Monophonic sheet music transcription from audio: An automated approach

Kazimierz Wilowski
Supervisor: Dr Kasim Terzic

Abstract

For most of history, musical transcription, the act of creating sheet music from a recording or a live performance of a piece of music, has solely been the domain of trained musicians, however this is slowly becoming not the case. As digital technology has improved gradually over time, it has become possible to tackle the problem of transcription using computers rather than the ear of an experienced musician. This project investigates a computational approach to generating monophonic musical performances automatically.

Declaration

Contents

1 Introduction

Musical transcription is the process of creating a piece of sheet music which captures the notes played in an already existing performance or recording of a piece of music, similar to how linguistic transcription consists of writing down words corresponding to what a person says out loud. The act of transcribing has for centuries and until quite recently been an activity performed exclusively by human beings, however over the last several decades, advances in computer science have meant that computers can now be used to assist with and sometimes be chiefly responsible for generating transcriptions from musical recordings.

The problem of transcription is a very interesting one, as it is a problem which is not entirely objective (like the answer to the question: What is the loudest moment in this recording of a musical piece?) but also not entirely subjective (like the answer to the question: is this performance of a piece of music good?). Furthermore, humans who are good at performing transcriptions do not gain their skill by memorising a set of techniques and methods to translate what they hear onto the page but instead build an intuition over years of practice, meaning if one were to ask a skilled transcriber "how" they know what to write, it would be extremely difficult for them to explain their process in precise terms (like asking someone "how" they know what words correspond to the sounds someone is making when they speak). As a result, new techniques must be developed to be used by computers which bear little resemblance to the methods used by humans to accomplish the same feat.

This project focused on the task of transcribing excerpts of monophonic western tonal music using traditional western music notation, from recordings provided as audio recordings. We investigate how the problem can be broken down into a variety of smaller sub-problems which can then be solved individually and the results of which can be cumulatively combined to form a sensible transcription for a given excerpt. To accomplish this, we draw on and borrow techniques from a variety of domains, including but not limited to, signal processing, statistics, and function optimisation.

2 Context Survey

This section discusses, introduces, and contextualises key concepts relating to automated music transcription and computer aided music information retrieval (MIR) with respect to the goals of this project. Included in this section is an overview of currently existing music information retrieval tools, the state of contemporary and historical research in the field, and further discussion of other important concepts relevant to this project, such as an introduction to some of the standard algorithms and techniques which were utilised over the course of the project.

2.1 Overview of Music Information Retrieval (MIR) literature

The field of computational MIR is relatively small in the grander scope of computer science disciplines but is the subject of an increasingly broad corpus of literature. There are a variety of widely studied problems in the field of MIR, some of which have been studied at great length and for which there exist robust solutions and standard algorithms for tackling them, while others are still quite challenging for computers to perform with a practical rate of success. Examples of problems falling into the former would be that of monophonic pitch detection, for which there is a vast corpus of literature dating back to 1970s [RCRM76], and for which there are a plethora of widely implemented solutions which can be found in many (at least for a musician) everyday places, such as guitar tuner effects pedals [LS09], mobile phone apps, and digital audio workstation (DAW) plugins [You22]. Another widely studied problem is the problem of content-based searching, which allows the app to determine from a short, possibly distorted (by background noise, poor recording quality, etc.), the song being played[CVG$^+$08][Foo97], and provides the framework used by apps like Shazam. As indicated by the success of the Shazam app, it is a generally reliable and robust system. Other problems in the field of MIR are significantly more challenging, and consistently reliable solutions for them do not exist yet. D Byrd and T Crawford have highlighted a number of challenging MIR problems, such as the analysis of polyphonic excerpts [BC02], which currently still remains a difficult task, although modern deep learning and neural network based approaches to the problem indicate good progress is being made towards reliable system for polyphonic analysis [TA22] [Elo20].

Due to the challenges present in MIR technology, for certain applications and systems, human beings still outperform computers at analysing and annotating music with the relevant information, even when dealing with large volumes of music. For example the Pandora music streaming service, which places a focus on recommending users songs based on their listening habits uses a database of annotated songs called the *Music Genome Project*, however the annotating of these songs is performed by human beings, not automatically by a computer [Wal09].

The problem of automated monophonic transcription does not fall into the former, easier category of problems, although there exist standard approaches to some of the constituent sub-problems which automatic transcription can be broken down into (e.g. monophonic pitch detection), the task as a whole represents a relatively challenging one, since for a given musical excerpt, there is a great deal of information which must be derived to build a full transcription (pitch and onset of each note played, time signature, key signature, etc.). Although there are a significant number of papers which discuss automatic monophonic transcription [BMS00] [GBS12] [MS00], it is difficult to find papers which go as far to fully generate typeset sheet music, instead, these papers generally focus just on extracting the pitch information and the relevant note onset and offset times. Only a small number of papers tackle the more challenging problem of taking a full *audio-to-score* approach to the problem could be found [RPCZ18] [Ryy04], these papers take a more abstract approach to the problem using deep/machine learning techniques to train neural networks to tackle the problem. This can be contrasted to the approach taken within this project, which involved breaking the problem down into easier to manage sub-tasks which were simple enough that they could be modelled using standard computer science techniques.

2.2 Review of currently existing software

The task this project was concerned with - the task of generating a piece of sheet music corresponding to a recorded excerpt of a musical piece, can be seen as a specific version of the more general problem of how can one extract from an audio recording enough information for a human performer to then attempt to play the relevant extract themselves. The average recording of a musical performance contains a vast amount of information and can be analysed in many different ways, and a user's requirements and goals will affect what specific characteristic information relating to a musical recording they are seeking to retrieve and include in their recreated representation of the music. For example, a hobbyist guitarist may simply wish to know what chords are used in a song and approximately when the chords change, since knowing this is enough information for an amateur musician to play along to a recording, Whilst a researcher analysing west African percussion music may be keenly interested in accurate rhythmic transcription but not be too concerned with the melodic or harmonic content of the music they are analysing. As a result of this, there exists many different software tools available which could be considered to some degree as automatic transcription tools. In this section we discuss a selection of relevant available software which in some way or another can be seen as answering this more general problem of extracting enough information for a performer to attempt recreating the recording, and discuss how they are both similar and different from the artefact developed over the course of this project.

