Digital audio analysis and processing for a rock-based musical instrument

Zaq Williams

May 19, 2010
# Contents

1 Introduction ................................................. 1
   1.1 The problem .............................................. 1
   1.2 Relevance to degree programme ......................... 1
   1.3 Minimum requirements ................................... 1
   1.4 Deliverables ............................................. 2
   1.5 Possible Extensions ..................................... 2
   1.6 Methodology ............................................ 3
   1.7 Schedule ............................................... 3

2 Background Research ........................................... 4
   2.1 FFTs and spectral analysis .............................. 4
   2.2 Pitch detection ........................................... 4
      2.2.1 Monophonic ......................................... 4
      2.2.2 Polyphonic ......................................... 6
   2.3 Appropriate technologies ............................... 7

3 Implementation ............................................... 8
   3.1 System Design ........................................... 8
   3.2 Fundamental Frequencies And Their Overtones .......... 9
   3.3 Extensions .............................................. 10

4 Evaluation .................................................... 11
   4.1 Results .................................................. 11
   4.2 Evaluation of the Project ................................ 11
   4.3 Possibilities for Further Research ....................... 12
   4.4 Conclusions ............................................. 12

A Reflection on the Project .................................... 15

B System Code .................................................. 16
   B.1 Matlab System ............................................ 16
   B.2 C Offline System ........................................ 20
   B.3 C Real-Time System (Incomplete) ......................... 25
Abstract

The aim of this project is to investigate the feasibility of the usage of pitch detection on a rock-based musical instrument developed by Leeds University Interdisciplinary Centre for Scientific Research in Music (ICSRiM) in order to ascertain the notes played. In order to decide the best approach to the problem, research into previous studies and existing pitch detection methods was conducted and analysed. A semi-polyphonic system capable of detecting notes in prerecorded samples was developed, along with the beginnings of a system for real-time analysis. The system was then evaluated using a number of sound files and a ground truth set to assess its accuracy.
Chapter 1

Introduction

1.1 The problem

Pitch detection is still a very active research field, with a wide range of proposed methodologies competing for the best solution to an attractive problem. A reliable pitch detection algorithm for human voice is a key component of many speech recognition methods[7], the development of which has many business and personal applications including automatic transcription of phone calls and the automated generation of subtitles for video media, as well as the ability to isolate one speaker in a noise. Pitch detection research has also garnered a lot of attention in the music world, the ability for a computer to accurately and consistently detect the notes present in a piece of music and when they are played would facilitate automatic transcription of music and other interesting applications such as the ability to separate a track into vocals and individual instruments or manipulation of the original sound in new intriguing ways.

Despite the large amount of research conducted into this area, it is still far from producing a solution capable of achieving a suitable accuracy for many of the applications it is best suited for. This is especially true for polyphonic pitch detection, the best Multiple Fundamental Frequency solution achieving an accuracy measure of 0.688 in the Mirex 2009 competition[13]. It is for these reasons that this project is a study into the feasibility of an accurate pitch detection method for the xylophone-like instrument being developed by ICSRiM being put into use, as if such a method can be developed then it opens up more possible uses and developments of the instrument.

1.2 Relevance to degree programme

This project utilises the skills learned from multiple School of Computing modules. The Software Systems Engineering and Computational Modelling provided invaluable experience in project management and algorithm design whilst Computer Vision, Parallel Scientific Computation and Artificial Intelligence familiarised me with some of the mathematics used within my project.

1.3 Minimum requirements

Below is a list of the minimum requirements for the project, listed and extrapolated on in order of expected implementation:

- **The production of an algorithm prototype for audio analysis: i) FFT to detect fundamental frequencies present in the signal.**
  
  The purpose of this algorithm is to use a Fast Fourier Transform to analyse a segment of audio signal and attempt to detect its fundamental frequency (F0), i.e. the lowest frequency present in the signal that adds energy to it.

- **The production of an algorithm prototype for audio analysis: ii) Detect the overtones associated with the fundamental frequency determined in i).**
The algorithm should attempt to detect the overtones/higher frequency components (F1..Fn) present that also add energy to the signal.

- **The production of an algorithm prototype for audio analysis:** iii) Measure the variations (decay) of the fundamental frequencies and overtones over time.

As a simple note progresses, there are typically two portions: the attack and the decay of the note. The attack refers to the initial burst of energy in the signal as, for example, a xylophone key is struck. The decay refers to the latter portion of the signal, as the note rings and fades over time. The algorithm should be able to monitor the decay of the F0 and overtones over time.

- **The production of a pitch detection algorithm for the rock instrument.**

For a small monophonic sound signal, pitch detection or fundamental frequency estimation using FFTs is relatively simple, but for longer signals where the pitch changes over time or multiple pitches are present at the same time it is much more difficult. The pitch detection algorithm should be able to detect pitch in the possible sounds produced by the rock instrument over a reasonable period of time.

### 1.4 Deliverables

- An algorithm that performs a FFT on an input signal taken from the rock instrument and outputs the estimated fundamental frequencies present.

- An algorithm which outputs the overtones present in a signal sample when given the fourier transform and the detected fundamental frequencies from the above algorithm.

- A pitch detection algorithm which accepts as input a signal represented in the time/frequency domain (to be decided based on research) and outputs predicted pitches present in the signal.

- An evaluation of the above algorithms detailing their accuracy and effectiveness.

- A project report to be submitted for the May 11th deadline.

### 1.5 Possible Extensions

There are a number of possible extensions to this project. One interesting area to investigate is whether an audio augmentation module could be created which detects the steady/sustain state of the sound and creates a longer sustain simulation, possibly purifying the input audio with selected overtones. As rocks do not usually have a lengthy sustain, implementing this would allow for the creation of a more pleasing or musical sound. Expanding upon this, integrating the developed modules into a software prototype which accepts input from a small xylophone-like rock instrument (provided by ICSRiM) and outputs a new augmented sound would adapt the product to be able to process sounds which are not pre-recorded.

