at octaves). Some also occur halfway between octaves - that is, perfect fifths. To prevent false positives arising from these overtones, it is necessary to "handicap" them, by reducing their effective amplitude within the system. It is not possible to completely remove them, as they may indeed be legitimate notes. (See lines 2 to 14 of Listing 3.3.)

Incomplete Notes

Whenever a frequency is encountered which is not currently within a list of playing frequencies, it is added to this list. (See lines 15 to 28 of Listing 3.3.)

Complete Notes

For each window, if a frequency which is listed as *playing* does not occur within the current window, it is moved into a list of "complete" notes. Notes of extremely short duration, however, such as just one window, may be discarded, at the user's discretion. (See lines 29 to 40 of Listing 3.3.)

3.4.4 Outputting Data

The notes from the combinator are finally converted into a suitable format for further use. In the prototype, the only fully-specified output format is XML, using a purpose-built Document-Type Definition (reproduced in Appendix B). The document which is output may be used in a variety of other pieces of software. For example, it is possible to use the system to take an audio signal and produce a representation in manuscript such that it may be performed by a musician.

Building on the concept of a near real-time system, with low latency, it may even be possible to use the system as a MIDI (Musical Instrument Digital Interface) controller. This would allow musicians to control virtual instruments using natural instruments; a flautist, for example, could play a synthesiser. It would also then be possible to record a musician's performance, and leverage the power of systems which already interface with MIDI controllers, and, for example, provide automatic quantisation.

```
public void processWindow(Complex[] window, int windowNum) {
  // Phase 1 - Overtone management and weighting
  for (int i = 1; i < window.length; i++) {
    if (window[i].imag > threshold) {
      for (int j = 1; j < overtones.length; j++) {
        int x = findFrequency(window[i].real * overtones[j],
                              window);
        if (x != -1) {
          window[x] = new Complex(window[x].real,
                        window[x].imag * overtonesWeight[j]);
        }
      }
    }
  }
  // Phase 2 - Manage incomplete notes
  for (int i = 0; i < window.length; i++) {
    if (window[i].imag > threshold) {
      int n = findFrequency(window[i].imag, incomplete);
      if (n != -1) {
        incomplete.get(n).addAmplitude(window[i].imag);
      } else {
        Vector<Double> v = new Vector<Double>(4, 4);
        v.add(window[i].imag);
        incomplete.add(new Note(window[i].real, v,
                                windowNum));
      }
    }
  }
  // Phase 3 - Managing complete notes
  Note tempNote;
  for (Enumeration<Note> e = incomplete.elements();
       e.hasMoreElements();) {
    tempNote = (Note) e.nextElement();
    if (!findNote(tempNote, window)) {
      incomplete.remove(tempNote);
      tempNote.setEndWindow(windowNum - 1);
      complete.add(tempNote);
    }
  }
}
```

Listing 3.3: Basic combinator function

CHAPTER 4

Experiments and Results

4.1 Overview

A prototype system developed in Java, capable of taking an audio signal and outputting XML data, will be used for these experiments. The prototype is constructed in a wholly modular fashion (see Figure 3.4). Each component may be implemented in a number of different ways. The goal of these experiments is to compare the accuracy and processing requirements of the analysis techniques implemented in the prototype. To do so, each technique should be tested under conditions which are as consistent between experiments as possible. To counteract aberrations each set of tests will be run a number of times, and the results averaged. Each test will be conducted using the same input data. These input data are constructed from five different sound sources, performing various combinations of performance style, for a total of 48 individual inputs. The general tests are outlined in Table 4.1, and the tests performed by each sound source and performance style are outlined in Table 4.2. From these tables, one can see that the piano does not perform in a fashion unnatural for that instrument; for example, it cannot perform a true glissando (slide from one note to another), and thus this is not tested.

The input data are generated manually using a purpose-written software synthesiser for the sine and saw waves, and, for the piano, using sampled instruments from the Vienna Symphonic Library (http://vsl.co.at) as available in Native Instruments Kontakt 3. An open-source piece of software, OpenMPT, will be used to control both the synthesiser and Kontakt 3, ensuring that performances vary only in the waveform, and not in tempo, pitch, or volume level. In addition to these conditions, all test data will adhere to the following specifications:

- All input data will be recorded as 16bit Mono PCM wave data with a sampling rate of 44.1kHz.
- Each test file will be normalised; that is, each file will have the same max-