Digital Audio Workstations

Digital Audio Workstations (DAWs) are where many professional and hobbyist musicians alike now find themselves spending most of their creative time. Such software suites offer a large variety of tools to aid in the recording, creation and manipulation of audio. Many DAWs contain features for MIR, for example, Ableton Live will attempt to extract from an audio file the onset and pitches of notes played and output to the user a MIDI representation of this information [Abl22]. This is an implementation of the weaker version of transcription discussed in the previous section. Ableton also includes "Groove Detection" features which can be used to extract rhythmic information from recordings so as to more naturally quantize and warp them [Abl22]. This involves detecting things like the beats-per-minute of a piece of music and how "swung" the beats are. Most DAWs are not designed with sheet music as a central aspect of the system, so as a result, the MIR features found in DAWs do not align incredibly closely with the features we require. Unfortunately, most DAWs are proprietary software and there is little information available as to how these features are implemented.

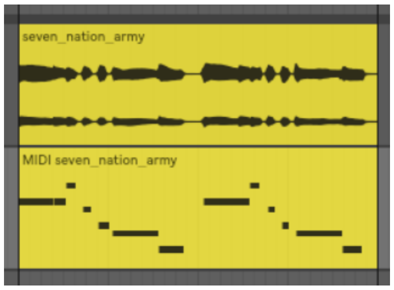


Figure 1: Ableton Live has functionality to extract MIDI information (below track) from audio clips (above track).

Other common DAWs such as Logic Pro, Cubase, and Protools contain similar functionality. None of these DAWs provide a full audio-to-score system.

Websites and mobile apps

There are many websites and mobile phone applications available which offer more niche and specific tools for performing MIR. As already mentioned, there are a plethora of guitar tuner apps and websites available [You22][Fen22][DaT22], which effectively perform real-time monophonic pitch detection. It is possible to find a selection of apps and websites offering functionality to accomplish a variety of other basic MIR tasks - for example detecting the BPM of a piece of music [Get22], or detecting the key of a piece of music [Tun22]. As was the case when looking at DAWs, these tools almost universally focus on a small MIR problem, which makes sense as websites or apps are generally designed for smaller, quick-to-perform tasks, this is in contrast to the goals of this project which seeks to provide a full audio-to-score system, the functionality required to accomplish some of the sub-tasks which must be solved as part of the creation of this system however is similar to the functionality provided by some of these apps and websites.

Automatic sheet music transcription

Finally we discuss currently existing software which attempts to perform a task similar to what we attempt to do in this paper - full audio-to-sheet transcription. Software such as MelodyScanner and AnthemScore attempt to generate sheet music based on live audio recordings, with the latter positioning itself as a composition assistant whereby the user can tinker and play with parameters in real time and can manually add and remove notes, change the piece's time signature or key signature, etc. It is difficult to provide an in depth discussion or evaluation of the functionality offered by these services since all examples of such software found placed their functionality behind a paywall or freemium business model. From the limited information which can otherwise be gleaned, it seems these pieces of software attempt to offer a system similar to that developed in this project, although they place a greater emphasis on user interfaces and human computer interaction, something which is not discussed in this project due to the limited scope available in a minor project. It is also difficult to compare these products to the system developed in this project due to the fact that all of these pieces of software are proprietary and not open source. Little information is made available on their websites concerning the inner workings of the system, and one is inclined to believe the vague allusions to "AI"s and "deep learning" made throughout their sites are more likely simply jargon in the name of marketing more than anything else.

2.3 libraries, formats, and resources

This section discusses a variety of standard MIR libraries and resources which were used or drawn on throughout this project, including file formats, standard Python libraries, and open source standard algorithm implementations. There exists a wide range of open source resources available for performing MIR. Use was made of a variety of these resources throughout the project. Compiled here is a list of these resources, including a discussion of the relevant functionality they provide.

- `MIDI`: No discussion of music related computing would be complete without some mention of the ubiquitous Musical Instrument Digital Interface (MIDI). Since the 80s MIDI has represented the de facto standard machine readable symbolic representation of music. Although MIDI was not as big a part of this project as originally anticipated, MIDI data was made use of when evaluating the artefact towards the end of the project. The standard MIDI pitch numbers were used throughout the project as a machine readable method of handling the pitches of notes

- `mido`: `mido` is a Python library which facilitates the straightforward reading and manipulating of MIDI information. `mido` proved to be a useful tool in extracting information from MIDI files used for evaluating the artefact, towards the end of the project.

- `MusicXML`: As the name suggests, `MusicXML` is an XML-based file format for storing western music notation. It a format supported by several of the other tools discussed in this section (Musescore, music21 - see below). While the functionality it provides is overkill for tasks involved in this project, it provides a good final format for musical transcriptions to be exported as, since transcriptions exported in this way can then be opened in a variety of music typesetting programs.

- `Lilypond`: `Lilypond` is an open source computer program for typesetting sheet music. It is a long running and well maintained project. Lilypond provides an easy to use system which produces high quality typesetting. As a result of being open source, many other resources available can interface with `Lilypond` (e.g. `music21` - see below) to provide high quality musical engravings with minimal low level input from the user

- `MuseScore`: `MuseScore` is a freely available and open source music engraving suite which provides support for `MusicXML` files. Similar to `Lilypond`, `MuseScore` provides a straightforward way of generating high quality sheet music with minimal effort. Additionally, `MuseScore` allows playback and editing of the typeset music. This provides some additional superfluous features to the system which are nonetheless interesting to have, for example a user can listen to the playback of a transcription or manually make changes to sections incorrectly transcribed by the system.

- `music21`: `music21` is a Python library and self-proclaimed "toolkit for computer-aided musicology" and provides a great many resources for manipulating symbolic representations of music in a standardised way, the scope of this project was such that only a small subset of `music21`'s features were relevant to the task at hand, including `music21`'s robust functionality for converting from symbolic, machine readable representations of musical information to more human readable formats with minimal time required to be spent typesetting. By building on the functionality of `lilypond` and supporting the ability to export information in `MusicXML`, `music21` provides means of producing machine readable music with little oversight from the user or developer.

- `aubio`: `aubio` is a long running, well featured open source library providing features and functionality for analysing and labelling audio signals, originally implemented in *C*, Python bindings are also available. It includes implementations of a variety of standard signal processing algorithms commonly used in MIR (including ones for pitch and onset detection for example).

It should be noted also that a variety of libraries not directly related to MIR were also used throughout the project, such as `numpy`, `scipy`, etc. More information on the use of these libraries can be found within the appropriate context later in this report.