Furthermore if the product were to be adapted to perform in real-time, taking the audio input from the instrument and outputting an augmented sound with such a short delay as to be unnoticeable, it would allow for the device to be played as an instrument. There are a number of factors which could affect the feasibility of this extension: hardware specifications will undoubtedly affect the speed at which the software can process audio, the efficiency and accuracy (possibly a trade off) of the pitch recognition method used may influence the computational complexity, and the amount of processing of the input signal to create the augmented output signal will also increase the computational load.

Another extension possibility is the production of a graphical user interface for the software, increasing its usability and possibly offering simple methods of controlling some of the parameters in the system such as note decay, overtone influence and timbre. Building on this, it may be of
interest to extend the software into multimedia, producing a visual representation of the sound output from the instrument system.

1.6 Methodology

The most suitable approach to tackling this project I believe is an iterative design methodology, first performing background research and planning how the project will be scheduled and determining the minimum requirements and then implementing and refining each minimum requirement turn before moving on to the next requirement. Once all requirements are met the project will enter an evaluation stage before being reviewed and revised. After a number of iterations of this method the system will enter a final evaluation and be finalised before the project report is complete. This has the advantage of setting clear checkpoints and deadlines in development and assigning project time to revisiting areas of the system to make any improvements.

1.7 Schedule

This is the original schedule for the project.

- **Background research 1/1/2010 - 15/2/2010 (6 weeks)**
  The initial stages of the project require gaining some background knowledge of the problem so that a suitable approach and methodology can be taken, for this reason a significant portion of the project will be dedicated to background research.

- **Implement Minimum Requirements 16/2/2010 - 17/3/2010 (4 weeks)**
  Implementation of the minimum requirements should be undertaken as soon as background research has been completed, to allow for the necessary time for evaluation and extension portions of the project to be fully conducted.

- **Evaluate Implementation 18/3/2010 - 19/3/2010 (2 days)**
  It is important that an initial evaluation of the minimum requirements is performed before further developments are made in order to assess the current quality of the product as well as assess the need for any refinements or bug fixes before continuing.

  The extension areas discussed provide a varied choice of routes to take with the further development of the project, but which to take will most likely be affected by which can be realistically completed within the given schedule.

  Once a prototype of the final product has been developed, it must again be evaluated to assess its proficiency and to test for any bugs. These will be fixed or altered accordingly and the final product will be left so that a final report can be made.

- **Write Project Report 16/2/2010 - 12/5/2010 (10 weeks)**
  The project report is the most important aspect of the project and as such must undergo a suitably long writing and drafting process. A mid-project report must be written first, followed by a full report to be handed in by the 12/5/2010 deadline.
Chapter 2

Background Research

2.1 FFTs and spectral analysis

The first three minimum requirements require analysis of Fast Fourier transforms (FFTs), so I began my research by reading papers on them. Fast Fourier Transforms\[3\] are a relatively efficient way of computing the Fourier series of a signal without having to perform a full discrete fourier transform. They reduce an operation bounded in $O(n^2)$ to one in $O(n \log n)$. There are a number of different FFT methods, one of the most popular of which being the radix-2 method. This method calculates the discrete fourier series of an array size N by separating the calculation of the equation $X(k) = \sum_{j=0}^{N-1} A(k) \cdot W^j$ into the addition of two smaller fourier transforms performed separately on the even and odd indices values of A, giving $X(k) = \sum_{m=0}^{N/2-1} x_{2m} W^j(2m) + \sum_{m=0}^{N/2-1} x_{2m+1} W^j(2m+1)$.

If the size of N is restricted to $2^n$, then this equation can be used recursively on all sub-transforms until a base case of $N = 2$ is reached. As of yet this doesn’t save much in computational cost, however because of the symmetric nature of the Fourier series we can reduce the computational cost greatly by reusing the calculated values of A and W for the first half of the Fourier series when computing the second half\[20\).

The main limitation of this method is that although it improves greatly on the discrete fourier transform, it requires that N be of size $2^n$. Other radix methods such as the radix-4 method suffer similar restrictions. There are a number of other FFT algorithms that have been developed which improve upon the savings in complex multiplications and additions achieved by Cooley and Tukey, such as one which uses lex complex numbers\[18\], the split-radix\[6\] and modified split-radix\[8\] FFT.

2.2 Pitch detection

The fourth minimum requirement relates directly to automated pitch detection which is still an ongoing field of research with many branches of study. The attraction of a system which is automatically able to detect the tone of a noise is high, with applications such as human speech recognition and understanding, automated transcription of music and isolation of voices in a signal. Methods are primarily split into monophonic and polyphonic pitch detection, and as such they will be discussed seperately in this chapter.

2.2.1 Monophonic

Monophonic pitch detection refers to sound signals which only have one voice or note present in the signal at a time for example trumpets and human speech. The computer recognition of human speech has been of interest for some time, so there are numerous studies into monophonic pitch recognition which have produced some interesting methods. One of the most basic of these is analysis in the frequency domain uses FFTs to translate a signal from the time to frequency domain. The immediate benefit of this is that pitch is directly represented in this domain and by analysing the fourier transform of an input signal it is possible to locate the lowest maxima above
a certain amplitude threshold and as such determine the fundamental frequency. A drawback of this method is that human perception of pitch tends to scale logarithmically, so in order to cover the full range of octaves that may be present in a signal a large transform may be needed[10].

Another method, known as 'Autocorrelation'[12] operates in the time domain and takes advantage of the similarity between two adjacent periods of a signal to attempt to find the fundamental frequency. By shifting a signal across a time window and using an autocorrelation function and a measure of similarity such as pointwise absolute difference, a plot of similarity against time step can be created. The lowest troughs in this plot indicate where the samples best match, and where the fundamental period is. From this the fundamental frequency of the signal can be obtained.

![Figure 2.1: The blue signal is the original, and the green represents a copy shifted by an amount s. Taken from [12]](image)

There are a number of limitations to the Autocorrelation method however. As it uses a shift in samples in its calculation of fundamental period, it is highly sensitive to sampling rate[12]. Ambiguities can form from this, and for higher pitch signals the accuracy of this method has been found to decrease significantly. It is also computationally expensive to perform, and still has difficulty performing in real-time using a fast-autocorrelation procedure. All of these faults become an issue when attempting to apply the method to music, as discussed by Kuhn[10].

