

Final Year Project Report

Automatic Music Transcription

David Jones

A dissertation submitted in part fulfillment of the degree of

BSc. (hons) in Computer Science

Supervisor: Dr. Chris Bleakley



UCD School of Computer Science and Informatics
College of Engineering Mathematical and Physical Sciences
University College Dublin

May 5, 2011

Contents

Abstract	2
1 Project Specification	4
1.1 General Information	4
1.2 Mandatory	4
1.3 Discretionary	4
1.4 Exceptional	5
2 Introduction	6
2.1 Project Motivation	6
2.2 Project Aims	6
2.3 Applications of Automatic Music Transcription	6
2.4 Report Structure	7
3 Background Research	8
3.1 Related Work	8
3.2 Tools and Technologies	10
4 Project Approach	13
5 Design and Implementation	15
5.1 Algorithm	15
5.2 Multi-Instrument Algorithm Extension	16
5.3 Matlab Implementation	17
5.4 Java Implementation	19
5.5 Problems Encountered	20
6 Testing and Evaluation	22
6.1 Pure Sine Wave Tones	22
6.2 Natural Instruments	24
6.3 Comparison of Java and Matlab Implementations	25
7 Conclusions and Future Work	28

Abstract

Automatic Music Transcription has long been attempted, and the proof of how difficult it is to produce an accurate algorithm to process the many facets of a piece of music can be seen by the various ways in which it has been attempted, and the different aspects involved. The aim of this project is to develop and test a system comprising of a musical note detection algorithm which produces a piece of sheet music representing the .wav file input to the system. Sheet music is the graphical representation of a piece of music, which when applied to voice or a musical instrument, can aid in the reproduction of said music. This system could be used as a great learning tool for musicians because as long as they have the recording for a song they like, it can be learned with greater ease when the corresponding sheet music is provided.

This project's aim is to successfully implement an automatic music transcription algorithm which will analyze a .wav file, find the individual notes that comprise that audio file, and after finding the corresponding time each note is played for, output the sheet music representation. This process will be performed for both pure sine wave tones as well as natural instruments.

Acknowledgements

I would like to acknowledge a number of people in assisting in the completion of this project. First of all I would like to thank Dr. Chris Bleakley for his guidance in how to best approach a project of this type and for using his knowledge and experience to provide me with insight into the field of signal processing. I would also like to thank Mark Kane of the MUSTER research group for giving me ideas on different techniques I could use to obtain accurate results. My class as a whole were also very supportive in the development of this project as they were also willing to let me talk through and problems I was having and give advice on what could be done to resolve it. Finally I am extremely grateful for all the support my father gave me during the development process.

Chapter 1: Project Specification

1.1 General Information

Music transcription is the process of listening to a piece of music and writing down the musical notation for the piece. This is useful for learning a piece when the musician does not have access to the original written score.

The goal of this project is to develop a program which will automatically transcribe a piece of music from an audio recording in uncompressed digital format to written form.

1.2 Mandatory

Develop and test a program which will:

- Read an audio track in an uncompressed format. e.g. .wav
- Analyze the audio track to determine the fundamental frequency of any musical notes.
- Convert the sequence of identified musical notes to abc notation.
- Save the abc notation in an ASCII file.

The program need not determine the relative duration of the notes (i.e. assume that all notes are crochets).

Using the program:

- Perform automatic music transcription for audio tracks containing a sequence of single sine wave tones
- Determine the accuracy of the transcription

1.3 Discretionary

Extend the program to include the following features:

- analyze the musical notes to determine their relative duration and include this information in the abc notation (i.e. include quavers, minims, etc)

- conversion of the abc notation to graphical (modern musical) notation using the abc4j library

1.4 Exceptional

Improve the note recognition algorithm such that the program can accurately recognize notes from natural instruments. Preferably the algorithm should work for multiple instruments and multiple notes being played at the same time.

Determine the accuracy of the new algorithm by accessing its performance over a number of pieces.

Chapter 2: Introduction

2.1 Project Motivation

The idea for the project was first realized in the April before commencing development. The developer is a musician and wanted to find something that would combine the two disciplines of computer science and music while also finding something that could be of everyday use. Also the way in which digital audio is becoming more and more integrated into every day society shows that this application should prove more and more useful as time passes. “Automatically extracting music information is gaining importance as a way to structure and organize the increasingly large numbers of music files available digitally on the Web. It is very likely that in the near future all recorded music in human history will be available on the Web”[1]. The project proposal of automatic music transcription was the result.

2.2 Project Aims

The main aim of this project is to develop a system which would allow for the accurate transcription of an audio track to modern classical notation. This means the ability to read in a .wav file as requested by the user and giving back an easily readable piece of sheet music which correctly depicts the recorded track (containing either computer generated or natural instrument audio) with a minimal error rate. The project needs to provide an easily navigable user interface which allows for the naming and addition of transcriptions.

2.3 Applications of Automatic Music Transcription

There are many applications of Automatic Music Transcription, one of the main focuses being that of a teaching and learning tool. For example, a music student would be able to practice playing music from sheet music, the music played by the student could then be automatically transcribed, before being compared to the existing sheet music and the accuracy of the students playing could be determined. It could also be used as a much cheaper and less labour intensive way to transcribe music, instead of having to hire music professionals to listen to music and attempt to transcribe it, the program can be run instead with minimal effort.

2.4 Report Structure

This report has been written to describe all of the research and preparation needed for this project as well as the extent of progress made on the development of the system, which demonstrates the level of success achieved during implementation. This also gives an overview of the various difficulties encountered and the amount work still to be completed.

This report begins with a chapter on background research which focuses on the related research in the field of music transcription, in the areas of both pitch and tempo detection, as well as any existing programs that have similar functionality. This section also talks about the different tools and technologies that were used in order to assist in the completion of the project.

The subsequent chapters give the reader an overview of the approach taken and the different aspects of implementation including a description of how the algorithm is intended to work. The final chapters then cover the testing and evaluation of the project and give the developers conclusions and recommendations for future work.

Chapter 3: Background Research

Research into the various methods which attempt to provide accurate music transcription provided a wide array of knowledge. This allowed for an insight into the pros and cons of using certain techniques within the field of signal processing. Papers were read on automatic music transcription and note detection extensively, information on digital signal processing in general was also researched. The main source of research within the field of digital signal processing in general was the “The Scientist and Engineers Guide to Digital Signal Processing”.

3.1 Related Work

There have been many different ways in which automatic music transcription algorithms have been attempted to be successfully implemented. These include a number of different digital signal processing techniques such as Fourier Transform and Cepstral Analysis for note detection, and the Comb Filter and Statistical Analysis techniques for beat detection. The papers used while researching were split into two categories. The first of which applied to the development of a note detection algorithm, while the second covers the papers used to help in determining the best way to measure musical tempo and beat.

3.1.1 Note Detection Research

The first paper analysed for information in order to supplement the development of the project was “Musical Fundamental Frequency Tracking Using a Pattern Recognition Method”[3]. A similar, more primitive method of pattern matching to develop a fundamental frequency tracker was later implemented once the problem of returning a note in the wrong octave became apparent. The problem of the “missing fundamental” that occurs when processing audio signals was solved in this paper by comparing the harmonics present to those stored in a template and finding which matched best. Cross correlation (which in signal processing is used to measure the similarity between two waveforms) was used to find the optimal pattern that matches the harmonics representing each note. Although one of the errors discussed in the conclusion of the paper was that the wrong octave would be selected. This is the same problem later encountered in this project.