2.4 Standard approaches and algorithms

In this section we discuss some well known approaches to standard problems which were encountered in the course of this project.

Maximum a-posteriori estimation

Maximum a-posteriori (MAP) estimation is a standard technique for estimating an unknown quantity within a statistical context. Certain problems in MIR can be modelled as statistical problems which this technique can be applied to. For example, the problem of quantizing note onsets from the continuous space of possible onset times to the discrete, quantized grid of a score can be viewed through a statistical lens. This is the approach taken to quantization by Cemgil et al. [CDK00], and we will use it as an example to outline how MAP estimations work. Firstly consider for a given quantized rhythm, that since humans cannot play perfectly in time, a human performer will vary from the quantized rhythm when they attempt to perform it by a certain amount. They are only likely to vary from the quantized rhythm so much, and in fact, we can consider the variance of their performance as being distributed randomly. For some performance A, we can allocate a likelihood that it was an attempt at performing the quantized rhythm B, this is $P(A|B)$. If we have some observed performance A and wish to find a good quantization B, we might choose our quantization to be B such that:

$$B_{ML} = \underset{B}{\operatorname{argmax}}\, P(A|B)$$

This is refered to as the *maximum likelihood* (ML) estimate, and can be imporoved if we consider what we know about the distribution of B itself, the so called prior, $P(B)$. In the context of quantization and sheet music, certain rhythmic phrases and durations are far more common than others, therefore it is possible that *prior* to knowing what the unquantized onsets are for a given performance, we can get some kind of estimate of how likely it is a score would contain a given rhythm. To get a new, hopefully stronger estimate, we can combine $P(B)$ and $P(A|B)$.

$$B_{MAP} = \underset{B}{\operatorname{argmax}}\, P(A|B)P(B)$$

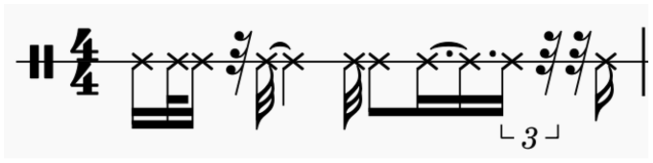
Note that Bayes' theorem states that:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

and since we are concerned with a specific unquantized rhythmic passage A, $P(A)$ is constant, so:

$$\frac{P(A|B)P(B)}{P(A)} \propto P(A|B)P(B)$$

So when we try to maximise the quantity $P(A|B)P(B)$, we are in fact also maximising $P(B|A)$, the *posterior* probability. Hence the name *maximum a posteriori* estimation. Later in the report we discuss in more detail how MAP estimation was used to implement quantization within the system.



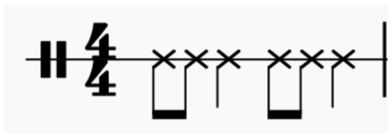


Figure 2: Above is a very unusual rhythm which even without knowing the unquantized onsets, we can say probably wouldn't be a sensible choice for quantizing any set of onsets to - this rhythm would have a low prior $P(B)$. Below is a more common rhythmic figure, this would have a higher $P(B)$.

Onset detection

A common signal process problem in MIR is that of detecting when the beginning, or onset, of a note is. Onset detection is a widely studied problem [Dix06][BDA$^+$05][Kla99], and there are a variety of reliable algorithms which exist for detecting onsets. In this section we outline how the *spectral flux* onset detection method works. It is a variant of the *spectral flux* method which is made use of later in the project for performing onset detection.

An onset detection function is a function whose peaks are intended to coincide with onsets of notes from an audio recording. A key idea utilised across many onset detection functions is that of the time-frequency representation of a signal. A Fourier transform can be used to determine the frequencies present in a signal but only over a given window of time, the larger the window of time, the more frequency information can be determined, but a sacrifice must be made in terms of the temporal resolution of the signal. Thus the time-frequency representation of a signal breaks the signal into discrete windows, the frequency distribution of each of which can then be analysed, and provides a balance between spectral and temporal resolution. A typical window size is 2048 samples [S06].

Spectral flux works by observing how the magnitude of different frequencies produced by Fourier analysis changes over time. If we consider the cumulative positive changes across all frequencies analysed by the Fourier transform, we have a function which will increase when the frequency content of the signal changes. This often indicates a note onset since new notes will often have different spectral content from previous notes, and also the onset of a note on most musical instruments has a significantly different spectral profile than the sustained part of the note (e.g. the pluck of a guitar string or bright "ping" of striking a glockenspiel).

pitch detection

Another MIR task overlapping with the field of signal processing, the extraction of musical pitch has a broad body of literature associated with it [RCRM76][DCK02][DLCMS01], and there are a plethora of standard algorithms and approaches to the problem. One of the most common and widely used fundamental frequency algorithms is the YIN algorithm, which uses a relatively involved process for determining fundamental frequencies. The first step of the process involves determining an amount we can shift the signal by such that it resembles itself closely again. This makes sense since if we are attempting to find the fundamental frequency of a signal, the signal will exhibit periodicity, further steps involve applying a variety of further signal processing techniques so as to deduce a relatively simple objective function which can then be optimised without simple standard techniques to give a final fundamental frequency estimate. This report made use of the YIN algorithm for performing pitch detection.

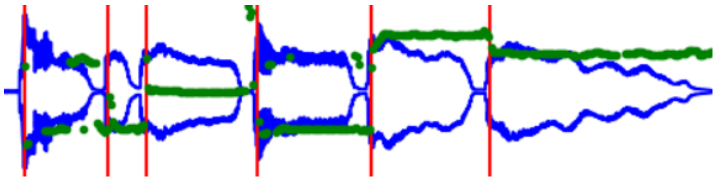


Figure 3: The waveform of an audio file (blue) with estimated onsets shown in red and estimated relative pitch shown in green (here and throughout, visualisations created using `matplotlib` show the maximum amplitude within consecutive blocks of 4096 samples), pitch estimate plotted with respect to MIDI note number and are shown vertically in line with the part of the waveform the estimate corresponds to. Excerpt is first two bars of *Happy Birthday*, played on Trumpet.