An improvement upon the Autocorrelation method was developed by McLeod and Wyvill[11] and uses the Autocorrelation function combined with a square difference function to create a normalised square difference function (NSDF) which produces a correlation measure ranging from 1 (perfect positive correlation) to -1 (perfect negative correlation). By searching for local maxima beginning at the first positively sloped 0-crossing to avoid the correlation at time step 0, the algorithm is able to obtain a list of possible fundamental frequency values. The accuracy of this is increased using parabolic interpolation and only selecting the highest maximum at each positively sloped 0 crossing. This leaves a small number of candidates for the maximum, the first of which corresponds to the period.

The advantage of this method over standard autocorrelation is that it is less affected by spectral leakage, i.e. signal energy appearing to affect unrelated frequencies. The cause of this in Fourier transforms and autocorrelation is the cutoff point of the signal sample not containing a whole number of periods of the signal. One solution to this is to use smoothing and a windowing function, however the normalisation of the McLeod method reduces the effect of leakage
substantially. Other techniques include:

**Fundamental Period Measurement**[10] This method uses a cascade of bandpass filters to separate the signal into several filtered signals, with a near sinusoidal representation of the fundamental frequency being represented in the lowest filter output which does not contain pure noise or silence. By scanning the filtered signals from lowest to highest and selecting the first signal that appears above a certain threshold amplitude the fundamental frequency can be obtained. The primary advantage of this method is that it is easily implemented in hardware giving a large increase in efficiency, but the accuracy is not as high as other methods.

**F0 determination from precise partial estimates**[14] By first converting the signal into the frequency domain and then using a procedure to detect and estimate the partials present in the signal, it can be assumed that the strongest partial detected is present in the main harmonic series and as such is a multiple of the fundamental frequency. By creating a set of candidate fundamental frequencies from this and assigning the detected partials to the best-match fundamental frequency candidate, the best candidate surfaces and is taken as the estimate for the fundamental frequency.

**YIN fundamental frequency estimator**[5] The YIN method expands on the autocorrelation method by including a number of algorithms which reduce the error, producing a final algorithm with a significantly lower gross error measure.

### 2.2.2 Polyphonic

The rock instrument which the sound samples used in the project will come from is xylophone like in design, and as such is capable of creating sounds which have new pitches appearing in the signal as the previous decays, or even two or more notes being struck simultaneously. This behaviour in a is known as polyphonic and refers to the presence of multiple pitches or tones present within a signal. While the human voice is monophonic, a piano or guitar can play multiple notes at once (chords) or strike a new note while a previous note is still playing and as such are polyphonic instruments. Monophonic pitch detection and tracking has received a large amount of attention in the last few decades and has gradually improved to a point where there are accurate methods available, polyphonic pitch detection has received much less attention. Because of this the relative accuracy of polyphonic methods do not match those of monophonic, and the field includes a very wide range of techniques with differing degrees of computational cost and accuracy. There are some interesting methods that make use of multimedia in pitch estimation, for example using computer vision to detect where fingers are located on a fretboard and reduce the range of possible frequencies present[17]. These are however outside the scope of the project, and the following discussion will relate to audio-only methods.

Robertson and Plumly[19] developed a technique which used the knowledge that a given pitch will demonstrate peaks at both the fundamental frequency and its partials to track multiple pitches. Their algorithm took the output from a Max/MSP object called *fiddle* which contained the top frequencies detected in the signal sample and worked through these determining whether they represented a note-on (a new note) or a note-off (the ending of a note). They then subtracted higher partials of estimated playing notes from the signal, making determination of other pitches and fundamental frequencies present in the signal easier. Using the de facto standard MIREX evaluation the paper claims an accuracy of over 50%, but their own evaluation on more limited data sets show both a higher percentage of true-positive results and false-positive results.

A similar method estimates the fundamental frequency of the most prominent sound, then uses spectral smoothing during the subtraction of this frequency and its partials to accurately remove it from the signal sample before performing the same procedure again. The algorithm works iteratively until a break-off point and offers some promising error rates ranging from 1.4% to 14% depending on the number of notes present[9].

There are a number of methods that build on the *fiddle* program. One such method differs from Robertson and Plumlys in that it uses a genetic algorithm and a k-nearest neighbour (K-
NN) approach instead. Although it only attempts to recognise the type of instrument playing and not the pitch of the notes, the architecture of the system is still interesting. By running the genetic algorithm offline (i.e. separate from classification) on a number of sample sounds and obtaining a weight vector, a K-NN classifier can be executed on a window of live mic input to determine which instrument the sound input is most like.

Cont[2] also used a machine learning approach, using non-negative matrix factorisation (NMF) and a matrix consisting of a number of learned pitch templates to predict which of these templates are active in the ongoing signal. With a suitable number of pitch templates for a specific instrument it may be possible to obtain a high degree of accuracy with this method, however their evaluation measure appears to be a positive detection during 80% of the note life which could result in note-on events appearing too soon or too late in the predicted output. It does however offer a Max/MSP object which is claims is able to perform in real-time. Bertin et al.[1] wrote a comparison paper evaluating the NMF and K-singular value decomposition (K-SVD) methods, and found that for their evaluation measures NMF achieved on average 33.4% accuracy while K-SVD achieved 27.2% accuracy for real audio. They also found that the accuracy of both methods decreased from 60% to 40% the more pitches were present in the signal.

There are also studies which make use of Markov models and the structured nature of music[21][16] to aid in polyphonic pitch detection, although these require time-consuming training stages and are beyond the scope of this project.

2.3 Appropriate technologies

There are a number of programming languages and development environments that offer libraries for audio and fast fourier transforms. Some of the most prominent of these are:

- Java
- Python
- C
- Matlab
- Max/MSP

Java and Python are both popular high-level languages with extensive audio libraries available for them such as Sonia for Java, offering realtime FFT analysis and reading/writing of Wave files, and PyAudio for Python (a binding to PortAudio). These languages are often reliable and as I have more experience in them than others development would progress at a quicker pace. They are however known to be less efficient than C and C++ for low-level computation and as the project relates to audio input and processing may not be the best choice.