The pattern matching technique that is to be implemented in this project is similar to that found in “Tune Retrieval from Acoustic Input”[4]. The method developed here was described as “essentially a matter of matching input strings against a database”[4]. This form of pattern recognition could be applied to the use of a database containing the power of the FFT representing each of the notes on the musical scale. This is the main adaption that needs to be achieved by the system which allows for the comparison of the signal values of individual notes instead of the comparison of sequences of notes.

Another method of tracking the fundamental frequency of an audio signal is proposed by Anssi P. Klapuri using a preprocessing technique followed by the use of the “harmonic-

ity principle”[5] and “spectral smoothness principle”[5]. The harmonicity principle is based upon the utilization of “harmonic relationships between frequency components, without assuming ideal harmonicity”[5]. Due to the fact that simple harmonic relationships are favored over dissonant ones in simultaneously played notes, the worst cases of harmonic relationships must be dealt with. This is done using the spectral smoothness principle proposed by Klapuri. When the dense harmonics of lower sounds coincide with partials of the higher ones, the “spectrum of the lower-pitched sound is smoothed”[5], which solves the above case of coinciding harmonics. Due to the large range of techniques used in the application of spectral smoothing, this method of fundamental frequency estimation has been written off as too time consuming to be implemented within the required time-frame.

In “An Adaptive Technique for Automated Recognition of Musical Tones”[6], a linear predictor is used in the event detection stage. Linear prediction can be described as follows: “Linear prediction is a technique of time series analysis, that emerges from the examination of linear systems. Using linear prediction, the parameters of a such a system can be determined by analysing the systems inputs and outputs”[7]. Then pitch detection was performed via the use of an FFT of the interval which was calculated using the event detection algorithm.

There were a number of other papers that were looked at in order to research the different forms of note detection that could apply to this application. These were papers such as “Automatic Polyphonic Piano Note Extraction Using Fuzzy Logic in a Blackboard System”[8] and “Maximum Likelihood Pitch Estimation”[9], but these papers were, for the most part, an overlap of those mentioned above.

3.1.2 Tempo Detection Research

Although the tracking of musical tempo was secondary to that of note detection, some research was needed on this topic to have some insight into what would later need to be accomplished. “Tempo and Beat Estimation of Musical Signals”[10] was one of 3 key papers used in the research of a tempo detection algorithm, alongside “A Tutorial on Onset Detection in Music Signals”[11] and “A Hybrid Approach to Musical Note Onset Detection”[12]. The difficulty in finding the beat of a piece of music varies dramatically depending on the musical genre being analysed. Musical genres that use a lot of percussion can be easily analysed. but “the robustness of beat tracking systems is often much less guaranteed when dealing with classical music”[10]. Due to the fact that the Automatic Music Transcription project needs to be able to accurately process all genres of music, a custom built tempo detection algorithm will need to be developed. The three papers mentioned above include details of different note onset detection algorithms, the results of which help to find the tempo of the music by finding the intervals between musical events found by these algorithms.

In note onset detection algorithms, “A central issue here is to make a clear distinction between the related concepts of transients, onsets and attacks.”[11] Due to the fact that different applications have different needs, these distinctions need to be made clear. In the paper cited above, the definitions for each are given as followed:

- “The *attack* of the note is the time interval during which the amplitude envelope increases.”[11]
- The *transient* can be described as “intervals during which the signal evolves quickly in some nontrivial or relatively unpredictable way.”[11]

- “The *onset* of the note is a single instant chosen to mark the temporally extended transient.”[11]

A note can ideally be differentiated from others contained in the audio signal using these methods by finding the midpoint of the transient, which is represented by finding the onset of the note, the amount of time for which the attack of the note occurs, and the same amount of time after the note begins to decay. The conclusions of this paper may strongly influence the development of this project in that depending on the type of audio signal being analysed, a different onset detection function should be used, however, the best overall results achieved were obtained via the use of statistical methods with a suitable training set.

In the final paper on beat detection which uses a hybrid approach, is based around a “sub-band and hybrid detection scheme”[12]. This involves the splitting up of the frequency range of the FFT into sub-ranges, or subbands. This was done by splitting the first half of the FFT output into 5 subbands and performing individual analysis on each of them. This approach proved to be too complex to attempt to implement in the time period allowed for development due to the fact that there were different detection algorithms needed, depending on which frequency range the subband represented. For example, “we propose using standard energy content analysis only for the upper subbands”[12] and “For the lower two frequency subbands...we propose the use of a distance measure between the vectors for each frame of an FFT”[12].

3.2 Tools and Technologies

This section aims to provide a general overview of the different tools and technologies required in the development of this project and how they aided in the systems implementation.

3.2.1 Algorithm Development Using Matlab

In order to develop an accurate algorithm for music transcription while also keeping the amount of time spent debugging to a minimum, the program Matlab was used, which is described by its developers as “a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.”[16]. Once the algorithm is developed, it can then in theory be translated to Java with minimal bug fixing and a very low rate of error. The algorithm will be developed in an incremental way, by developing a small section of the algorithm, and once this is fully functional, a new file is created in which the previous functionality can be extended. In this way a store of all previous milestones and developmental breakthroughs are stored for future reference.

3.2.2 WAV File Overview

The WAV acronym stands for Waveform Audio Format which is a high-quality file format used for storing an audio bitstream. It was introduced in 1991 by Microsoft and IBM for use on the Windows 3.1 operating system and is used for applications such as CDs. The format is based on the Resource Interchange File Format (RIFF), this stores chunks and sub-chunks

which are each indexed. The usual bitstream encoding is the linear pulse-code modulation (LPCM) format.

There are a number of reasons why the .wav format was selected for the development of this project. One of the main reasons is due to the fact that these files are uncompressed, they contain more data, allowing for the production of better, more detailed audio. One of the weaknesses of the WAV format is the size needed to store the audio, for example a WAV file generally needs 10MB of memory for every 1 minute of audio, whereas the MP3 format requires about 1MB for every 1 minute of audio. This could result in a longer file processing time within the context of this project, but will allow for the implementation of a simple algorithm. It is also one of the most widely supported digital audio formats making it highly useful, while ensuring a high level of documentation and support for it.

3.2.3 Tools for Sound Generation

There were a number of pieces of software used to aid in the generation of sound files containing both natural instruments and pure tones. The two core programs for sound generation in this system were Matlab and GarageBand. For the accurate generation of pure sine wave tones, Matlab was used as it allowed for the accurate generation of tones of a certain frequency and of a particular length. In order to find the frequencies corresponding to each note, a site developed at Michigan Technological University was used[17]. This site also became useful during the first method attempted in the pure tone implementation.

For the generation of natural instrument audio, GarageBand was used due to its ability to use a musical keyboard which helps to make sure that the notes being played are of a certain note while also allowing for the ability to produce multi-instrument tracks. The lack of noise included with the audio generated helps in the calculation of the systems accuracy although the addition of noise is later performed as some form of noise is present in most recordings and this needs to be taken into account.

Both Audacity and iTunes were used in order to provide additional processing of the generated audio. Audacity was used to clip unwanted segments from the generated audio file, for example prolonged periods of silence before and after the audio files main content. Whereas iTunes was simply used as a method of converting audio files to correctly structured .wav files.

3.2.4 Java Libraries

There are a number of Java libraries available which allow for the integration of functional components required by the application. The first of these is the abc4j library which is used to handle the management of musical notation using Java[18]. The main need of this library within the context of the project is that of retrieving a tune from an abc file and displaying the corresponding sheet music. It also has the ability to store the generated music as a file, therefore allowing it to be stored for later viewing, instead of having to run the transcription process again in order to view it.