	Legato	**Stac.**	**Gliss.**	**Fast**	**Slow**	**Mono.**	**Poly.**
0	•			•		•	
1	•			•			•
2	•				•	•	
3	•				•		•
4		•		•		•	
5		•		•			•
6		•			•	•	
7		•			•		•
8			•	•		•	
9			•	•			•
10			•		•	•	
11			•		•		•
12	•			•	•	•	
13	•			•	•		•
14		•		•	•	•	
15		•		•	•		•
16			•	•	•	•	
17			•	•	•		•

Table 4.1: General test cases

	Legato	**Stac.**	**Gliss.**	**Fast**	**Slow**	**Mono.**	**Poly.**
Sine	•	•	•	•	•	•	•
Saw	•	•	•	•	•	•	•
Piano	•	•		•	•	•	•

Table 4.2: Sound source tests

imum amplitude.

- All tests will be recorded at a tempo of 120BPM (Beats Per Minute).
- Tests will be run with a window size of 5514 frames; that is, the number of frames taken by a semiquaver at 120BPM.

The sine wave is used as a baseline for the results, as it provides a very clearly-defined result in a single frequency, as the sine wave has no overtones. By contrast, the saw and piano waveforms both have many overtones; they are termed "harmonically rich". It is conjectured that the results arising from harmonically rich waveforms will naturally generalise well to harmonically poor sounds, such as that produced by woodwind and brass instruments.

All tests will be conducted on the same computer, in conditions as consistent as possible. The general specifications of the computer used for testing are as follows:

- Processor: AMD 64bit 3200+ running at 2.01GHz;
- RAM: 2.00GB of DDR-2;
- Operating System: Windows XP Professional (SP-2);
- Java Runtime Environment: Java 1.6.0 (Beta)

All code was written using the NetBeans IDE (Development version), and the core functionality was validated by separately implementing it in GNU Octave and Matlab. It should be noted that the prototype has not been exhaustively tested outside this environment.

4.2 Automated Testing

To determine the accuracy of the system, it is necessary to have a base-line test. To this end, all input data must exist in three formats: the original source file, the audio signal, and a human-generated XML file based on the original source file. It is then possible to compare this XML file with the output of the system. An artifact of the various analysis techniques, however, is that they may produce equivalent results, but have different numbers of windows for each note. The sliding analysis, for example, will tend to produce results with approximately twice as many windows per note than the simple DFT analysis. To prevent this showing up as an error, it is then necessary to use the relative lengths of the

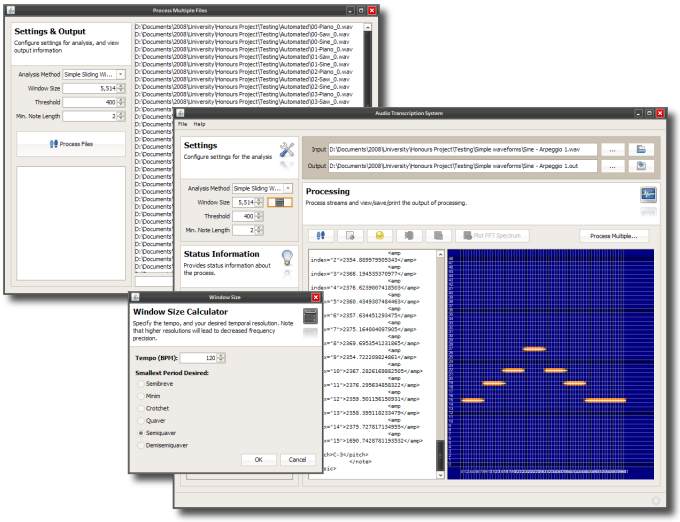


Figure 4.1: Software Screenshot

notes, rather than their absolute lengths. In other words, notes will be compared based on their proportional length within the entire analysis: a note lasting two windows in a file with twenty windows in total will have be assigned a "note length" of 0.1, for example.

4.3 Prototype Availability

The prototype desktop software produced, as well as its full source code, is available at the author's website, www.barryvan.com.au. Although this prototype should run on any system with a full Java environment installed, this cannot be guaranteed, and no documentation is provided with the prototype.

4.4 Experiment 1

The first experiment is designed to fine-tune the overtone weighting system used by the prototype. Internally, weightings are stored in two arrays, one containing the relative frequencies of the overtones, and the other the multiplicative factor to be applied to that frequency, should it be detected. (This is outlined in Listing 3.3, between lines 2 and 14.) The relative frequencies of the overtones will not be altered, but the weightings applied to those frequencies will be.