I found that Portaudio and Libsndfile for C were easy to use, had favourable reviews and were cross-platform - something that is useful to me as I have yet to confirm what platform the final product will be expected to run on. C has the added advantage of being a relatively fast language, which may be of concern if I attempt to implement the extension making the software perform in real-time. Matlab is notoriously slow in comparison to C, but its scripting and interpreter features allow for quick experimentation and prototyping which is useful in the beginning stages of a project.

Max/MSP is a software package consisting of a visual programming language and GUI extensions for music and multimedia development. As hinted at previously in the chapter when discussing the fiddle object usable in Max/MSP, it is a popular application amongst researchers and composers. It has a large userbase and numerous addons which are available for download and use, and as such it may be a useful platform for development of the project.
Chapter 3

Implementation

3.1 System Design

From the background research conducted it is apparent that that for instrument the project is focused on a polyphonic pitch detection method would be most appropriate. As a musician may strike multiple keys at once, or strike a key while a previous is still ringing, it is useful for the algorithm to be able to separate different pitches within the signal. I chose to implement a system similar in design to the signal cancellation approach of Klapuri[9] as this approach appears to achieve good results and the iterative approach to pitch detection appears less time-consuming to implement than other methods such as Non-negative Matrix Factorisation[2]. A model of the flow of the system is shown below:

The proposed system would accept as input an audio sample taken from the rock instrument, perform preprocessing and noise reduction on it and then perform a Fast Fourier Transform. This transform would then be used by a peak detection algorithm to detect the largest frequency components of the audio sample, which could then be used to find the fundamental frequency (F0). A ratio calculation algorithm would then find the overtones associated with F0 before a pitch decision is made and these associated peaks are removed from the input. The remaining input can then be reprocessed by the algorithm finding a new F0 and associated overtones, and the process continue until all notes have been found. The system would initially be designed in Matlab because of the advantages of a simple scripting language and visual plotting functions which will aid in rapid development and prototyping.
3.2 Fundamental Frequencies And Their Overtones

The first three minimum requirements of the project; FFT to detect fundamental frequencies present in the signal, detect the overtones associated with F0 determined and measuring the variations (decay) of the fundamental frequencies and overtones over time are all closely related. The first of which requires that a Fourier series is produced for an input signal from which the various frequency components and their amplitudes can be analysed. An algorithm must then decide the F0 from the resulting data. The initial system analysed a prerecorded sample taken from the rock instrument, splitting it into n-millisecond intervals which were analysed individually. The length of these windows is of important concern, if they are too short then the accuracy of the Fourier Transform is reduced due to the lower frequencies in the signal suffering from clipping but too long and the resulting data is less accurate in its placement of a note within the time periods of the sample. During early testing it was found that a 20ms buffer was sufficient, however later on when much lower frequency keys from the prototype instrument were included in the sample audio it transpired that at least 50ms was required for suitable accuracy.

The fundamental frequency in a signal can be viewed as the lowest frequency peak in a Fourier Transform of ‘significant’ amplitude. For the purpose of this project significant was defined as a peak above a noise threshold of a scaled amplitude of 0.2. A standard hill climbing algorithm was implemented to avoid the over-detection of peaks that are more likely to have been caused by noise and oscillation in a central frequency than by multiple notes. Once the peak detection algorithm compiled a list of significant frequencies in the current sample, the lowest of these is selected and output as F0, thus fulfilling the first minimum requirement of the project.

![Figure 3.1: An plot of the detected peaks in a sample file containing two fundamental frequencies.](image)

The second minimum requirement was to detect the overtones associated with the fundamental frequency, F1..Fn. From investigations with samples of the instrument it became clear that each note had at most two harmonics associated with F0, which in musical theory would lie at integer multiples of the fundamental[4]. However calculations from the ICSRiM samples revealed that for the rocks used in the instrument the first harmonic actually lay in the range of 2.52-2.83 times the fundamental. This stalled the project for a while, the initial plan was for the ratios of all the significant peaks in the signal in relation to the fundamental to be calculated, and the best candidate for the first harmonic being tied to the fundamental. Similar issues could arise with the second harmonic, and with the expectation that multiple notes appearing at once would be common the accuracy of the system would be very poor. These results suggested that it was entirely possible for another note which lay within this range to be deemed the harmonic of the fundamental when in fact it was a unique note in the sample. Eventually it was decided that the best course of action would be to limit to system to semi-polyphony, that is that it...
would be able to detect multiple notes on some occasions but not others. Specifically the system would find the fundamental, associate its first harmonic with it, and then find other notes as long as their fundamental lay within the range of the first fundamental and its first overtone. This would limit the scope of the system, but also reduce the likelihood of false attribution of harmonics and fundamental frequencies.

The third minimum requirement was for the system to measure the decay of the fundamental frequencies and their overtones over time. The final algorithm output to a data file for each line (equating to a timestep in the algorithm); the number of notes present in the timestep, and then an ordered list of the fundamentals and their first harmonic with their associated amplitudes. The listing of the notes in each sample along with their variations in amplitude over time satisfies this minimum requirement.

The final minimum requirement was for the production of a pitch detection algorithm or software prototype for the rock instrument. The combination of the above minimum requirements, along with a Matlab function which combines them to read in a sound file, split it into timesteps specified by the user with an optional noise threshold I believe fulfills this criteria.