Another important library being used is “JTransforms” which is described as “the first, open source, multi-threaded FFT library written in pure Java”[19]. This is very important as it allows for the use of the FFT methods contained in this library instead of the developer having to custom build their own. This saves a lot of time as the creation of a custom FFT

algorithm would have been very time consuming, and the use of an external library allows the focus to be put on some of the more important areas of the project.

Finally the core of the project is built around the Java Sound API. This is used to control the input and output of the audio files being used. It gives a high amount of low level control which will allow for the input audio file to be read in, the details of which can then being stored for later processing by the fourier transform and library comparison mechanisms. Closely related to the Java Sound API is that of the Java Media Framework, which is involved in the capturing and playing back of time-based media such as audio and video. But due to the fact that this only provides high level features, it was decided that only the Java Sound API will be used for the purposes of this application.

3.2.5 Current Software

A number of software programs have been developed in the past but do not fully implement an all encompassing program for music transcription. There are a number of limitations imposed on these programs such as having to find a musical solo in order to make sure it doesn't become confused by multi-instrument tracks (this limitation applies to the PitchScope[13] software) and also the Solo Explorer[14] software). The successful processing of these multi-instrument tracks is one aspect this project aims to overcome. In December 2010 there was also a piece of software revealed which specialized in what the developers are describing as "Re-performance". This is the process of audio processing which results in the production of music played in the style of certain musicians[15].

Chapter 4: Project Approach

Before any development could be started, it became apparent that a large amount of research needed to be completed in order to try and combat the many problems that have plagued the area of musical signal processing for a long time. This mainly relates to the analysis of natural instruments compared to the relatively easy task of processing pure sine wave tones.

Music transcription can be described as the process of reducing live or recorded music to notation. The process of making a program that will do this automatically has a number of problems that will need to be overcome. The first of these being that each of the individual notes must be recognized. This can be done by changing the time domain into a frequency domain, which is achieved within this project by using Fast Fourier Transform(FFT) which can compute the frequency content of a signal such as the contents of a .wav file. Once the FFT of the signal has been obtained, the next process to be performed to allow for the matching of notes, was narrowed down to 2 possible options, this was either to make a pattern matching algorithm, or else attempt cepstral analysis. Cepstral analysis can be described as taking the Fourier Transform of the log spectrum. Due to the fact that there are a high number of calculations that need to be performed, including multiple fourier transforms, this would be computationally intensive. It was therefore decided that the pattern matching technique would suit the needs of the project much better.

Another problem that needed to be solved is that the beat and tempo of the piece of music has to be detected. This project aims to overcome this problem by using statistical analysis of the sound energy at different points within the music. This is done by using the fundamental frequency and power of a peak, and comparing those values to the peaks that appear before and after them in the elements of the signal. The comb filter technique may also be attempted depending on time constraints in order to find which of the two methods provide the best results. The comb filter is a bandpass filter used to process signals by mixing the original signal with a delayed signal. This allows for the blocking of harmonics and unwanted frequencies.

The first step in the developmental phase of this project was to assign a certain amount of time for different functionality that was assigned in the project specification. The allocated time would need to allow for an adequate amount of flexibility in order to correctly handle any problems that should occur in the different stages of development. The first task to be completed was to research the many forms of signal processing methods involved in the act of music transcription and use the results to decide the best methods to be used for music transcription.

Following the research performed, an incremental approach to software development was taken which relies on the building of functionality upon previously implemented software. This was to be done by implementing just an individual piece of functionality and making sure that it performs to the level necessary, before adding another piece of functionality upon that. This approach is a good way to help in the time management aspects of the project in order to maintain a good idea of how the project is progressing.

It was decided that due to the complicated nature of the project at hand that it would be beneficial to perform the implementation in two stages beginning with a Matlab development stage to allow for rapid development of an algorithm to provide accurate transcriptions of audio. This would be followed by the Java implementation which would convert the algorithm developed into Java. The reason the matlab implementation could not be used as the

final deliverable was due to the fact that it used a command line interface, therefore reducing the user accessibility of the completed application. Java was chosen due to the developers familiarity with the language as well as it allowing for the easy creation of a user-friendly interface.

Chapter 5: Design and Implementation

5.1 Algorithm

The structure of the transcription process is separated into a series of distinct stages, the first of which is to add each of the notes to the note library. This is done by iterating through a folder of .wav files (each file corresponding to one note on the musical scale) and adding the result of their normalised power to a .txt file with an appropriate name. This process only needs to be performed once as each of the note values are stored in persistent storage. However if more notes need to be added to the library, the process will have to be performed on all of the notes again. Once the GUI has been implemented this can be changed to allow for the addition of new notes to the library and only perform processing on them, instead of having to perform the unnecessary task of re-processing all of the files.

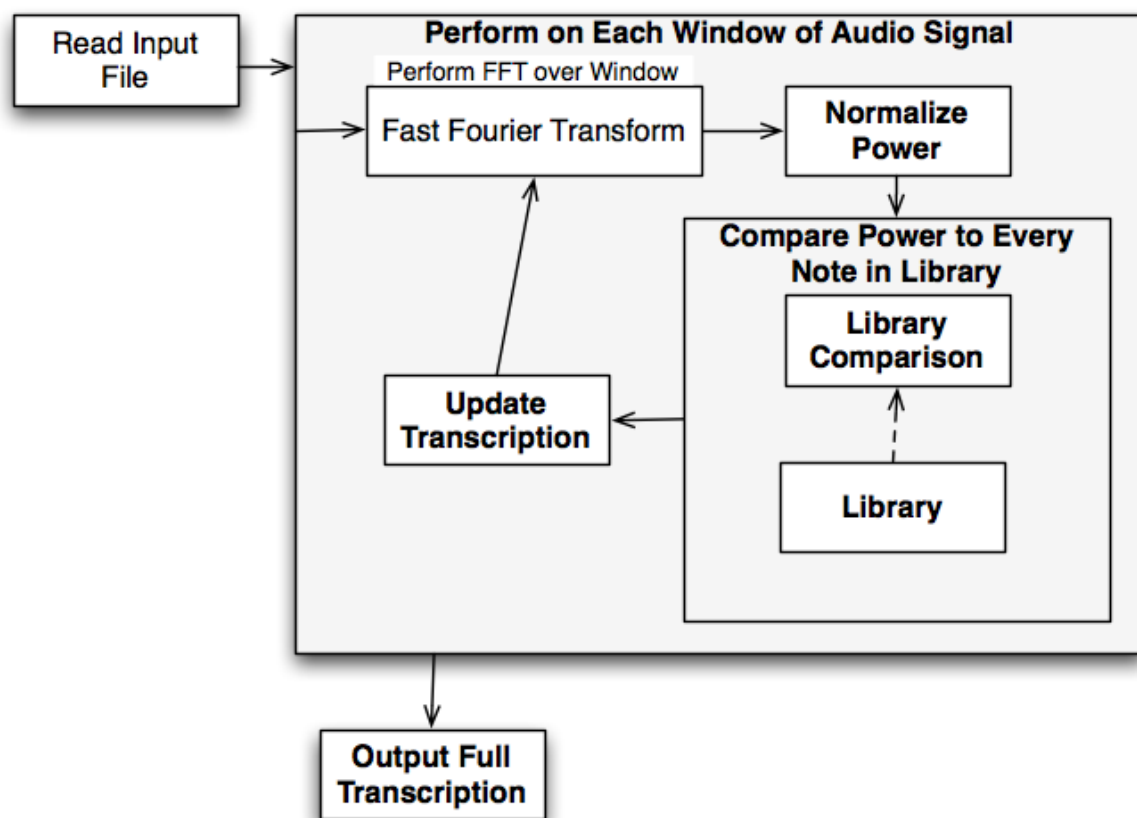


Figure 5.1: Overview of the Transcription Process.