The difficulty with this experiment comes from the fact that different instruments exhibit different overtones at different amplitudes (it is this, to a large extent, that determines the tone of an instrument). This makes finding a set of weightings that work well in the general case complex. Additionally, instruments that have overtones at unusual frequencies are not accounted for; for example, the clichéd "SuperSaw" sound used in dance music is composed of two or more saw waves detuned by 0.05 to 0.50Hz, and thus cannot be appropriately managed by the system.

4.4.1 Method

Pure sine waves do not have overtones, and for this reason the sine wave data will be used as a base line for this experiment. The goal is to have the results of analysing saw and piano waves correspond as closely as possible to the sine wave data. The initial overtone weightings are presented in Table 4.3. These values will be altered independently for the saw and piano data, such that they approximate the sine data. The values will then be averaged, and, if necessary, fine-tuned by hand. The procedure used for testing will be as follows:

1. A sine wave will be analysed, to be used as a baseline. The threshold will be set at 400, the analysis will be conducted using the Simple Sliding Window analyser, and the window size will be set at 5514 samples.
2. For both of the saw and piano waveforms, the values of the weightings will be adjusted incrementally such that the result of the analysis most closely matches that of the sine wave. The same pattern of notes will be used for each.
3. The resulting weightings will then be combined together, and, if necessary, fine-tuned.

The following conditions will be used in all tests:

- The twelfth test-case will be used in each waveform; that is, a monophonic legato phrase with a mixture of fast and slow notes.
- The threshold will be set at 400.
- The analysis technique used will be the simple sliding DFT analyser outlined in Section 3.4.2.

Relative Frequency	Weighting
1.5000	0.5000
2.0000	0.7500
4.0000	0.6000
8.0000	0.5000

Table 4.3: Experiment 1 – Initial overtone weightings

- The window size used for analysis will be 5514 samples; this corresponds to the number of samples required for a quaver (eighth note) at 120BPM, the tempo of the test-cases.
- The minimum number of windows which a note must occupy to be included in the output will be set at 2.

4.4.2 Results

On the baseline tests, the initial results indicated a strong overtone presence for both the saw and piano waves at an octave above the true frequency; that is, at twice its frequency. The relatively generous weighting of 0.75 was shifted to a more conservative 0.50; this removed all overtones at this frequency in the saw waves. However, the piano continued to exhibit strong overtones at this level; over a number of trials, the optimal weighting for this overtone was found to be 0.34 to satisfy both the piano and saw waveforms. An overtone which was overlooked in the initial table was that occurring at seventeen semitones above the true frequency; that is, an octave and a perfect fifth above, or more precisely, at 2.9966 times the true frequency. Given that the overtones at eight times the value cannot be detected within the limited four-octave range of the prototype, it was replaced by the overtone occurring at 2.9966. The value ascertained as being optimal for this overtone was 0.50. The resulting overtone weightings are outlined in Table 4.4. Having determined these values, an exhaustive test was run on the full set of test data; that is, all forty-eight files. The results were, overall, very good. Examples of glissandi, however, were a notable exception to this: there tended to be significant noise around the notes. This, however, was expected, as the slides meant that little time was spent on the actual notes in the slide. It is also interesting to note that the piano analysis was generally good, however lower-frequency notes were often not detected.

Relative Frequency	Weighting
1.5000	0.5000
2.0000	0.3400
2.9966	0.5000
4.0000	0.6000

Table 4.4: Experiment 1 – Empirically-determined overtone weightings

4.4.3 Discussion

It is conjectured that the reason piano notes with a lower frequency are not detected is that the piano is built to accommodate the ear's frequency bias. That is, the ear can detect lower frequencies more easily than higher frequencies, and thus, for notes at two different pitches to sound as if they have the same amplitude, they may well have very different amplitudes. Determining the validity of this conjecture, however, would require further research, particularly into the physiology of the human ear.

Generally speaking, the updated weightings provide better results than the initial weightings. Based on the results observed, it is conjectured that these weightings will generalise well across other instruments, as the saw and piano waveforms are harmonically rich.

4.5 Experiment 2

4.5.1 Method

This second experiment is designed to determine the relative "worth" of each analysis technique implemented. It will do so by determining the computation time required to analyse each input file, the number of frames analysed in each input file, and the accuracy of the output. From these data, the average computation time required for each frame will be calculated, and the ratio of computation time per frame to accuracy will be determined. This ratio will be used to rank the analysis techniques. The procedure will be as follows:

1. A set of eleven test cases (those outlined in Table 4.1, excluding the glissando tests) will be converted exactly into XML data files from the source used to generate the sound samples; these will serve as the baseline for the testing.