Salience

Salience refers to the perceived prominence of a note in a piece of music. A note with high salience is one that jumps out at the listener, that the listener percieves as more "important" than a note with low salience, which will draw less attention. Salience is determined by many factors (loudness, pitch, timbre, harmonic relation to key, etc.). Although there is a variety of literature published on the topic of musical salience, there is little consensus in it about a standard metric for measuring it, with a variety of models being proposed. Based on the available literature, two of the most widely considered measures of salience are found in [Dix01], and these were the measures considered later within this project. The first salience measure, which we will refer to as the *nonlinear* salience measure is given as:

$$S_{\text{nonlin}}(d, p, v) = d \cdot (c - p[p_{\min}, p_{\max}]) \cdot \log(v)$$

where c = 84 is a constant, d is the note's duration in ms, p is the MIDI note number associated with the note's pitch, and v is the velocity of the note. In the paper, v is defined as the MIDI velocity, but since we do not know that quantity, we take v to be the peak amplitude of the note normalised so that the note in the recording with largest peak amplitude has $v = 127$, the maximum MIDI velocity possible (i.e. a note with a peak amplitude half that of the maximum amplitude in the recording would have velocity $V = 63.5$). Finally, $p[p_{\min}, p_{\max}]$ is defined as:

$$p[p_{\min}, p_{\max}] = \begin{cases} p_{\min} & p \leq p_{\min} \\ p & p_{\min} < p < p_{\max} \\ p_{\max} & p_{\max} \leq p \end{cases}$$

The second salience measure used (the *linear salience)* is given as:

$$S_{\text{linear}}(d, p, v) = c_1 \cdot d + c_2 \cdot p[p_{\min}, p_{\max}]) + v$$

Where the symbols have the same meaning as in the definition of nonlinear salience, and the constants c_1, c_2 are given as $c_1 = 300$, $c_2 = -4$.

3 Requirements Specification

Below is the original set of goals listed in the project's DOER deliverable.

Primary	• Conversion from audio to MIDI data: Investigate algorithms and techniques capable of monophonic pitch, rhythm, and note duration detection from audio clips of a specific monophonic instrument playing a melody, and attempt an implementation of such an algorithm.
	• Investigate algorithms and techniques capable of reasonably estimating the tempo and time signature of a set of monophonic MIDI data, and attempt to implement such an algorithm
	• Investigate algorithms and techniques capable of reasonably estimating the key signature of a set of monophonic MIDI data, and attempt an implementation of such an algorithm
	• Implement a means of systematically typesetting and rendering monophonic MIDI data as sheet music, incorporating the deduced tempo, time signature, key signature (from Objectives 2 and 3), and the MIDI pitch, rhythm and note duration data (from Objective 1).
Secondary	• Investigate extending monophonic pitch and note detection to a larger pool of different instrument timbres
	• Investigate extending monophonic pitch and note detection to simple polyphonic recordings.
	• Investigate and implement means of detecting and analysing dynamical information (relative loudness/quietness) about notes in an audio clip, and include this information as dynamic queues (cruciendo, forte, etc.) in the rendered sheet music for the audio clip.

Once work was began on the project, after a relatively small amount of research had been done and time had been spent studying the topic, it was realised that the original goals developed did not necessarily align with what it would make sense for the project to involve. For example it quickly became apparent that it would be superfluous and add unnecessary overhead to use MIDI as an intermediate format for data. It was also realised that certain original goals grouped too many tasks under one heading. For example the second task groups tempo detection and time signature detection together, both of which it was soon realised were unique and challenging problems in their own right. As a result of these developments, after discussion with the project's supervisor, the goals of the project were slightly reevaluated to better align with what the project would come to be. Below are those reevaluated goals.

Primary	• Investigate and integrate algorithms and techniques capable of monophonic pitch, and onset detection from audio clips of a monophonic instrument playing a melody.
	• Investigate and integrate algorithms and techniques capable of reasonably estimating the tempo of extracted discrete notes, including controlling for the natural tempo drifts present in human performances.
	• Investigate algorithms and techniques capable of reasonably estimating the key signature of a set of discrete notes
	• Investigate algorithms and techniques capable of reasonably estimating the time signature of a set of discrete notes
	• Investigate and integrate means of systematically typesetting and rendering a computer representation of a collection of discrete notes as sheet music, incorporating the deduced tempo, time signature, key signature, and the pitch, rhythm and note duration data.
Secondary	• Investigate extending monophonic pitch and note detection to a larger pool of different instrument timbres
	• Investigate extending monophonic pitch and note detection to simple polyphonic recordings.
	• Investigate and implement means of detecting and analysing dynamical information (relative loudness/quietness) about notes in an audio clip, and include this information as dynamic queues (cruciendo, forte, etc.) in the rendered sheet music for the audio clip.

The project's secondary goals remained unchanged.

4 Design and implementation

This section discusses the approach which was taken in order to design and implement a system which could fulfil the requirements outlined in the previous section. The main task of generating a transcription from an input audio file was broken down into several smaller sub-tasks whereby the output of one sub-task becomes the input of another. A discussion of each of these sub-tasks is included along with discussion of how the larger problem was broken down and how each of the sub-tasks fit together to specify a solution to the original task. Details regarding the implementation for each sub-task are also included and discussed.

4.1 Breaking down the problem

Like when solving most complex problems, it was found to be advantageous to break the large problem of automatic monophonic transcription down into smaller, more manageable pieces. The following sub-tasks were determined as necessary steps required to reasonably construct a piece of sheet music from a given audio recording.

1. From the input audio file, the pitch and onset transient information should be extracted.
2. The transient and pitch information should be analysed to produce a discrete list of notes with well defined onsets and pitches.
3. From the derived list of notes, an estimation (or several candidates) of the tempo of the piece of music should be deduced.
4. Using the estimated tempo, the absolute timestamps of the piece's rhythmic pulse should be deduced. This process should take into account natural deviations in the tempo over time since such derivations are a natural part of music performed by human beings.
5. From the downbeat timestamps and the information already known about note onsets, the most appropriate duration to denote the notes in sheet music should be determined.
6. From the information known about the pitches present in the piece of music given, the key signature of the excerpt should be estimated
7. From the information known about the notes, the time signature of the piece of music should be estimated.
8. Using all the information derived from previous steps, the music should be typeset and exported in an appropriate format such as an image or in MusicXML.

Below is a figure illustrating how these separate tasks combine to form a full solution to the initial task.