3.3 Extensions

As the goal of the project was to develop a pitch detection algorithm for the instrument in development in the Leeds University School of Music, the immediately appealing extension was to adapt the existing system for use in real-time. Work on this began by converting the system to C, using the libraries Portaudio and FFTW to handle Fast Fourier Transforms and real-time recording and analysis of sound. The conversion of the offline system from Matlab to C was completed successfully which should have the benefit of improving the runtime of the system, and the full code can be seen in Appendix B. Due to personal health issues and the aforementioned setback in relation to the second minimum requirement, the schedule for the project was severely pushed back and the development of a real-time system had to be cancelled. The system was near completion, able to receive live input and execute the algorithms on it however it provided inaccurate output and as such has not been included in evaluation. The code can however be found in Appendix B along with the code for the completed project. The delays in the project also mean that the original iterative methodology is not feasible to complete within the given time-frame, and a more basic approach whereby the system is designed, tested and then evaluated is more suitable.
Chapter 4

Evaluation

4.1 Results

To evaluate the system, results needed to be obtained for a number of individual note samples, along with results for samples containing multiple concurrent notes. A ground truth for comparison was also needed, and as the system is designed specifically for the rock instrument in development at Leeds University, there are no other systems available with which to compare meaningfully the results of the system developed within the project except for one which was provided by the Leeds School of Music for the same instrument. The results table below shows the note determination (fundamental frequency decision) for a number of samples of individual notes, split into 50ms intervals and the predominant F0 taken.

<table>
<thead>
<tr>
<th>Sample</th>
<th>F0 (system)</th>
<th>F0 (ground truth)</th>
<th>F1 (system)</th>
<th>Peak -&gt; Fundamental</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hf 1.1</td>
<td>4258</td>
<td>4276.6</td>
<td>-</td>
<td>First</td>
<td>20.60</td>
</tr>
<tr>
<td>Hf 1.2</td>
<td>3806</td>
<td>3818.28</td>
<td>-</td>
<td>First</td>
<td>10.28</td>
</tr>
<tr>
<td>Hf 1.3</td>
<td>3148</td>
<td>3112.3</td>
<td>-</td>
<td>First</td>
<td>-35.70</td>
</tr>
<tr>
<td>Hf 1.4</td>
<td>2543</td>
<td>2520.4</td>
<td>6627</td>
<td>First</td>
<td>-22.60</td>
</tr>
<tr>
<td>Hf 1.5</td>
<td>1900</td>
<td>1901.51</td>
<td>5056</td>
<td>First</td>
<td>1.51</td>
</tr>
<tr>
<td>Hf 1.6</td>
<td>1685</td>
<td>1676.96</td>
<td>-</td>
<td>First</td>
<td>-12.04</td>
</tr>
<tr>
<td>Hf 1.7</td>
<td>1582</td>
<td>1559.19</td>
<td>1523</td>
<td>First</td>
<td>-22.81</td>
</tr>
<tr>
<td>Hf 1.8</td>
<td>1057</td>
<td>1052.71</td>
<td>2854</td>
<td>First</td>
<td>5.71</td>
</tr>
</tbody>
</table>

As the figure shows, the results obtained via the developed system and the ground truth provided by ICSCRiM do not differ greatly, having a variation of 56.3 and an average difference of 16.4Hz. The Peak-Fundamental column relates to which peak in the Fourier Transform is associated with the fundamental frequency, if we use the ground truth as the deciding measure. All results point to the first peak, which coincides with the project algorithms decision to choose the first peak as F0.

Unfortunately due to the scheduling issues encountered during the project ground truths for samples involving multiple concurrent notes were unobtainable, so the semi-polyphonic portion of the system cannot be tested and claims to its accuracy cannot be reasonably made. The above results do however show that the system functions reasonably as a monophonic pitch detection system, as the small variations in frequency between the system and the compared system are likely to be difficult for a human to detect and the system was designed for a musical instrument.

4.2 Evaluation of the Project

In order to assess whether or not the project has been successful it is necessary to consider the objectives and minimum requirements. The objective of the project was to conduct research into the feasibility of a pitch-detection system for a rock-based musical instrument. The development of a basic pitch-detection system shows that it is indeed feasible for a reliable version of such a system to be created, however one more accurate and with greater polyphonic capabilities than mine would be required. Nevertheless the research still proves that it is possible.
The minimum requirements of the project have all been met with varying degrees of success, while I set out to create a polyphonic system and did not succeed, the system still functions adequately as a monophonic one. The system is capable of detecting the fundamental frequency in an audio sample taken from the rock instrument as well as its overtones, it is able to measure the amplitudes of these over time to display their decay, and is capable of outputting the pitches within these samples.

4.3 Possibilities for Further Research

While the minimum requirements for the project were met, there were many avenues which could have been pursued to extend the research. The first of which (real-time analysis) was almost completed, however with the results obtained via the pitch-detection system many other extensions could have been implemented. Given the notes present in a signal and their overtones, it may be possible to create a model of note decay which could be used to augment the sounds produced by the instrument by for example extending the decay so that notes ring longer. A small amount of preliminary research was conducted into this possibility, however to see it implemented fully would be interesting. Another possibility for extension ties in with a scheme currently ongoing involving Leeds University called Ruskin Rocks[15]. One of the aims of the scheme is to promote geology through interactive media such as the rock xylophone, so a project developing it into a multimedia system with visual displays would also be interesting to conduct.

4.4 Conclusions

I have not achieved everything I initially intended to with this project, a combination of improper scheduling, poor health and inexperience with the subject matter and the difficulty involved in the field of research pushed back the completion date and reduced the scope of the project. However I believe the research I conducted led to the design of a rudimentary system that with more time could be made far more useful. I think of the many methods of pitch recognition currently practiced I selected the most suitable one with which to base my system design on and that the system fulfills all the goals set out during the project planning stage.
Bibliography


Appendix A

Reflection on the Project

The undertaking of this final year project has been the most challenging and stressful event in my academic life. Now that it is completed I am both relieved that it is over and glad for the lessons it has taught me to take into the future. Most of these lessons come from problems that I encountered during the course of the project that I feel I could have better dealt with, and seeing the results of dealing with these problems incorrectly will spur me on not to make the same mistakes in future. For the benefit of future students I shall explain some of the problems encountered and what may be a good strategy for avoiding them in future.

- **Poor scheduling**
  Early on in the project I felt I had done enough research to move on to the next stage, however as seminars I attended expected the production of papers recently read and peers on my course were still in the background research stage, I did not continue on. I strongly recommend working at your pace and not what pace you see others working at. While at the beginning of the project I set deadlines and stuck to them strictly, later on when deadlines could not be reasonably met rather than push to finish them quickly to make up lost time I continued at a steady pace. This became an issue later on when there wasn’t enough time to meet further deadlines and the whole project got pushed back.