Trial	Window size	Threshold	Min. note length
1	5514	400	0
2	5514	400	1
3	5514	400	2
4	5514	200	0
5	5514	200	1
6	5514	200	2

Table 4.5: Experiment 2 – Analysis settings

2. The sine wave, saw wave, and piano performances of these test cases will then be analysed five times each for windowed DFT analysis and sliding window DFT analysis (see Section 3.4.2).

3. The length of the streams (in samples) and the time taken to completely analyse them will be recorded. From these data, the average time required to analyse each sample will be calculated, and the results averaged.

4. The results of each type of analysis will be compared in terms of accuracy by determining the number of notes correctly detected, the number of notes missed, the number of notes incorrectly pitched (but correctly timed), and the number of notes correctly pitched, but incorrectly timed (with a tolerance of two windows), as compared to the baseline XML files.

5. The percentage of correctly-detected notes will be averaged across all waveforms for both analysis techniques; this is termed the "accuracy" of the analysis technique.

6. A numerical "score" will be awarded to each algorithm by dividing the achieved accuracy by the average time required to analysis each sample; a higher score is thus desirable.

7. The entire suite of tests will be conducted several times, using different settings (as outlined in Table 4.5), and the final results collated and averaged.

4.5.2 Results

The average accuracy results of the two analysis techniques are illustrated in Figure 4.2. Somewhat surprisingly, the standard DFT analysis correctly detected none of the notes; whilst it detected an average of 24.85% of the notes, these were

all picked up at the wrong time (that is, more than two window lengths distant from their correct position). By contrast, the sliding window analysis correctly identified 82.1% of the notes; all of these were exactly timed. Approximately 17.88% of the notes, however, were mispitched by the sliding window algorithm.

Table 4.6 shows the timing results of the two algorithms, and their assigned scores. It should be noted that due to its abysmal accuracy, the windowed DFT algorithm achieved a score of 0.00 on all tests. The sliding window algorithm took approximately 1.75 times as long to analyse the samples as the standard windowed algorithm.

In Figure 4.3, a selection of the graphical representations of the analyses' results are shown. These figures are taken from the third trial, and are the results of analysing the sine data using the sliding window. In these figures, the pitch is along the vertical axis, whilst time is along the horizontal. The brightness of the blocks shows the amplitude of the frequency, where a darker block is a lower amplitude and a white line indicates a note that has been detected and recorded. What is important to observe in these figures is the contrast between the detection at high and low frequencies. In Figure 4.3c, all of the notes are relatively high, and there is little noise. In Figures 4.3a and 4.3b, by contrast, the lower frequencies are surrounded by a large amount of noise. This is an artifact of the detection system; as outlined in Section 3.4.2, higher frequencies may be

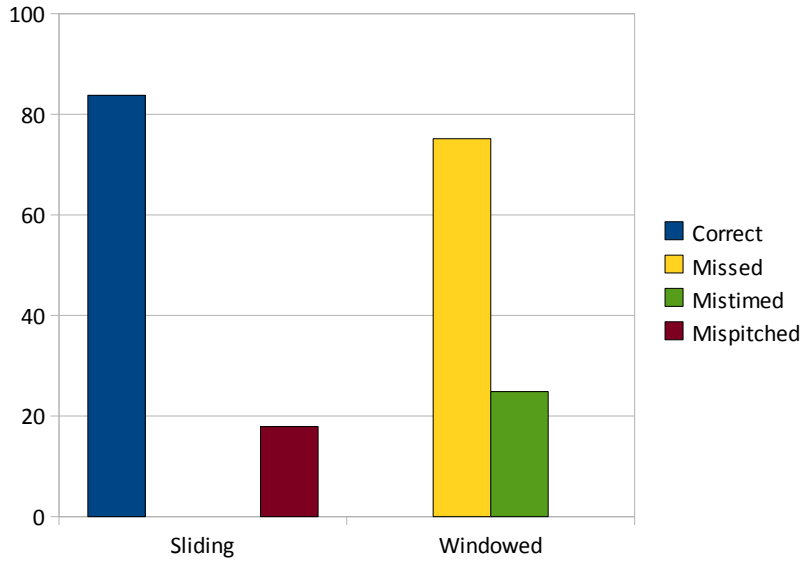
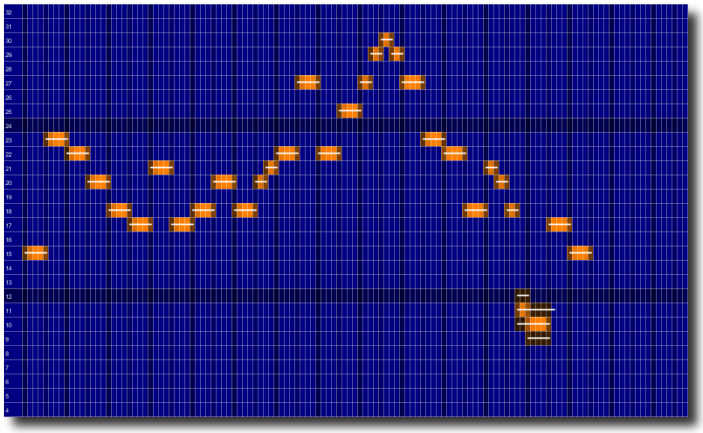
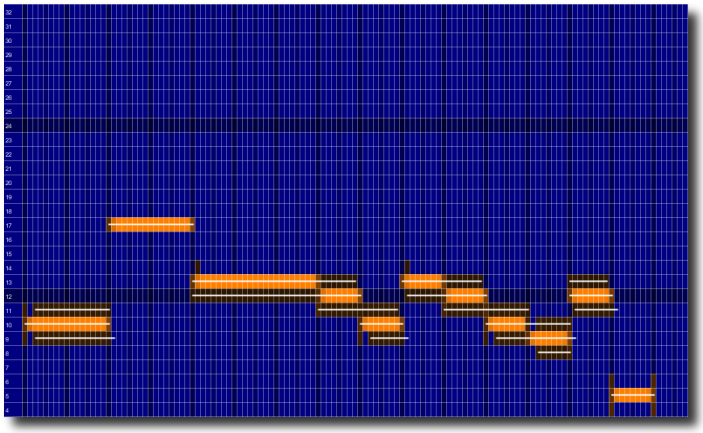