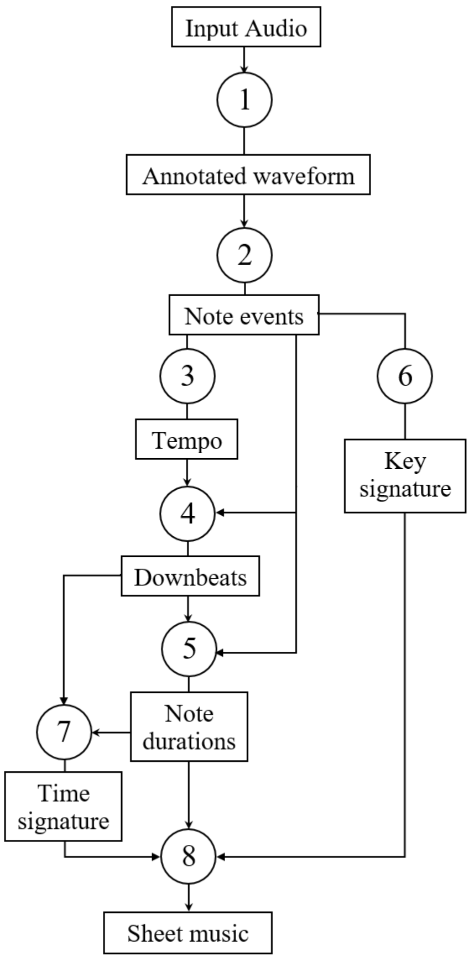


Figure 4: Rectangular nodes indicate input and output types and circular nodes indicate the index of the task being performed (as indexed in the numbered list on the page above).

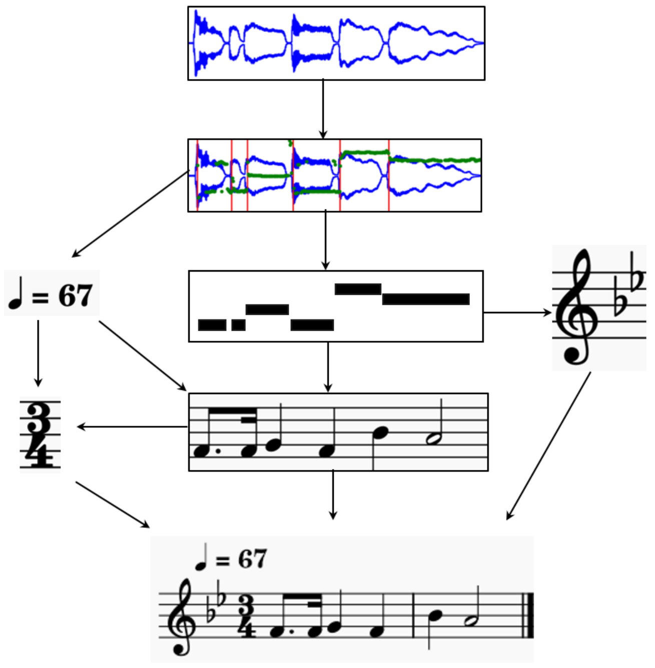


Figure 5: Flowchart showing visually the process of constructing sheet music from an input audio file.

It should be noted that almost all of these tasks could in their own right form the basis for a project in and of themselves, and so the final artefact developed over the course of this project does not purport to be a program encapsulating the most advanced and cutting-edge technology and research currently available with regard to each sub-task, but instead necessarily strikes a balance between quality and complexity necessitated by time constraints and the author's prior knowledge of relevant topics.

4.2 Restricting the problem

An important step in the design stage of this project was considering exactly what sort of recordings would be considered for transcribing. Already we have discussed that the primary focus of the project is on *monophonic* recordings of western music, recordings only containing a single melody line. Additional restrictions to the input set of the system allow us to develop a more robust and refined system without having to consider outlier cases. The restrictions were chosen so as to remove what would be considered as highly unusual or extreme musical performances. Firstly, the range of tempi considered was limited to the range 60-200bpm, this range contains the vast majority of western music, and pieces composed with tempi higher than 200bpm or lower than 60bpm can generally be notated at a tempi in this range (e.g. a 45bpm piece can be notated at 90bpm without much lost in terms of information, all note durations would just be doubled from the original notation).

A further restriction considered was that of time signatures. Technically, there is an unlimited number of possible time signatures, however in western music, the the majority of pieces are in either

$\frac{4}{4}$, $\frac{3}{4}$, or $\frac{6}{8}$. Other time signatures can be seen as falling into two further categories, the first is time signatures which very closely resemble the three already given, for example a $\frac{2}{4}$ march will still be easily readable to a musician, even if it is typeset in $\frac{4}{4}$, similarly a piece in $\frac{12}{8}$ can be notated in $\frac{6}{8}$ without much issue. Finally there are odd meters which are not directly related to any other time signature. Examples include $\frac{5}{4}$, $\frac{7}{8}$, $\frac{11}{8}$, etc. Such time signatures are exceedingly rare and so are not considered in this project, since realistically any implementation which tried to consider them would generate more false positives than it would ever come accross actual examples of music notated in these signatures. The final restriction we will discuss here relates to the pitch content of the recordings analysed. Pitches were restricted to pitches in the range of A0 to E7. Again, it is possible to record and perform music using pitches outside this range, however as with the case with unusual time signatures, it is overwhelmingly more likely that considering a wider range would detect false positives significantly more often than true positives.

There were other, more subtle assumptions made about the music being analysed which are discussed later in this report - in the relevant context.

4.3 General implementation approach

Below we discuss each sub-task and the design approach taken towards implementing them.

4.4 Pitch and onset detection

The process taken to extract the pitch and note onset information was relatively straightforward and not the main focus of work undertaken in this project. Since, as discussed in the context review at the start of this project, both these problems are extremely well studied [G$^+$03] [BKS12], and open-source implementations of the standard methods for solving them are available online, the decision was made to make use of these implementations rather than spending time implementing the same algorithms from scratch. Small tweaks were made to the algorithms used in determining the pitch and onset points of notes in order to tune the algorithms more finely to the specific problem at hand rather than the broadest general use case.

Given an input file, using these standard algorithms it was possible to generate a list of discrete note onset points and a continuous pitch estimation function which mapped a given point in time to a MIDI pitch and confidence level in that pitch estimation.

For pitch detection, the python library `aubio` was used. The library provides a Python binding for the `C Aubio` library, which provides a variety of functionality for manipulating and analysing musical signals. From aubio, the YIN fundamental frequency estimation algorithm was used. Taking a hop size of 512 samples allowed for a continuous pitch estimation with good resolution: A file samplerate of 48000Hz meant there were $48000 \div 512 = 93.75$ pitch and confidence estimates per second. This resolution is more than adequate for analysing music performed by humans since this rate is significantly higher than any practical bound on the frequency at which human beings can play discrete notes. A confidence cutoff of 0.8 (where the confidence was a number in the range $[0, 1]$) was found to provide a good balance between culling incorrect pitch estimates and leaving enough correct estimates to perform analysis on.