- **Contact with Third Parties**
  Third parties would include your supervisor, assessor, project coordinator and anyone else who may be aiding you with your project. Throughout my project I did not keep in sufficient contact with my supervisor and many issues or problems I had with the implementation of my project I could have solved much sooner with their aid. One of the primary reasons for this was infrequently checking my university email account due to it rarely being used during the first and second years of my degree. I would advise future students to make sure that they keep in regular contact with their supervisor and make a habit out of checking their university account.

- **Health Concerns**
  I had some issues with stress that greatly disrupted my sleeping patterns and made it very difficult to function normally, rather than consult a doctor I attempted to remedy the problem myself to no success. Towards the final project deadline the issue became worse and I eventually sought out a doctor and contacted the project coordinator, both of whom helped greatly. I strongly urge anyone with health concerns to consult their doctors and let them decide how important or unimportant it is.
Appendix B

System Code

B.1 Matlab System

function [peakx, peaky] = writeNotes(wavfile, filename, timestep, threshold);

%Read Wave file and calculate sample step (sstep) and number of full blocks
%(whole iters)
[x fs y] = wavread(wavfile);
soundtime = length(x) / fs;
sstep = floor((fs/1000)*timestep);
wholeiters = floor(length(x)/sstep);
otelist = [];

%Open output file
results = fopen(filename, 'w');

disp('Processing sound file...');
%Take a chunk of data of size sstep from x, perform FFT and pitch detection
%on it, output to file. Iterate over chunks.
for i = 1:wholeiters
    [amp, freq] = positiveFFT(x(((i-1)*sstep)+1:i*sstep), fs, 1);
    notelist = getNotes(freq, amp, threshold);
    for j = 1:length(notelist)
        fprintf(results, '%f ', notelist(j));
    end
    fprintf(results, '
');
end

%Run FFT processing on remainder
[amp, freq] = positiveFFT(x((wholeiters*sstep)+1:length(x)), fs, 1);
notelist = getNotes(freq, amp, threshold);
for j = 1:length(notelist)
    fprintf(results, '%f ', notelist(j));
end
fprintf(results, '
');
fclose(results);

function [notelist] = getNotes(freq, amp, threshold);

%Reading files in
% [x fs y] = wavread(file);
% [amp, freq] = positiveFFT(x, fs, 1);
absamp = abs(amp);
[peakx, peaky, troughx, troughy] = getPeaks(freq, absamp, threshold);
notelist = [];

%Creating upper and lower troughs
temp = [0 troughx' freq(length(freq))];
troughx = temp';
temp = [0 troughy' 0];
troughy = temp';

p2x = peakx';
p2y = peaky';

%Loop through this until 0/1 peaks left
while (length(p2x) >= 1)
    amplist = [];
pcount = 1;
    %Calculating ratios of peaks in relation to F0
    fund = p2x(1);
    peakx = [peakx(2):length(peakx)];
    ratios = [];
    partials = [fund];

    %Calculate partial nominations
    if length(p2x) > 1
        p2x = p2x(2:length(p2x));
temp2x = p2x;

        for i = 1:length(p2x)
            if (fund == 0)
wibble
            end

wibble
ratios = [ratios 0];
else
    ratios = [ratios (p2x(i)/fund)];

    \%First harmonic found
    if (ratios(i) > 2.6) & (ratios(i) < 2.8)
        partials = [partials p2x(i)];
        pcount = pcount + 1;

        \%Create a temp list of remaining freq peaks
        if (i == 1)
           temp2x = p2x(2:length(p2x));
        else
           temp2x = [p2x(1:i-1) p2x(i+1:length(p2x))];
        end
    end
end

end

\%replace peaks with temp list
p2x = temp2x;

else
    p2x = [];
end

notelist = [notelist pcount];

for (j = 1:length(partials))
    pindex = find(peakx == partials(j));
    yvalue = peaky(pindex);
    notelist = [notelist partials(j) yvalue];
end

end

function [peakx peaky troughx troughy] = getPeaks(x,y, threshold)

[maxlist, minlist] = peakdet(y, threshold, x);

if size(maxlist) > 0
    peakx = maxlist(:,1);
    peaky = maxlist(:,2);
else
    peakx = NaN;
    peaky = NaN;
end

if size(minlist) > 0
    troughx = minlist(:,1);
    troughy = minlist(:,2);
else
    troughx = NaN;
    troughy = NaN;
end
troughy = minlist(:,2);
else
    troughx = NaN;
    troughy = NaN;
end

%plot(x,y);
%hold on;
%plot(peakx, peaky, 'ro');

function [X,freq]=positiveFFT(x,Fs, cut)
N=length(x); %get the number of points
k=0:N-1; %create a vector from 0 to N-1
T=N/Fs; %get the frequency interval
freq=k/T; %create the frequency range

X=fft(x);
%X = X / max(X); % normalize the data

%only want the first half of the FFT, since it is redundant
cutOff = ceil(N/2);

%take only the first half of the spectrum
if cut == 1
    X = X(1:cutOff);
    freq = freq(1:cutOff);
end

function [maxtab, mintab]=peakdet(v, delta, x)

maxtab = [];
mintab = [];

v = v(:);

if nargin < 3
    x = (1:length(v))';
else
    x = x(:);
    if length(v)== length(x)
        error('Input vectors v and x must have same length');
    end
end

if (length(delta(:)))>1
    error('Input argument DELTA must be a scalar');
end

if delta <= 0
    error('Input argument DELTA must be positive');
end
mn = Inf; mx = -Inf;
mmpos = NaN; mxpos = NaN;

lookformax = 1;

for i=1:length(v)
    this = v(i);
    if this > mx, mx = this; mxpos = x(i); end
    if this < mn, mn = this; mmpos = x(i); end
    if lookformax
        if this < mx-delta
            maxtab = [maxtab ; mxpos mx];
            mn = this; mmpos = x(i);
            lookformax = 0;
        end
    else
        if this > mn+delta
            mintab = [mintab ; mmpos mn];
            mx = this; mxpos = x(i);
            lookformax = 1;
        end
    end
end