The diagram above provides a basic representation of the transcription process and now each of the steps in the diagram will be explained in slightly more detail. This begins with the WAV file first being input into the system and storing its signal data. The signal data is processed in a series of windows, this has been set to 4096 as it allows for good accuracy and speed with the Fast Fourier Transform Algorithm which maps time sequences into a frequency representation.

The power of this, which is demonstrated in Fig 5.1, is then normalised to allow for easy comparison between those being output at the point of execution, and those being stored in the note library for fingerprinting purposes. The normalised power representing the current window of the input signal is then compared against every note contained in the library in order to find the one that most closely represents it.

The transcription being built is then updated depending on the result from the library comparison stage. The note name is appended to the transcription along with the corresponding duration. The durations of the notes are determined by finding the energy of them and increasing the duration value associated with it while the signal's energy is decreasing (representing note decay) while also maintaining the same fundamental frequency. This continues until every window representing the input signal has been processed, at which stage the final tune output is presented.

5.2 Multi-Instrument Algorithm Extension

Considering the great difficulty experienced in the implementation of a multi-instrument transcription system, it was decided to cover the details of the developer's attempt in a separate section. The main idea behind the attempt of multi-instrument transcription was to extend the library system that had already been implemented. This focused on the calculation of the energy found within each window, and the subtraction of the nearest corresponding library reference. With the subtraction of each library reference, the energy in the window recalculated, and the subtraction of the nearest matching library reference would continue until the energy in that window would fall below the set threshold. The image below shows the plots of an FFT window from a multi-instrument track and the plot of the nearest matching library reference.

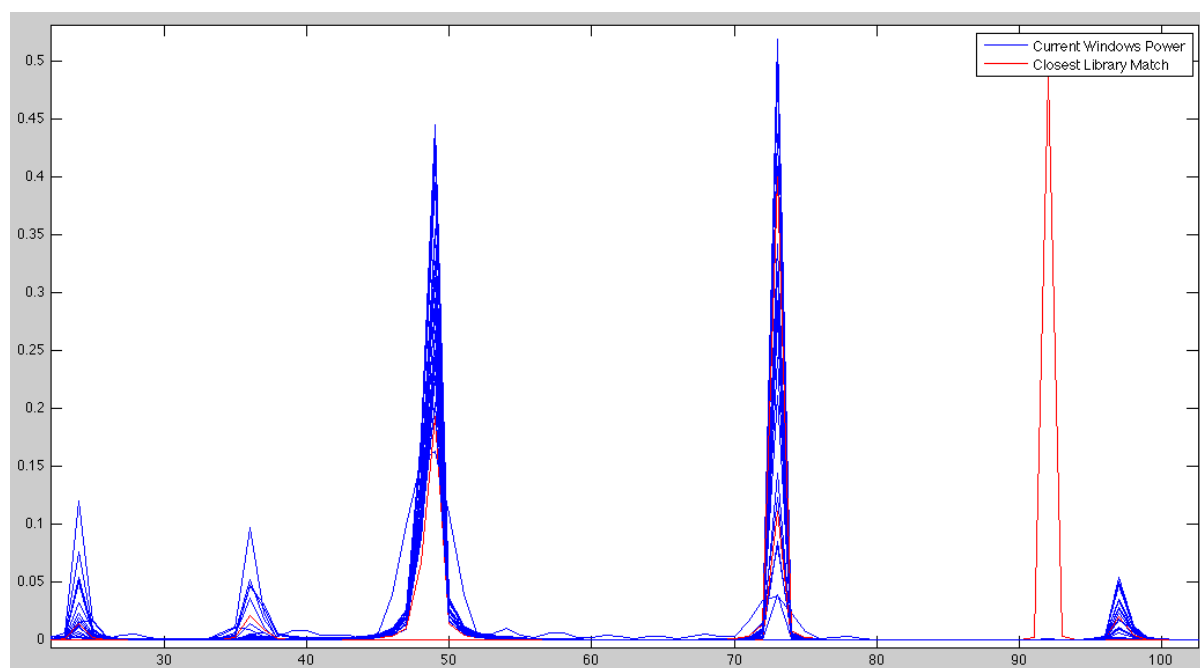


Figure 5.2: Multi-instrument plot with nearest match.

When the nearest library match is subtracted from the window's power, the algorithm makes sure that the remaining power at any given index in the window doesn't become negative, therefore setting any negative numbers in the power array to be 0. A version of this implementation was successful up to a point using Matlab. For example, when an audio file was input that composed of both a C3 and G3 being played simultaneously, the notes the program found were these two notes, as well as a single instance of another note which was removed after being interpreted as noise. The processing of multi-instrument tracks caused a high increase in run time as the the algorithm had to be run a number of times for every FFT window.

Due to the fact that this was a trial implementation, the degree to which it could be tested was limited. The testing performed produced mixed results but the developer is confident that if some more time was available this could be significantly improved.

5.3 Matlab Implementation

The Matlab system is currently at a stage where the individual notes of a tune and their corresponding lengths are being interpreted with almost 100% accuracy. This was achieved due to the fact that Matlab allows for the easy modelling of algorithms. The system development was undertaken in a series of stages. Development began with the production of a simple program to read in a .wav file which contained a single note before trying to accurately calculate which note it was. This was done using two arrays, one containing the names of each note, with the other containing the corresponding fundamental frequencies. This allowed for the calculation of the fundamental frequency of the note contained in the recording using an FFT algorithm, before a comparison to the frequencies in the array was performed. This allowed for easy matching, and therefore the element containing the closest frequency, would be the corresponding index from which to retrieve the note name.

The method in which the durations for each of the notes were calculated was via the use of two arrays, one that would store the note names, and the other containing the note durations. This was achieved by calculating the fundamental frequency of the first section of the tune, and incrementing a counter which counted the number of sections for which the fundamental frequency remained the same while the power decreased. Once the fundamental frequency changed or the power of the section increased, the previous note name, and corresponding length were added to the appropriate arrays. The positions in which two notes overlapped were filtered out by putting a minimum length on a note, and then distributing any notes with a value below this to the surrounding note lengths. Once this distribution had been completed for all notes, the beat length for the tune was calculated using the modulus of the lengths of all notes and finding the lowest common divisor.

The biggest set back encountered in this project was a limitation presented by the use of the FFT technique. This made it much harder to locate the octaves in which the notes belonged due to the fact that occasionally the peak corresponding to the fundamental frequency of the note would appear to be missing, therefore placing the note in an octave above where it should be placed. In order to find a solution to this, it was thought necessary to develop a library system. This involved the building of a folder containing .wav files which represented every note on the musical scale. These were then processed and the results stored as .txt files. This allowed for the problem of the missing fundamental frequency to be ignored as the FFT could simply be matched to those found in the tune being transcribed. There are two possible methods for pattern matching currently being explored, these involve the matching

of the power vector obtained by the FFT, or else by the calculation of the energy of the FFT in decibels which allows for a bigger differentiation in values across the length of the array. These two methods were tested extensively to find the one that provides a higher accuracy of note detection. The graphical representation of an octave after being processed by each method is represented in the figures below.