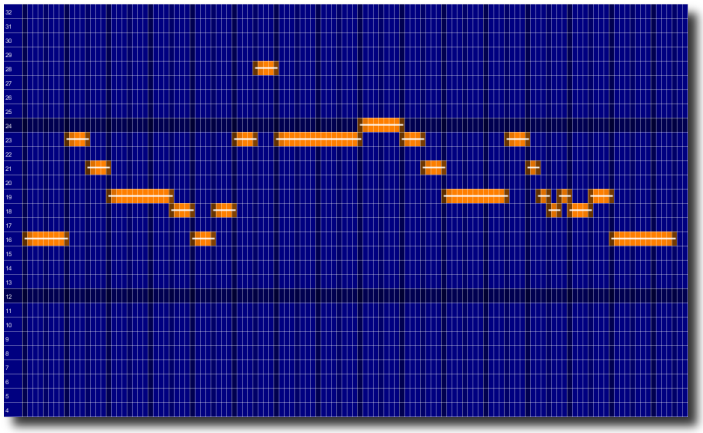
Figure 4.2: Experiment 2 – Accuracy summary



(a) Test Case 0



(b) Test Case 2



(c) Test Case 12

Figure 4.3: Experiment 2 – Graphical output

Trial	Technique	Time/sample (ms)	Accuracy (%)	Score
1	Sliding Window	4.97×10^{-3}	82.25	165.33
	Windowed DFT	2.73×10^{-3}	0.00	0.00
2	Sliding Window	5.01×10^{-3}	82.25	164.14
	Windowed DFT	2.71×10^{-3}	0.00	0.00
3	Sliding Window	5.06×10^{-3}	82.25	162.66
	Windowed DFT	2.70×10^{-3}	0.00	0.00
4	Sliding Window	$4.97 \times 106-3$	85.29	171.53
	Windowed DFT	2.79×10^{-3}	0.00	0.00
5	Sliding Window	5.03×10^{-3}	85.29	169.7
	Windowed DFT	2.70×10^{-3}	0.00	0.00
6	Sliding Window	4.55×10^{-3}	85.29	187.47
	Windowed DFT	3.14×10^{-3}	0.00	0.00
Average	Sliding Window	4.93×10^{-3}	83.77	170.14
	Windowed DFT	2.79×10^{-3}	0.00	0.00

Table 4.6: Experiment 2 – Summary of results

detected more easily than lower frequencies.

4.5.3 Discussion

The results of this experiment are somewhat surprising. Whilst it was expected that the sliding window analysis would outperform the windowed DFT analysis, the score of 0.0 in all tests by the windowed DFT analysis all tests was not anticipated. The fact that the sliding window analysis makes use of the windowed DFT analysis makes these results even more surprising. It is, however, conjectured that these results are the result of too large a window size for the analysis; a difficulty which is overcome by the use of a sliding window.