B.2 C Offline System

#include <stdio.h>
#include <malloc.h>
#include <sndfile.h>

//For FFT
#include <math.h>
#include <complex.h>
#include <fftw3.h>

#include "peakdetect.h"
#include "fftFunc.h"

int main(int argc, char *argv[])
{
    int i, j;
    float *peaks;
    fftw_complex *results;

    printf("Wav Read Test\n");
    if (argc != 2) {
        fprintf(stderr, "Expecting wav file as argument\n");
        return 1;
    }
// Open sound file
SF_INFO sndInfo;
SNDFILE *sndFile = sf_open(argv[1], SFM_READ, &sndInfo);
if (sndFile == NULL) {
    fprintf(stderr, "Error reading source file '%s': %s", argv[1], sf_strerror(sndFile));
    return 1;
}

// Check format - 16bit PCM
if (sndInfo.format != (SF_FORMAT_WAV | SF_FORMAT_PCM_16)) {
    fprintf(stderr, "Input should be 16bit Wav\n");
    sf_close(sndFile);
    return 1;
}

printf("sndInfo.channels: %d\n", sndInfo.channels);

// Check channels - mono
//if (sndInfo.channels != 2) {
//    fprintf(stderr, "Wrong number of channels\n");
//    sf_close(sndFile);
//    return 1;
//}

//Retrieve number of channels
int channels = sndInfo.channels;

int buffsize = sndInfo.frames * channels;

// Allocate memory
float *buffer = malloc(buffsize * sizeof(float));
if (buffer == NULL) {
    fprintf(stderr, "Could not allocate memory for file\n");
    sf_close(sndFile);
    return 1;
}

// Load data into buffer
long numFrames = sf_readf_float(sndFile, buffer, sndInfo.frames);

// Check correct number of samples loaded
if (numFrames != sndInfo.frames) {
    fprintf(stderr, "Did not read enough frames for source\n");
    sf_close(sndFile);
    free(buffer);
    return 1;
}
// Output Info
printf("Read %ld frames from %s, Sample rate: %d, Length: %fs\n",
numFrames, argv[1], sndInfo.samplerate, (float)numFrames/sndInfo.samplerate);

///////////////////////////////////////////////////////////////////////////FFT Calculations///////////////////////////////////////////////////////////////////////////

//Calculate frame chunks
int timestep = 20; //in miliseconds
int sstep = ((sndInfo.samplerate/1000)*timestep);
int wholeiters = numFrames / sstep;
int n = (sstep * channels);
int startpoint;

double absamp;
double absvalues[sstep];

//Vector k of bins
int k[n];
for (j=0; j < n; j++)
{
k[j] = j;
}

//Frequency interval
float t = (float)n/(float)sndInfo.samplerate;

//Calculate the frequency range
float freq[n];
for (j=0; j < n; ++j)
{
freq[j] = (float)k[j]/t;
}

printf("Whole iters: %d", wholeiters);

for (i = 0; i < wholeiters; i++)
{
startpoint = i * sstep;
printf("startpoint: %d\n", startpoint);
//int n = (sndInfo.frames * channels);

results = fftFunc(buffer, startpoint, n);
//Calculate absolute amplitudes
for (j=0; j < sstep/2; ++j)
{
    absamp = sqrt(creal(results[j])*creal(results[j]) + cimag(results[j])*cimag(results[j]));
    absvalues[j] = absamp;
}


//Peak detection
peaks = peakDetection(absvalues, 40, sstep/2);

if ((int)peaks[0] != 0)
{
    for (j = 0; j<(int)peaks[0]; j++)
    {
        printf("Peaks found: %.2f %.0f %.0f\n", peaks[1+2*j], peaks[2+2*j], freq[(int)peaks[2+2*j]]);
    }
}
else
{
    printf("j: %d peaks[0]: %.2f\n", j, peaks[0]);
}

printf("\n----------------------------------------------------------\n");

//Iterate over remaining frame amount
int remainder = buffsize - startpoint;

printf("Last frame! %d\n", (int) remainder);
startpoint = wholeiters * sstep;
printf("startpoint: %d\n", startpoint);
results = fftFunc(buffer, startpoint, buffsize - startpoint);

//Calculate absolute amplitudes
for (j=0; j < remainder; ++j)
absamp = sqrt(creal(results[j])*creal(results[j]) + cimag(results[j])*cimag(results[j]));
absvalues[j] = absamp;

//Pad results with 0 amp
for (j=remainder; j<sstep/2;++j)
{
    absvalues[j] = 0.0;
}

peaks = peakDetection(absvalues, 50, sstep/2);

if ((int)peaks[0] != 0)
{
    for (j = 0; j<(int)peaks[0]; j++)
    {
        printf("Peaks found: %.2f %.0f %.0f\n", peaks[1+2*j], peaks[2+2*j],
        freq[(int)peaks[2+2*j]]);
    }
}
else
{
    printf("j: %d peaks[0]: %.2f\n", j, peaks[0]);
}

printf("\n-------------------------------------\n");

///////////////////////////////////////////////////Clean up///////////////////////////////////////////////////

free(peaks);
fftw_free(results);
sf_close(sndFile);
free(buffer);
return 0;
B.3 C Real-Time System (Incomplete)

```c
#include <stdio.h>
#include <stdlib.h>
#include <sndfile.h>
#include "portaudio.h"

//For FFT
#include <math.h>
#include <complex.h>
#include <fftw3.h>

#include "peakdetect.h"
#include "fftFunc.h"

//Set some constants
#define SAMPLE_RATE (44100)
#define FRAMES_PER_BUFFER (7000) //Smaller due to real-time sound output
#define NUM_CHANNELS (2)
#define PA_SAMPLE_TYPE paFloat32

//Create a sample type
typedef float SAMPLE;

//Prototype functions
int end_function(int error);
int findovertones(void *inputpeaks, void *freq);

/A hollow callback function, just takes the input and puts it in the output buffer
static int hollowCallback( const void *inputBuffer, void *outputBuffer,
    unsigned long framesPerBuffer, const PaStreamCallbackTimeInfo *timeInfo,
    PaStreamCallbackFlags statusFlags, void *userData)
{
    //Variable initialisation
    int i;
    fftw_complex *results;
    float *peaks;
```
//printf("WIBBLE\n");