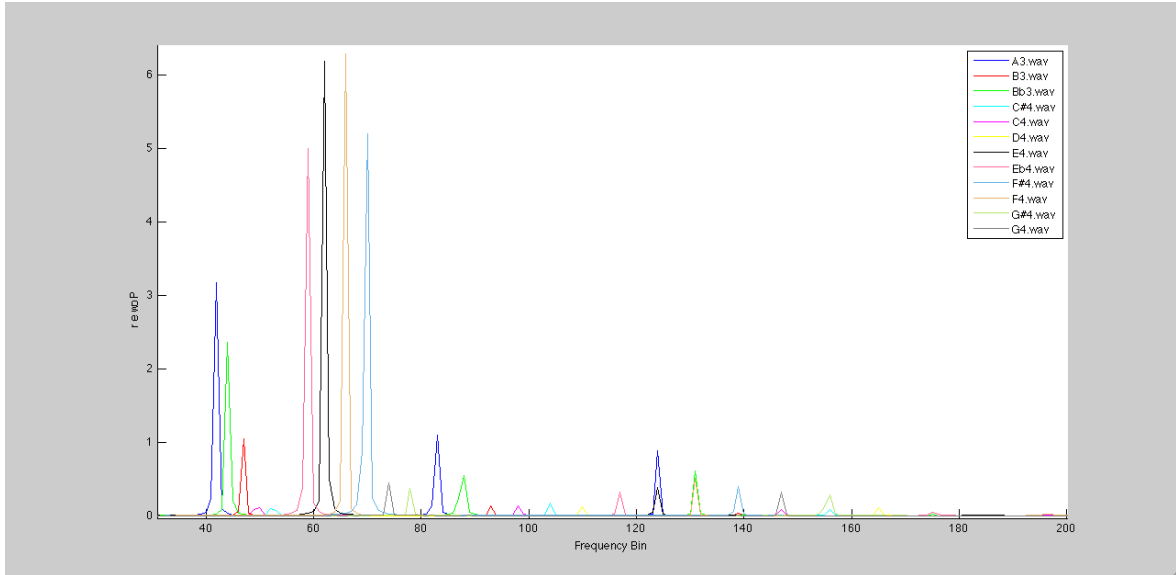


Figure 5.3: Power of FFT Output.

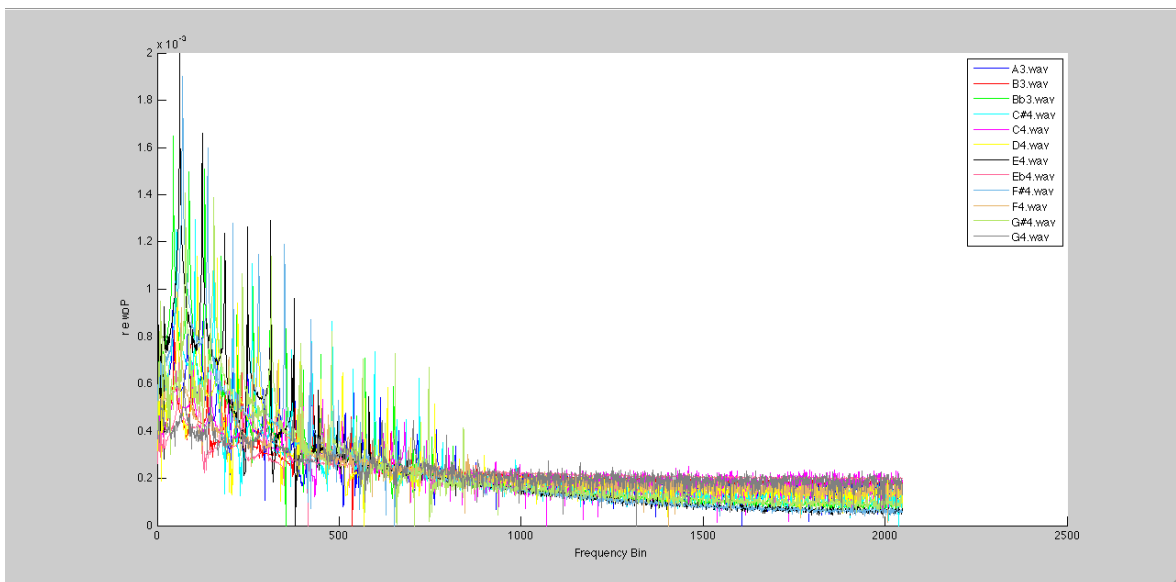


Figure 5.4: Energy in Decibels Output.

As represented by the two figures above, the way in which the signal information is represented varies greatly, allowing for a large difference in the results obtained by both methods. The results obtained showed that the energy in decibels method did not function correctly due to the large variance in values at the start of the range, and the difference between each reducing as the position in the range increases. This resulted in the note that had the highest energy at the start of the range being returned as the successful candidate every time. Therefore the method of analysis using the FFT power was chosen to be used in the remainder of the project.

In order to make sure that any big problems were removed from the project before beginning implementation on the next stage, test runs were performed on a number of tunes in order to make sure that the program performed the transcription with a relatively high level of success. This helps to make sure that the problems do not escalate and can still be removed when they are in their simplest form.

5.4 Java Implementation

The Java implementation of this project was much different to that developed using Matlab. This was mainly due to the fact that there were a number of mathematical operations that were available for use within Matlab but had to be implemented by the developer for use with processing via Java. The main component that was missing was that of an FFT algorithm. Fortunately this was available in an external library called "JTransforms" which was covered in the Tools and Technologies section of the Background Research chapter. This saved a great amount of time as the implementation of a custom FFT algorithm for this project would have been a long and arduous task. There was also the abc4j library, also mentioned in the Tools and Technologies section, which allowed for the conversion of correctly structured abc notation into the corresponding sheet music. This library also aided the development of this project greatly as it removed the need for a custom graphics library to be created.

There were a number of core features that were developed first in order to get the Java development started. The first was a simple GUI to allow for the selection of files to transcribe, instead of having to hard-code the file name every time it needed to be changed. Next was the development of a mathematical library to allow for the use of the operations present in the Matlab development environment but missing from the java JDK. This involved the creation of dot-multiplication and dot-division methods, as well as a method which retrieved the conjugate of the float array returned by the JTransforms library. There were also a number of small supplementary methods which could be called for repetitive tasks.

For the transcription of the pure sine wave tones, there were two methods tried taken. The first involved calculating the fundamental frequency of each window of the FFT. This obtained frequency would then be compared to the frequency array, with the position of the minimum difference giving the index of the note the frequency corresponds to. The second method is the same as the approach taken for natural instruments, by comparing power outputs of FFT windows against those found in the library. With great surprise it was found that these two methods both created very similar results due to a number of issues found in both. Due to the large amount of storage space having an extra sine wave library would use, it was decided that the first approach involving the calculation of the fundamental frequency would be used as the final sine wave implementation.

When plotting the output from the JTransforms FFT method, and comparing this to the

Matlab output, the graphs were slightly different, the main difference being that the secondary peaks in the power graphs were much lower relative to the highest peak when using JTransforms, than when the Matlab FFT function was used. Due to this difference in output, the initial approach of using the note library created during the Matlab development phase was seen to be a problem. This resulted in a slight reduction in accuracy for the overall implementation. This could be remedied with the implementation of a library managed system for Java but due to time constraints, this was considered a minor issue.

The overall system architecture can be illustrated as a series of five distinct phases that are necessary to complete the transcription process. A quick overview of the steps are illustrated below:

- **File Input and Management** - This phase involves the input of an audio file, followed by the processing of its byte information to make sure it is in the correct format to allow for transcription to take place.
- **FFT and Power Retrieval** - This step takes the information obtained from the first phase and applies the FFT algorithm to the audio which is divided into segments called "windows". The power of each segment is then calculated for use in the 3rd phase.
- **Library Lookup** - The library lookup phase involves the comparison of each window's power to the reference files in the library. The closest match being chosen in order to obtain the note which is believed to be contained in that window.
- **Note Length Analysis** - Once all of the notes have been obtained, it is necessary to analyse the sequence returned to calculate whether consecutive notes are new notes, or the same note still continuing. This phase also checks for notes that occur once in a section of dominated by 2 notes, this note can then be interpreted as noise and removed from the transcription.
- **ABC Generation and Graphical Output** - The final phase involves the correct formatting of the program output to ABC notation as well as the use of the abc4j library to generate the sheet music as graphical output.