In performance, the sliding window analysis required an average of 2.79×10^{-3} milliseconds to analyse each sample in the signal. This means that a signal sampled at 44.1kHz lasting ten seconds would require approximately 1.23 seconds to analyse. This then strongly suggests that near real-time analysis using sliding windows is feasible. The accuracy of the sliding window analysis was high; as illustrated in Figure 4.2, all the timings were correctly detected, and no notes were missed, only mispitched. It may well be that providing a means of forcing the detected notes into a given scale would alleviate this problem; this, however, is a matter for future work.

Although the sliding window analysis detected all of the notes present in the signal, it also detected additional notes not present in the signal around notes of a low frequency; Figure 4.3b illustrates this very clearly. It should be noted that the majority of these false positives could be removed by raising the amplitude threshold for notes. An alternative solution would be to investigate the performance of a pyramid-based analysis scheme, which uses multiple overlapping windows of varying size to detect different frequencies; Section 3.4.2 details this method more fully.

4.6 Experimental Results

The results of the experiments performed can be summarised as follows.

1. Harmonically rich waveforms have many overtones of varying strength. It is necessary to handicap the amplitudes of these overtones so that they are not erroneously detected as notes. However, it is also necessary to be able to detect two notes playing concurrently.

2. There is a trade-off between timing resolution and the accuracy of notes detected, particularly for notes of lower frequencies. Frequencies surrounding those of the actual (low) note are generally awarded a relatively high amplitude, leading to additional, erroneous notes being detected. This problem may be partly assuaged by increasing the amplitude threshold used.

3. The use of the sliding window analysis provides very accurate results at a reasonable speed. Although some notes are mispitched using this technique, forcing them into a particular musical scale may well alleviate this difficulty.

4. Near real-time analysis using the sliding window analysis is certainly feasible on the test system, and therefore, it is conjectured, on mobile devices.

CHAPTER 5

Conclusion

The results of this project work clearly indicate that a system for the near real-time transcription of music, which is capable of running on mobile devices, is feasible. The prototype which was developed is already a useful tool for musicians in and of itself, as it is capable of transcribing music to a very high degree of accuracy when using the sliding window analysis; the results of the experiment in Section 4.5 indicate this. It was hypothesised that the windowed Fourier transform-based analysis technique would not produce results to the same level of accuracy as the more complex sliding window technique, and this hypothesis was borne out in the experimental results. Indeed, the windowed analysis failed to correctly detect any notes, whilst the sliding window analysis was able to achieve an average accuracy of 83.77%.

The constraint that the system be capable of running in near real-time is one of the main reasons for avoiding the use of computationally expensive analysis techniques, such as wavelets. Experimental results indicate that whilst the sliding window analysis took approximately 1.75 times as long to complete analysis on the tests, this equated to 1.23 seconds of analysis time for every ten seconds of audio (at 44.1kHz, CD quality). Whilst the test system is admittedly significantly more powerful than the majority of mobile devices, this is a very promising result, particularly given the rapidity of computation power increases in such devices. The project thus successfully demonstrated that the system developed in the prototype could be ported successfully to a mobile device.

5.1 Future Work

The scope for future work in this area is very large. Having ascertained that it is indeed feasible to develop a transcription system for mobile devices, such a system could indeed be developed. This could be developed in Java, and so use much of the existing code. It could include a variety of features designed to facilitate the use of the system for vocal transcription, such as a metronome, a “drone” pitch,

which would help to keep singers on-pitch, a built-in editor to quantise, move, and delete notes, and much more besides. It would also be possible to implement a system which would constrain the notes to a given scale, such as C major; this would reduce error incidences, at the expense of musical complexity. Harmonic analysis is another possibility; that is, the recognition of chords. This would be particularly useful for pop and jazz pieces, which are based very heavily on chord progressions.

Continuing to develop this system for computers, a greater emphasis could be placed on accuracy, and much more computationally intensive methods for pitch detection could be explored, such as wavelets. As processing power continues to increase, this could, in the near future, result in an extremely accurate real-time transcription system. Eventually, such a system could replace interfaces such as MIDI (Musical Instrument Digital Interface), which has been around since the 1980s, for most musical tasks. Instead of plugging a keyboard into their computer, musicians could simply perform to a microphone; this would be a far more cost-effective method of entering music in a traditionally "musical" fashion. Additionally, improvisatory passages which have been recorded live could be transcribed, and used as teaching aids by musical tutors.