For onset detection, the script implemented by [BKS12], which provided python implementations for a variety of transient onset detection algorithms. The algorithm chosen was the *spectral flux log filtered* algorithm, a modified version of the spectral flux algorithm discussed earlier, which was found by [BKS12] to provide the overall best onset detection. Bock et al. modify the ordinary spectral flux onset detection method by applying a set of filters to the input signal (a filter bank). The filters are selected to filter out frequencies so as to leave the frequencies corresponding to the notes of western tonality more pronounced. This decreases the chance of false positives due to non-musical noise, and increases the likelihood of more subtle true positives being picked up in the system.

4.5 Determining discrete notes

This sub-task is concerned with determining when given notes start and end based on a continuous estimation of pitch, estimation confidence, and discrete note onset points. Two approaches were implemented, both with slightly different use cases.

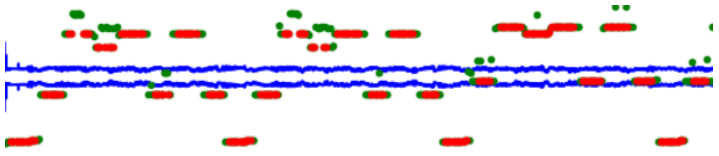


Figure 6: Pitch estimates attaining confidence threshold (red) can indicate the onsets of notes played on an instrument without obvious transients (note the consistent amplitude of the waveform). A visual representation of the waveform of the audio clip is shown in. Excerpt from *Bach: Cello Suite No 1 - Prelude*.

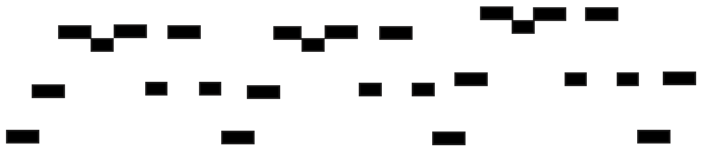


Figure 7: Symbolic representation of notes derived from Bach excerpt.

The first method implemented made use only of the pitch estimation data and was found to be more appropriate for instruments which lacked distinctive transients at the start of notes played (e.g. bowed string instruments like cellos and violins). The first method scanned through the input audio file from start to finish and used certain features of the pitch estimation function to determine when a note started and stopped. Firstly, any estimations of pitch outside of the restricted range mentioned earlier were not considered. As the program scanned through the time domain, a new note would begin (and the prior note end) when the pitch estimation changed and the estimation confidence attained a confidence threshold (see figure 6).

The information deduced for each note was used to instantiate a `Note` object which encapsulated the information needed to perform further relevant analysis. It was decided it would be beneficial to implement a custom class to represent notes within the system rather than using the already existing MIDI format for a couple of reasons. Firstly MIDI is very much designed as a system for encoding a continuous sequence of musical events, rather than discrete musical objects - there is no such thing as a MIDI note object, just `Note On` and `Note Off` signals defined relative to previous signals in the sequence. Secondly, implementing a customised class allowed functionality to be tailored to the task at hand. For example, different salience measures were defined as class methods, which made implementing latter parts of the system more straight forward and neater.

The second method used was more effective for transcribing notes played by instruments with well defined transients (e.g. piano, guitar). In this scenario, the onset transients already detected would likely represent quite a faithful representation of the actual note onsets present in the recording. Each pair of transients would then represent a note, the pitch of which would be determined by whichever pitch was most frequently estimated and which attained the confidence threshold within the two transients.

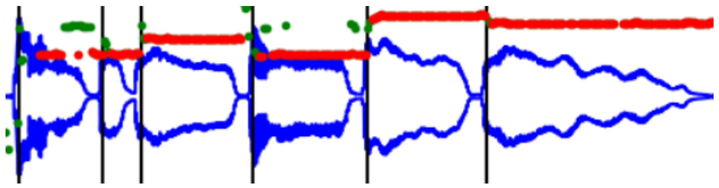


Figure 8: Transients (vertical lines in black) help more clearly delineate notes played on instruments with clear transients. Excerpt is first two bars of *Happy Birthday*, played on Trumpet.

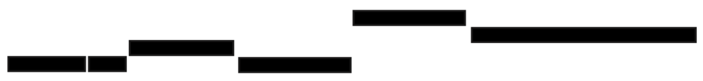


Figure 9: Symbolic representation of notes derived from *Happy Birthday* excerpt. The transients help us to deliniate between the first two notes of the same pitch and help us to register the start of the first note closer to when it was actually played.

It should be noted that for the most part, the end of one note was simply taken to be the start of the next, further metrics for determining when a note ends were not considered in great detail in this project, except for the final note in a recording where there would be no following note to mark its end. In this scenario, the end of the note was simply taken to be the last point in time where the pitch detection function estimated a pitch matching the final note's pitch with confidence attaining the confidence threshold.

Below can be found pseudocode outlining the two note detection methods which were implemented.

Algorithm 1 Non-transient note detection method

```
 1: procedure GetNotesNonTransient(confidence, pitches, times, threshold)
 2:     $i = 0$
 3:     notes = [ ]
 4:     while $i <$ len(confidences) do:
 5:         if not currently recording note and confidences[$i$] >threshold then
 6:             create note $N$
 7:             $N$.start = times[i]
 8:             $N$.pitch = pitches[i]
 9:             while confidence[i] >threshold and pitches[i] = $N$.pitch do
10:                 i = i + 1
11:             end while
12:             $N$.end = times[i]
13:             note.add($N$)
14:         end if
15:     end while
16: end procedure
```

Algorithm 2 Transient based note detection method

```
 1: procedure GetNotesNonTransient(onsets, confidence, pitches, times, threshold)
 2:     for $O_i$ in onsets do
 3:         create note $N$
 4:         find $j$ such that times[j] = $O_i$
 5:         find $k$ such that times[k] = $O_{i+1}$
 6:         $N$.start = times[j]
 7:         $N$.end = times[k]
 8:         $N$.pitch = mode of pitches[j to k]
 9:     end for
10: end procedure
```

4.6 Tempo estimation

In order to translate the absolute onset information known about notes in the recording to actual sheet music, it is extremely important to determine the tempo of the overall piece in question. For example, we may know that a note lasts 700ms and starts 3500ms after the beginning of the recording, however if we don't know the frequency of the pulse of the excerpt, it is impossible to notate the note using standard sheet music, which does not deal with absolute durations, but instead durations relative to the pulse frequency of the piece (i.e. the tempo).