5.5 Problems Encountered

There was a wide range of problems encountered while developing the system. The most influential problem encountered while progressing through development was the occurrence of missing fundamental frequency peaks in the FFT output. This was due to the significant difference in the type of implementation required to process pure tones and those produced by natural instruments. This is due to the fact that the FFT returned after processing of a pure tone contains only a singular peak representing the fundamental frequency, whereas the FFT returned after the processing of a natural note returns a number of harmonics as well as the possibility of a missing peak which would represent the fundamental frequency as mentioned above. This required a completely new approach to be taken in terms of finding the pitch of a particular piece of audio. Instead of the original approach which found the highest peak in the power output of an FFT, and used the location of that peak within the window to find the fundamental frequency, which was the approach used for a lot of time at the start of the project, it was decided that a good approach to take in finding the correct note, would be to employ a library lookup system. This would store a library of power outputs corresponding to each of the individual notes for a particular instrument and compare them with the output of the current FFT window. In theory this would give the closest match in pitch to the

note being played. Crucially this removed the problem of retrieving notes of the wrong octave.

The next problem was that of the note lengths. This proved to be the most difficult of characteristics to process accurately. With the varying note lengths inherent in music, and with notes beginning before the previous has fully died out to silence, many different problems in the precise transcription of music are present. The key approach taken in the analysis of note lengths within this project is to measure the total power within each FFT window and compare it to the that of the previous window. This would allow for an attempt at estimating whether or not the sound is dying down or increasing and therefore assist in checking whether a note that was previously played is dying out, or whether or not the note has been newly struck.

A minor problem encountered during the transition phase from Matlab to Java development involved the use of the audio files generated by the matlab MusicMaker class. This allowed the developer to create pure sine wave tones of different lengths and frequencies to allow the accuracy of the system to be easily tested for a large range of inputs. The problem was that the .wav files being generated were not being successfully read within the Java system and causing a "file not found" exception to be raised. It was concluded that this was due to the way in which Matlab generates in .wav files. This issue was rectified by importing the piece into iTunes and performing the "Create Wav Version" operation on that file. This would then restructure the file, and therefore allow it to be read within the Java system. It is believed that the malformed piece of the file was in the header which is a core structural component of all wav files.

A problem outside the control of the developer involved the use of the abc4j library. Under certain conditions there were a number of issues that occurred depending on the size of the audio being transcribed. When only a very short test piece (< 15 notes long), this resulted in the generation of the sheet music, but if attempting to use the TunePlayer class included within the library, the audio wasn't being output. However, when transcribing a piece of audio greater than a certain length, the opposite occurred, with the audio output of the transcription being clearly audible, yet the graphical sheet music wasn't being shown. This was even after the corresponding .abc notation had been generated in a file for reading.

A last, but very crucial problem encountered in the development of the system was the management of library files. With any new instruments being added resulting in the addition of 96 new .wav files to represent all of the octaves, but there is then the generation of text files representing the power found in this files, resulting in the addition of 192 new files every time a new instrument needed to be added to the system as a possible result of the transcription process. Due to the hours of work the generation and processing of these files would take to be completed, it was decided to only use the central octaves for testing purposes, as these are the ones most commonly found in music.

Chapter 6: Testing and Evaluation

Extensive testing needed to be performed in order to find how the program handled input files of varying content, from both different instruments and different speeds of music. These input files were generated using a collection of programs such as Matlab, Audacity, iTunes and GarageBand. The following sections will give an overview of how this testing was performed, as well as giving the results for both the Matlab and Java implementations for certain criteria. These criteria include the factors of pitch detection accuracy as well as note length detection for different audio types. Due to the fact that the implementation of a multi-instrument transcription method was not fully implemented in Matlab and not implemented at all in Java, the results obtained for the matlab implementation are not representative of a fully developed multi-instrument algorithm, however, the fact that the process of determining note length is the same as with a single instrument, the accuracy of note length detection remained the same. The lack of a Java implementation for this process meant that no results could be obtained.

For the testing of the application's performance for pure tones was performed over a collection of musical pieces generated in Matlab. The collection comprised of about ten single note tracks, ten consecutive note tracks, and only two multi-instrument tracks due to the small amount of implementation time available for this. The same number of tracks were used in the testing of natural instruments, the only difference being the fact that these tracks were generated using GarageBand. Each of these tracks varied greatly in characteristics such as note length, song duration and type of music. The average duration of each piece of music needed to be below a length of approx. five minutes due to a limitation imposed by that of the abc4j library. The number of notes per transcription ranged from one note, up to a maximum of 200. The majority of the audio files used were those generated using GarageBand and matlab although there were a number of commercially available audio tracks used in the testing process.

6.1 Pure Sine Wave Tones

The testing and evaluation of the transcription of various sequences of sine wave tones was relatively straightforward in comparison to the valuation of the systems performance when handling natural instruments, but nonetheless required a lot factors to be taken into account. The generation of pure sine wave tones was performed in Matlab by implementing a "Music-Maker" class whose custom purpose was to create either a single note, or a series of notes. A small segment of code is given below which shows how, in this case, a sine wave tone corresponding to an A3 is generated.

The first test to be performed was the analysis of a single note which meant that an audio file with a single sine wave tone was input and analysed to check whether or not the program could correctly identify a single note. These tests were run for various sine wave tones of different pitch and note length. The algorithm, both in Matlab and Java returned the correct note with almost 100% efficiency. The reason for which pure sine wave tones are much easier to transcribe than those produced by natural instruments, is because there is no issue involving the lack of a fundamental frequency peak at the correct position. There

are also no harmonics within the generated tone meaning that there is a greatly reduced chance of the transcription returning the wrong note. As a result, given a set of fundamental frequencies, the frequency of the current window could be calculated and matched to the corresponding note.

For musical pieces using pure sine wave tones, the matlab algorithm was successful in transcribing the piece with 100% accuracy. This was a big step within the progress of the project, as this signified that it was time to begin development in Java. The below is a some output from the Matlab algorithm after transcribing the first 13 notes of the Irish tune “The Maid Behind the Bar”. For testing purposes, the length of the final note was doubled. The two figures below show the sheet music available online for the first section of this piece, and the program’s output after transcribing the pure sine wave version.

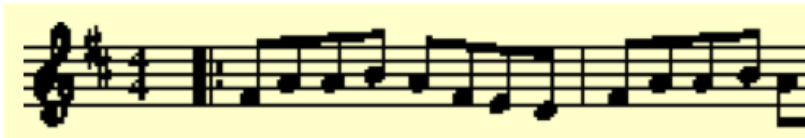


Figure 6.1: Sheet music for first 13 notes of “The Maid Behind the Bar”.

Full Tune: 1*Fsharp3 1*A3 1*A3 1*B3 1*A3 1*Fsharp3 1*E3 1*D3 1*Fsharp3 1*A3 1*A3 1*B3 2*A3

Figure 6.2: Transcription for first 13 notes of “The Maid Behind the Bar”.

The program output following transcription shows that it was executed perfectly resulting in not only the correct note order, but also accurately calculates their lengths. Below is some analysis of the transcription results providing an overview of both the Java and Matlab transcription process.