APPENDIX A

Original Honours Proposal

A.1 Background

Music is a uniquely human phenomenon. In 1871, Darwin [3] observed that man's musical abilities "must be ranked amongst the most mysterious with which he is endowed". Yet only a small minority has the ability to not only play, but also to create music. Even now, the works of famous composers, such as Vivaldi, Rachmaninoff, and Beethoven are still enjoyed across the world.

In 1985, computer music became a market-centred enterprise, and, with the accompanying reduction in the price of producing music, there was an increase in the global availability and output of music [8]. Through all of this, however, the act of writing music has remained largely unchanged. New systems and techniques, such as sequencers, trackers, and computer-aided notation systems, despite their differences, are all alike in one aspect: they rely on the composer to manually enter each note with pen, mouse, or keyboard. Although it is also possible to use specially modified or constructed instruments for inputting these data, there are very few solutions for the simplest, most accessible, and arguably the easiest instrument of all: the human voice. Existing solutions tend to exhibit two significant defects:

1. They operate only on pre-recorded sounds; and
2. They are not portable.

A system which was capable of transcribing a melody as it is sung, and which is portable – operating, perhaps, on a mobile telephone – would be of use to people of all levels of musical ability. When a musical fragment suggests itself, the composer would simply be able to sing it to their telephone for later use. Mobile telephones present a logical choice for this system, as they support a Java environment, are increasing rapidly in processing power, and have the requisite hardware.

A.2 Aim

This project's aim is to develop a real-time system for music transcription of the human voice. Numerous techniques for pitch-recognition will be investigated, including:

1. Using Fast Fourier Transforms (FFTs) on discrete "windows" of audio; for example, analysing the audio from 0–50ms, then from 50–100ms, and so on.

2. Using FFTs under a sliding-window system; for example, analysing the audio from 0–50ms, then from 10–60ms, and so on, averaging the 50ms blocks to obtain the results.

3. Using frequency intensity-based pattern matching. This would be based on the presence of overtones in the audio; for example, a very intense frequency of 440Hz sounding concurrently with a weaker frequency of 880Hz might suggest that an *A* is being played [1].

Each of these techniques will produce as output what is known as "sonogram" or "voiceprint", which allows the relative intensities of frequencies to be easily determined over time [7]. A prototype system using each of these techniques will be developed to facilitate experimentation; the efficiency and accuracy of the methods will be compared. Using these data, a system capable of running on a mobile phone with a reasonable accuracy and efficiency will be proposed, and, if time permits, developed.

The prototypes will be written in Java or C/C++. They will feature real-time visual feedback of the transcription process, and will store their data in an XML-based format. If time permits, alternate tuning mechanisms (including the use of quarter tones) will be also be considered.

A.3 Method

The project will comprise the following tasks:

1. **Background Research**

   (a) Research into pitch analysis. Initially, the focus will be on monophonic pitch analysis, as this is the simplest case. If time permits, this may be extended to include polyphonic pitch analysis as well. The goal is

to build up sufficient data to enable each of the three pitch recognition algorithms outlined above to be developed.

(b) Research into automated transcription systems and their typical algorithms, such as the non-sliding and sliding window FFT-based systems outlined in the Aim Section above. Existing systems, such as Transcribe! [13] and TS-AudioToMIDI [15] will be investigated.

This research will include methods for transforming the raw data of the the pitch analyses into a usable, XML-based format. It may also be desirable to display the XML in an easy-to-understand format; perhaps a piano-roll as described by Gaare [4].

(c) Research into variables associated specifically with the human voice (vibrato, tremolo, etc.), and their impact on pitch analysis and transcription.

2. **System Implementation and Experimentation**

(a) A framework for prototyping the three pitch-recognition algorithms outlined in the "Aim" Section; that is,

i. Using Fast Fourier Transforms (FFTs) on discrete windows of audio;

ii. Using FFTs under a sliding-window system; and

iii. Using frequency intensity-based pattern matching.

This framework will provide several facilities used by each algorithm, including a well-defined interface for the algorithms; the presentation of the audio stream; a standardised output format, which may be converted into XML; and, eventually, display routines for the data.

(b) A prototype for each pitch-recognition algorithm, running on a computer.

(c) Accuracy and efficiency tests for each algorithm.

These will be performed using a variety of standardised testing data; the "real-time" aspects of the system will be simulated by streaming the data into the system. Metrics for comparing algorithms will include CPU usage, accuracy (compared to a human-notated standard), and the ability to handle errors.

3. **Extension Work**

(a) Based on the experimental data, a prototype suitable for a mobile telephone.

(b) Polyphonic pitch detection and notation: for example, multiple singers or an ensemble of instrumentalists.