Two methods were explored for finding the tempo of the excerpt. The first method will be referred to as the "naive" method, as it stood out as a relatively obvious but not incredibly robust or efficient way of determining the tempo of a piece of music. The second method investigated was proposed by Dixon [Dix01] and provides an interesting alternative approach to tempo estimation.

Naive method

Firstly we will discuss the naive method. In essence, the method consists of transforming the problem into an optimisation problem by defining a way of scoring different tempo candidates such that we can apply standard techniques of optimisation to find an satisfactory tempo. In this scenario, a tempo has two characteristics, the beat duration, i.e. the length of time between successive pulses in the piece of music, and an offset, to handle the scenario where the first note in the excerpt is not on a down beat (an anacrusis). While the range of the beat interval is continuous, the offset is quantised with respect to the beat duration. This is because in western notation, there are only certain note durations with respect to the beat duration which can be easily notated, so only these subdivisions of the beat must be considered as offsets. Tempo candidates are scored based on how closely they line up with the start of notes in the excerpt.

To define an optimisation problem, we must define an objective function to optimise. The objective function chosen for this optimization took as dependent variables the tempo candidate (i.e. the distance between pulses, and a starting offset), and calculated a score for each note based on the distance between the note and the nearest pulse predicted by the tempo candidate. The sum of these scores was taken as the score of the candidate tempo, with a smaller score indicating a better candidate. The objective function can thus be given as:

$$F(\Delta, \varepsilon) = \sum_{i=0}^{\#\text{ of notes}} \min_n E(O_i, n\Delta + \varepsilon)$$

Where Δ is the distance between pulses, ε is the offset, O_i is the onset of the ith note, and E is the score given for each note, defined as:

$$E(O_i, \Delta') = \sqrt{\frac{|O_i - \Delta'|}{D_i}}$$

Where D_i is the duration of note i. This expression punishes longer notes which are out of time less than shorter ones, and is non-linear such that the larger the difference between the actual onset and the tempo pulse the greater the punishment. To find the minimum value of the objective function, the `scipy.optimize.differential_evolution` optimisation function was used. This optimisation algorithm was chosen since it uses stochastic methods rather than gradients, which can prove to be more effective when dealing with more unusual objective functions such as the one defined in this problem, which we have no reason to believe will necessarily be smooth and well behaved.

Differential evolution does not make use of the objective function's gradient, instead, the method works by generating a random population of candidate "agents" spread over the search space, in this scenario, this would involve generating a set of candidate tempi with beat intervals in the range [300ms, 1000ms]. Agents are then moved around using a semi-random method. By picking a specific agent $\mathbf{X}$ and a set of other discrete agents, we can define a new agent $\mathbf{Y}$ in terms of $\mathbf{X}$ and the other agents we chose. If the score of $\mathbf{Y}$ is better than the score of $\mathbf{X}$, the agent $\mathbf{X}$ is replaced by the new agent $\mathbf{Y}$. This process is repeated until some termination condition is met (e.g. maximum number of iterations, a bound is achieved, etc.). Since agents will only be replaced by agents with better scores than them, over time, the average score of the population should decrease and tend towards a global minimum of the objective function. The objective function F defined above takes two parameters, however since as already discussed, the offset parameter ε is quantized to a small list of possible values, instead of applying two-dimensional differential evolution over both parameters, one-dimensional differential evolution was performed to determine the optimum beat interval for each offset, and the minimum across all offsets was chosen by brute force. i.e.

$$\operatorname*{argmin}_{(\Delta,\varepsilon)} F(\Delta, \varepsilon) = \operatorname*{argmin}_{(\Delta,\varepsilon)}(\min_{\varepsilon}(\min_{\Delta}{}_{\text{DE}} F(\Delta, \varepsilon)))$$

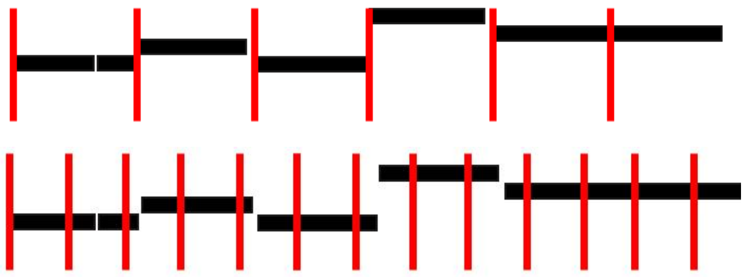


Figure 10: The pulse (shown in red) of a good tempo estimate (above) will coincide closely with the start of the notes detected previously, while a bad (below) tempo estimation will rarely line up closely with the start of notes.

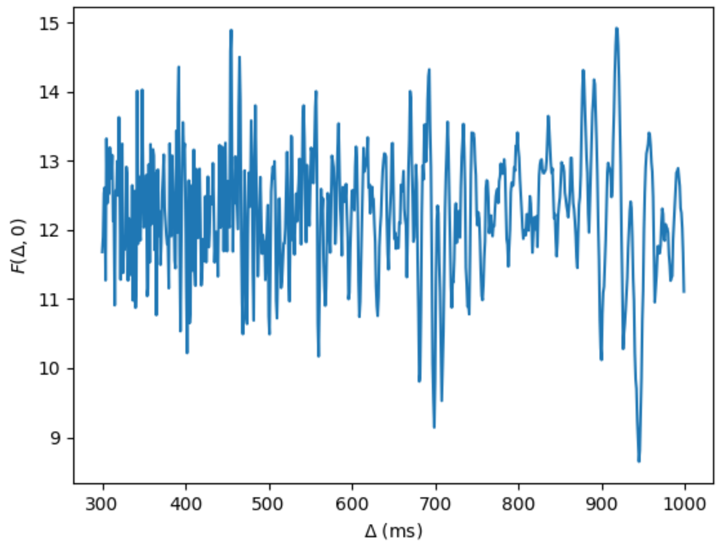


Figure 11: The value of the objective function F can oscillate wildly with respect to Δ, making it difficult to determine global minima using optimisation methods which utilise the function's gradient. The figure shows a plot of $F(\Delta, 0)$ with respect to Δ for the detected onsets of Happy Birthday.