//Create a read pointer, pointed at the input buffer
SAMPLE *readptr = (SAMPLE*) inputBuffer;

//Create a write pointer, pointed at the output buffer
SAMPLE *writeptr = (SAMPLE*) outputBuffer;

//Deal with silence
if (inputBuffer == NULL)
{
    for (i=0; i<framesPerBuffer; i++)
    {
        *writeptr++ = 0; //left amplitude set to 0
        *writeptr++ = 0; //right amplitude set to 0
    }
}
else
{
    for (i=0; i<framesPerBuffer; i++)
    {
        *writeptr++ = *readptr++; //Set output to input
        *writeptr++ = *readptr++;
    }
}

///////////////////////////////////FFT CODE///////////////////////////////////

//Calculate frame chunks
//int timestep = 20; //in miliseconds
//int sstep = ((SAMPLE_RATE/1000)*timestep);
int n = (framesPerBuffer);
//int startpoint;

int j;
double absamp;
double absvalues[framesPerBuffer];

//Vector k of bins
int k[n];

for (j=0; j < n; j++)


```c

// Frequency interval
float t = (float)n/(float)SAMPLE_RATE;

// Calculate the frequency range
float freq[n];
for (j=0; j < n; ++j)
{
    freq[j] = (float)k[j]/t;
}

// Get FFT for this buffer
results = fftFunc(inputBuffer, 0, framesPerBuffer);

// Calculate absolute amplitudes
for (j=0; j < n/2; ++j)
{
    absamp = sqrt(creal(results[j])*creal(results[j]) + cimag(results[j])*cimag(results[j]));
    absvalues[j] = absamp;
}

// Peak detection code
peaks = peakDetection(absvalues, 100, n/2);

if ((int)peaks[0] != 0)
{
    findovertones(peaks, freq);
    for (j = 0; j<(int)peaks[0]; j++)
    {
        printf("Peaks found: %.2f %.0f %.0f\n", peaks[1+2*j], peaks[2+2*j], freq[(int)peaks[2+2*j]]);
    }
    printf("----------------------------------\n");
}

// Cleanup
```
fftw_free(results);

//Continue
return paContinue;
}

/**
 * 
 */

int main(int argc, char *argv[])
{
    //Create the PortAudio variables needed to open a stream
    PaStreamParameters inputParameters, outputParameters;
    PaStream *stream;
    PaError err = paNoError;

    //fftw_complex *results;

    printf("Running realtimesound.c...\n");

    //Initialize PortAudio
    err = Pa_Initialize();
    if (err != paNoError) return end_function(err);

    //Setup the input device (retrieves default)
    inputParameters.device = Pa_GetDefaultInputDevice();
    inputParameters.channelCount = 2;
    inputParameters.sampleFormat = PA_SAMPLE_TYPE;
    inputParameters.suggestedLatency = Pa_GetDeviceInfo(inputParameters.device) ->
    defaultLowInputLatency;
    inputParameters.hostApiSpecificStreamInfo = NULL;

    //Setup the output device (retrieves default)
    outputParameters.device = Pa_GetDefaultOutputDevice();
    outputParameters.channelCount = 2;
    outputParameters.sampleFormat = PA_SAMPLE_TYPE;
    outputParameters.suggestedLatency = Pa_GetDeviceInfo(outputParameters.device) ->
    defaultLowOutputLatency;
    outputParameters.hostApiSpecificStreamInfo = NULL;

    //Open up the stream, call hollow callback function
    err = Pa_OpenStream( &stream, &inputParameters, &outputParameters, SAMPLE_RATE,
    FRAMES_PER_BUFFER, paClipOff /*disables clipping*/, hollowCallback,
    NULL /*no storing*/);
    if (err != paNoError) return end_function(err);

    //Start the stream
    err = Pa_StartStream(stream);
    if (err != paNoError) return end_function(err);
}
printf("System initialized!\n");fflush(stdout);

// Goes to callback until a char is received?
printf("Hit ENTER to stop program.\n");fflush(stdout);
getchar();

// Close the stream
err = Pa_CloseStream(stream);
if (err != paNoError) return end_function(err);
printf("Stream closed, shutting down.\n");fflush(stdout);

return end_function(err);

}/**

// End function, cleaning up and returning
int end_function(int error)
{

// Close PortAudio
Pa_Terminate();

// Print out error message if an error has been encountered
if (error != paNoError)
{
    fprintf(stderr, "An error occurred during execution.\n");
    fprintf(stderr, "Error number: %d\n", error);
    fprintf(stderr, "Error message: %s\n", Pa_GetErrorText(error));
    error = 1;
}

// Return value to main
return error;
}

// Overtone association function
int findovertones(void *inputpeaks, void *freq)
{
    int j, k;
    int count = 1;
    int arrayend = 0;

    float *input = inputpeaks;
    int size = input[0];
float *frequencies = freq;
int fundamental = (int) frequencies[(int)input[2]];
float lower = 2.6;
float upper = 50;

float *notes = malloc(50*sizeof(float));

if (size > 1)
{
    float partials[size];
    partials[0] = fundamental;
    float ratio;

    for (j = 0; j<size-1; j++)
    {
        ratio = frequencies[(int)input[4+2*j]] / fundamental;
        if (ratio > lower && ratio < upper)
        {
            partials[count] = frequencies[(int)input[4+2*j]];
            //printf("partials[count] = %.2f
", partials[count]);
            count++;
        }
    }

    for (k = narrayend; k < narrayend+count; k++)
    {
        notes[k] = partials[k];
        //printf("notes[k] = %.0f", notes[k]);
    }

    for (k = 0; k < narrayend+count; k++)
    {
        printf("notes[k] = %.0f\n", notes[k]);
    }

}

printf("Fundamental: %d\n", fundamental);

free(notes);
return 0;