Table 6.1: Matlab Results for Pure Tones

Matlab Performance	Pitch Accuracy	Note Length Accuracy
Single Notes	100%	100%
Consecutive Notes	100%	100%
Simultaneous Notes	65%	100%

Table 6.2: Java Results for Pure Tones

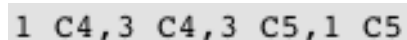
Java Performance	Pitch Accuracy	Note Length Accuracy
Single Notes	100%	100%
Consecutive Notes	95%	90%
Simultaneous Notes	N/A	N/A

6.2 Natural Instruments

The testing and evaluation of natural instrument note detection proved to be extremely challenging and required a lot of intensive testing. This was due to the large variance in instrument types, tunings, recording quality and the number of instruments. The main instrument used for testing was that of the piano, this was selected due to the fact that the sound of a piano note has a loud attack and a fast decay, helping in the separation of individual notes.

As with the testing of sine wave transcription, the first aim of the testing process was to obtain an accuracy reading for the transcription of a single note played by a natural instrument. This proved to be greatly more difficult than that of the sine wave tones as different instruments playing the same note can vary greatly, therefore the size of the note library can grow quickly due to the extra library entries required for each additional instrument to be tested, even different brands of the certain instrument causing a difference in transcription due to the overtones produced by the different instruments.

The first instrument to be tested was the piano, the notes of which were generated using the software GarageBand, with editing of the recorded notes being performed using Audacity to remove and silent sections of audio that are not required. This helps the transcription complete in a shorter time due to the removal of unnecessary transcription of silent audio. The final results for the transcription of a single note played by a natural instrument were extremely good. This was under the constraint that the library provided all of the note patterns necessary for an accurate output. The problematic area of a notes transcription was during the decay time, as the system had a tendency to interpret this section of the note's lifetime as being an octave above that of the test note. Below is some output from the transcription of a C4 piano note, showing the octave change as the note decays.



```
1 C4,3 C4,3 C5,1 C5
```

Figure 6.3: Output of a C4 single note transcription.

The number before each note represents the systems interpretation of how long the note lasts for, this is based on the energy of the audio within a given window relative to the windows surrounding it. The C4 note of length one is there due to the note being in the attack phase, therefore having a lower energy than that of the second window, interpreting that as a new note being played.

This pattern of transcription is persistent across all of the single note test files, and therefore this issue continues into the transcription of complete musical pieces. There were a number of different approaches that could be used to solve this issue. The approach chosen was that of finding consecutive notes that were the same but of a different octave, these were considered to be the same note, and were all assigned to be the lower of the octaves found. This approach was only used when a higher octave followed a lower one, if a lower octave was then found after a higher, this is considered to be the start of a new note. As with the pure tone analysis, the following tables give an overview of the accuracy obtained by each implementation.

Table 6.3: Java Results for Natural Instruments

Java Performance	Pitch Accuracy	Note Length Accuracy
Single Notes	100%	100%
Consecutive Notes	80%	85%
Simultaneous Notes	N/A	N/A

Table 6.4: Matlab Results for Natural Instruments

Matlab Performance	Pitch Accuracy	Note Length Accuracy
Single Notes	100%	95%
Consecutive Notes	100%	90%
Simultaneous Notes	50%	100%

The results given above for simultaneous note analysis were retrieved using only a very limited data set, therefore providing results that would not be as accurate given a larger dataset. The results also show the drop in accuracy given the transcription of consecutive notes due to the issue presented by analysing a window that would contain the end of one note and the start of another, therefore showing the need for the development of the multi-instrument algorithm previously mentioned.

6.3 Comparison of Java and Matlab Implementations

This section covers an evaluation of all results obtained during testing as well as comparisons between the Matlab and Java implementations for the different types of audio files tested. With the difference in language structure between Java and Matlab being taken into account, the difference in run-time was comparatively low with Matlab averaging a speed about half that of the Java implementation. This was calculated by transcribing the same tracks in both environments for a selection of audio files, and comparing the average run time for each.

There is also a great contrast in user interface as shown by the figures below. The Java GUI being used as the deployable application and the Matlab environment being used for development purposes.

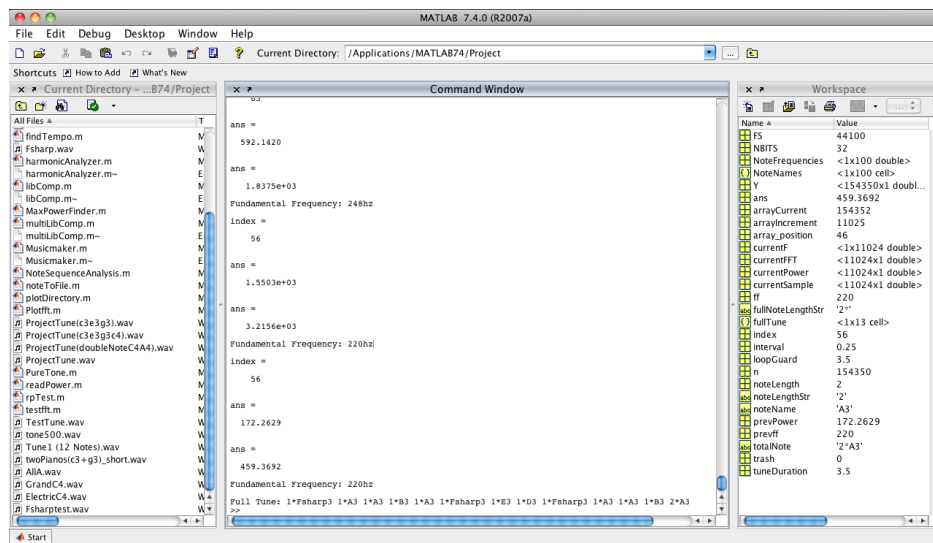


Figure 6.4: Matlab development environment and user interface.

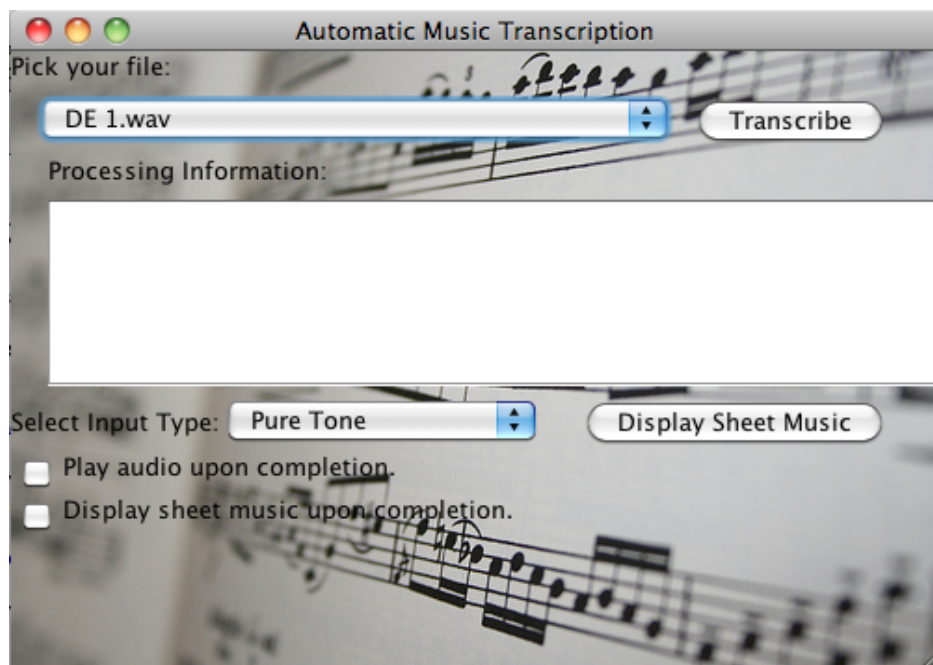


Figure 6.5: Java graphical user interface.