(c) Instrumental notation; that is, the ability to notate a violin or recorder, as well as the human voice.

(d) The inclusion of a metronome, to assist the singer's timing; this will facilitate quantisation.

(e) The inclusion of a "base-note" sound, to assist the singer's pitching. That is, a "drone" pitch. If the singer is singing in the key of *A Major*, a single *A* note may be played,

A.4 Software and Hardware Requirements

The project has the following requirements:

1. **Hardware**

   (a) A computer with a soundcard;

   (b) A microphone (with a 3.5mm jack compatible with the computer's soundcard);

   (c) A set of headphones and/or speakers (with a 3.5mm jack compatible with the computer's soundcard);

   (d) A late-model Sony Ericsson mobile telephone with a cable for transfer of data with the computer.

2. **Software**

   (a) The Java Compiler and/or GCC;

   (b) Assorted Java and/or C/C++ packages for frequency analysis in streamed audio;

   (c) Sony Ericsson JDK and related documentation (for future work).

APPENDIX B

XML Output Document-Type Definition

The Document-Type Definition (DTD) used for the XML files output by the system is reproduced here. This DTD was constructed as a way of minimally capturing all data pertinent to the audio signal.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT music (header, note*)>

<!ELEMENT header (timestamp, windowSize?,
                  samplingRate?, threshold?,
                  analysisMethod?, inputFilename?)>
<!ELEMENT timestamp (#PCDATA)>
<!ELEMENT windowSize (#PCDATA)>
<!ELEMENT samplingRate (#PCDATA)>
<!ELEMENT threshold (#PCDATA)>
<!ELEMENT analysisMethod (#PCDATA)>
<!ELEMENT inputFilename (#PCDATA)>

<!ELEMENT note (start, end, freq, pitch?, amp+)>
<!ELEMENT start (#PCDATA)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT freq (#PCDATA)>
<!ELEMENT pitch (#PCDATA)>
<!ELEMENT amp (#PCDATA)>
<!ATTLIST amp index CDATA #REQUIRED>
```

Listing B.1: Output XML DTD

Bibliography

[1] Alm, J. F., and Walker, J. S. Time-frequency analysis of musical instruments. *SIAM Review 44* (Sept. 2002), 457–476.

[2] Bosanquet, R. H. M. Temperament; or, the division of the octave [part i]. *Proceedings of the Musical Association 1* (1874), 4–17.

[3] Darwin, C. *The descent of man and selection in relation to sex.* John Murray, Albermarle Street, 1871.

[4] Gaare, M. Alternatives to traditional notation. *Music Educators Journal 83* (Mar. 1997), 17–23.

[5] Gómez, E., Streich, S., Ong, B., Paiva, R. P., Tappert, S., Batke, J.-M., Poliner, G., and Ellis, D. A quantitative comparison of different approaches for melody extraction from polyphonic audio recordings. Technical report, Universitat Pompeu Fabra, Apr. 2006.

[6] Klapuri, A., and Vnnen, M. Automatic transcription of music. Tech. rep., in Proceedings Stockholm Music Acoustics Conference, 1998.

[7] Lepain, P. Polyphonic pitch extraction from musical signals. *Journal of New Music Research 28* (1999), 296–309.

[8] Moore, F. R. Dreams of computer music: Then and now. *Computer Music Journal 20* (1996), 25–41.

[9] Nikolova, I., Davey, L., and Dean, G., Eds. *The Illustrated Encyclopedia of Musical Instruments.* Könemann Verlagsgesellschaft mbH., 2000.

[10] Nolan, C. *The Cambridge History of Western Music Theory.* Cambridge University Press, 2002, ch. Music Theory and Mathematics, pp. 272–304.

[11] Paiva, R. P., Mendes, T., and Cardoso, A. Melody detection in polyphonic musical signals: Exploiting perceptual rules, note salience, and melodic smoothness. *Computer Music Journal 30* (2006), 80–98.

[12] Recordare LLC. MusicXML definition. Tech. rep., Recordare LLC., 2008. Available at http://www.recordare.com/xml.html.

[13] Seventh String Software. Transcribe! Website, 2008. Available at http://www.seventhstring.com/xscribe/overview.html.

[14] Steedman, M. The well-tempered computer. *Philosophical Transactions: Physical Sciences and Engineering 349* (Oct. 1994), 115–131.

[15] Tallstick Sound Project. Ts-audiotomidi. Website, 2004. Available at http://audioto.com/.

[16] Wolf, D. J. Alternative tunings, alternative tonalities. *Contemporary Music Review 22* (Jan. 2003), 3–14.