This allows for the use of the application by users who have no prior programming experience or are new to computers entirely.

6.3.1 Matlab Performance

The overall system transcription process was first implemented in Matlab and given that this would later be converted into Java, the performance of this implementation is very important in order to provide a benchmark to aim for when using Java. The testing performed, the results obtained, and the comparison of its accuracy against that of other systems and approaches that have been developed in the past shows that the system in question has performed to an adequate standard given the time constraints involved. The transcription of sine wave notes, both in the context of single and consecutive note processing, had a very high accuracy rating of $> 90\%$ allowing a lot of confidence in the robustness and reliability of the system. This high result applied to both the quality of the pitch detection, as well as the note length detection.

The speed at which Matlab processed the audio of completed the transcription was also extremely fast compared to the speed of the Java implementation. However due to the lack of a user friendly interface, Java was picked as the final development language.

6.3.2 Java Performance

The performance of the Java system was good for the transcription of pure tone audio files, although it wasn't as strong as the Matlab implementation, it performed to a high standard. However in strong contrast to the results found for the pure tone audio files was that of the transcription of natural instruments. This showed a high reduction in the overall accuracy of the system, due to the large variance in the way notes from a natural instrument can be played, and the wide range of instruments that can be used to produce these notes.

The transcriptions generated were made possible due to a large collection of files generated representing the FFT powers of certain notes, but due to the increasing size of the library as more notes are added, the speed of the system suffers due to the greater amount of files being processed. However the reliability of the natural instrument transcriptions lacked consistency as when notes began before the previous had fully died out, the harmonics from the two notes would result in some inaccuracies within the transcription.

Chapter 7: Conclusions and Future Work

The goal of this project was to implement an automatic music transcription system that can be used to produce the corresponding sheet music to allow for a musician to add to their repertoire easily. Due to the fact that this project involves both a high amount of research as well as implementation, a high number of factors needed to be taken into account.

The background research performed to date was highly beneficial in the development of a fast and accurate algorithm. It has helped with the selection of the correct methods of signal processing, as well as the key information related to each of these methods that help to increase the efficiency of the algorithm. It also gives insight into the best ways to interpret the information retrieved from the .wav files and how this can be examined to pick out the individual notes. This research has also provided an insight into the many problems that have been encountered in this field of computer science pertaining to the analysis of audio containing natural instruments.

The overall progress of the project was good, with all mandatory and discretionary requirements being met. However, the exceptional requirements which ask for the transcription of natural instruments resulted in an implementation that was not to the standard the developer was hoping for. With the large amount of work involving the transcription of simultaneous musical notes, very few of them being able to successfully perform accurate transcription, it was realised that the development of a system to achieve this was unfeasible in the scope of this project. The transcription of music containing natural instruments was therefore limited to those containing only one instrument. With this said, there were a number of attempts at implementing a system which would handle multi-instrument transcription. However this resulted in an accurate multi-instrument detection being unsuccessful. The table below represents the extent to which the requirements of the project were completed.

Table 7.1: Overview of Project Completion

	Complete	Not Complete
Mandatory		
Read an audio track in an uncompressed format. e.g. .wav	✓	
Analyse the audio track to determine the fundamental frequency of any musical notes	✓	
Convert the sequence of identified musical notes to abc notation	✓	
Save the abc notation in an ASCII file	✓	
Perform automatic music transcription for audio tracks containing a sequence of single sine wave tones	✓	
Determine the accuracy of the transcription	✓	
Discretionary		
Analyse the musical notes to determine their relative duration and include this information in the abc notation	✓	
Conversion of the abc notation to graphical (modern musical) notation using the abc4j library	✓	
Exceptional		
Improve the note recognition algorithm such that the program can accurately recognize notes from natural instruments.	✓	
Determine the accuracy of the new algorithm by assessing its performance over a number of pieces.	✓	
Extend the algorithm to work for multiple instruments and multiple notes being played at the same time.		✗

Due to the great amount of research that has been performed in the past relating to the accurate transcription of natural notes, it is quite evident that there is still a wide array of aspects that apply to this field that have not yet been explored within the scope of this project, or were simply eliminated as a feasible implementation approach due to their high complexity.

There is a large amount of future work that could be explored using the implementation and research acquired during this project's life-cycle. An increase in the library size is potentially one of the major factors that would be improved if more time was provided as the addition of more reference files to the library allows for an increase in accuracy, although the creation of a better file selection algorithm to make sure only the necessary files were processed would need to be implemented in order to keep run-time to a minimum. There is also the necessity for the implementation of the multi-instrument detection algorithm to allow for the system to correctly transcribe all forms of audio input. Once this extra work is fully implemented there could be a number of previously un-mentioned applications that the program could be used for such as using the fundamental frequency acquisition provided to help in other areas such as voice recognition and the removal of noise from audio files.

Bibliography

- [1] Tzanetakis, G., and Cook, P. : *Musical genre classification of audio signals*. Volume 10, 2002.
- [2] Smith, D.W. : *The Scientist and Engineer's Guide to Digital Signal Processing* ISBN 0-9660176-3-3, 1997
- [3] Brown, J.C., and Massachusetts Institute of Technology. Media Laboratory. Vision and Modeling Group : *Musical fundamental frequency tracking using a pattern recognition method*. 1992.
- [4] McNab, R.J., and Smith, L.A., and Witten, I.H., and Henderson, C.L., and Cunningham, S.J. : *Towards the digital music library: Tune retrieval from acoustic input*. 1996.
- [5] Klapuri, A.P. : *Multiple fundamental frequency estimation based on harmonicity and spectral smoothness*. Volume 11, 2004.
- [6] Davies, S.C., and Etter, D.M. : *An adaptive technique for automated recognition of musical tones*. 2002.
- [7] Cinnide, A., Dorran, D., Gainza, M., and Coyle, E. : *Linear Prediction: The Technique, Its Solution and Application to Speech*. 2008.
- [8] Monti, G., and Sandler, M. : *Automatic polyphonic piano note extraction using fuzzy logic in a blackboard system*. 2002.
- [9] Wise, J., and Caprio, J., and Parks, T. : *Maximum likelihood pitch estimation*. Volume 24, 2003.
- [10] Alonso, M., and David, B., and Richard, G. : *Tempo and beat estimation of musical signals*.
- [11] Bello, J.P., and Daudet, L., and Abdallah, S., and Duxbury, C., and Davies, M., and Sandler, M.B. : *Acoustics, Speech and Signal Processing, IEEE Transactions on*. Volume 13, 2005.
- [12] Duxbury, C., and Sandler, M., and Davies, M. : *A hybrid approach to musical onset detection..* 2002.
- [13] Creative Detectors, 2007. : <http://www.creativedetectors.com>
- [14] Gailius Raskinis, 2002. : <http://www.recognisoft.com>
- [15] Zenph Sound Innovations, Inc., 2002-2011 : <http://www.zenph.com/the-music.html>
- [16] The MathWorks, Inc., 1994-2011 : <http://www.mathworks.com/products/matlab/>
- [17] B. H. Suits, Physics Dept., MTU, 1998-2010.: <http://www.phy.mtu.edu/suits/notefreqs.html>
- [18] abc4j, 2011. : <http://code.google.com/p/abc4j/>
- [19] Piotr Wendykier, 2011. : <http://sites.google.com/site/piotrwendykier/software/jtransforms>
- [20] Chris Walshaw, 1995-2011. : <http://abcnotation.com/wiki/abc:standard:v2.0#introduction>