Dixon method

The method outlined by Dixon does not transform the problem into an optimisation problem as such. Dixon's algorithm outlines a more deliberate and measured approach. Effectively, the algorithm determines a selection of reasonable tempo estimates by considering the gaps between the starts of notes, so-called "inter-onset intervals" (IOIs), by considering how often relatively similar IOIs occur and how different IOIs subdivide each other, we can deduce information about the overall metrical structure of the excerpt in question. More explicitly, IOIs of the same size (within a given tolerance, taken to be 50ms) are grouped into "clusters". Once each IOI has been placed in a cluster, clusters with similar average IOI size (again similar is taken to mean within 50ms) are merged together. After all the clusters have been created, clusters are scored based on how well smaller clusters "subdivide" into them. Clusters are scored via the following system.

Algorithm 3 Cluster score

```
procedure SCORE($C$)
    for each $C_i$ do
        if $|n \cdot C_i$.interval- $C$.interval$| < 50$ms then
            $C$.score = $C$.score + $f(n) \cdot C_i$.size
        end if
    end for
end procedure
```

Where f is given by:

$$f(n) = \begin{cases} 6 - n & 1 \leq n \leq 4 \\ 1 & 5 \leq n \leq 8 \\ 0 & \text{otherwise} \end{cases}$$

The best candidates are then chosen as the clusters with the highest scores.

$$C_{\text{best}} = \underset{C}{\operatorname{argmax}} \, \textsc{Score}(C)$$

The implementation outlined in the paper given was modified so as to restrict the range of possible candidate tempos, since it was found that large multiples of the correct tempo would be disproportionately favoured, so much so that the highest ranked candidate tempos would often be tempos closer to the length of a whole bar rather than the single note tempo. Tempos were restricted to the range 60-200bpm. It was considered to not be an acceptable consequence of this that pieces with tempos outside of this range then tended to have their tempo estimated as a simple integer multiple of their actual tempo (e.g. a piece at 45bpm might be estimated at 90bpm, and a piece at 280 might be estimated at 140bpm).

An object oriented approach was taken to implementing this algorithm, defining a class based on the clusters described by Dixon, which were then manipulated as outlined above.

4.7 Tempo tracking

A key characteristic of music played by human beings is that the exact tempo the music is performed at will vary by small amounts over time. Sometimes this is used for dramatic or emotive effect (so-called "rubato" in classical music) or sometimes it is unintentional. The fact of the matter however is that almost no human being can play for more than an extremely short amount of time without drifting slightly from the original tempo without something like a metronome keeping them in time. For this reason, to determine where the pulse of a musical excerpt lies it is not enough just to find one pulse duration, since over a relatively small amount of time, a consistent pulse will drift out of time from a human performer. Instead we must be able to adjust the tempo subtly over time to stay in sync with the performer.

We use the method outlined by S Dixon (2010) in the same paper of theirs which discussed tempo estimation. Their method involves creating "agents" who traverse through the notes in steps proportional to internal metronomes they possess. Agents who drift completely out of sync with the note onsets are ignored completely but agents who land close to onsets adjust their internal metronomes so as to try to stay better in sync. Scores are allocated to agents based on the size of the errors between pulse estimations and note onsets, the salience of notes is also incorporated into the agent's scores, so agents which hit the onsets of more salient notes are rewarded. The highest scoring agent should traverse a path consisting of pulse beats synced well with the note onsets present.

Several modifications were also made to the implementation outlined in the paper. Firstly, agents whose associated tempo increased by too extreme an amount would be removed from the list of potential agents. It was observed that agents had the tendency to increase their tempo so much that they effectively "hit" every note onset, however the pulses generated by such agents clearly did not represent the actual tempo of the piece, and often would involve the agent's tempo doubling or tripling. Agents whose tempos changed to be more than twice as fast as the original tempo or less than half as fast were removed.

Another modification made was to limit the total number of agents which were active at any time, since for pieces longer than a few seconds, the number of agents was found to increase exponentially, and after a short while, the program ran too slow to be useful. So the number of agents which could simultaneously be active was capped at 50. If a new agent was to be added, it would only be added if there were less than 50 active agents, or if it had a score higher than the 50th current agent (in which case it would replace the "worst" of the 50 already existing agents).

Below is pseudocode outlining the modified tempo tracking system which was implemented. To initialise the algorithm, agents with internal metronomes set to the top three tempo candidates from the previous step are assigned to each of the first five note onsets in the piece. At the start of the algorithm, there are therefore 15 active agents.

Algorithm 4 Tempo tracking algorithm

```
1: procedure TrackTempo(Notes)
2:     for each note $N_i$ do
3:         for each agent $A_j$ do
4:             if $N_i$.start - $A_j$.history.last >timeout then
5:                 remove agent $A_j$
6:             else if $0.5 \cdot N_i$.original_tempo <$N_i$.tempo <$2 \cdot N_i$.original_tempo then
7:                 remove agent $A_j$
8:             else
9:                 counter = 0
10:                while $A_j$.prediction <$N_i$ - $A_j$.pre_tolerance do
11:                    $A_j$.prediction = $A_j$.prediction + $A_j$.tempo
12:                    counter = counter + 1
13:                end while
14:                if $N_i$.onset - $A_j$.pre_tolerance <$A_j$.prediction <$N_i$.onset + $A_j$.post_tolerance then
15:                    if |$N_i$.onset - $A_j$.prediction| >inner_tolerance then
16:                        if # agents <50 then
17:                            duplicate $A_j$
18:                        else if # agents = 50 and $A_j$.score >$A_{\text{worst}}$.score then
19:                            duplicate $A_j$ and replace $A_{\text{worst}}$
20:                        end if
21:                    end if
22:                    Error = $N_i$.onset - $A_j$.prediction
23:                    Relative_error = Error/($A_j$.pre_tolerance + $A_j$.post_tolerance)
24:                    $A_j$.tempo = $A_j$.tempo + Error/counter
25:                    $A_j$.prediction = $N_i$.onset + $A_j$.tempo
26:                    $A_j$.history = $A_j$.history + $N_i$
27:                    $A_j$.score = $A_j$.score + (1-Relative_error) $\cdot N_i$.salience
28:                end if
29:            end if
30:        end for
31:        Add newly created agents
32:        Delete agents whose tempo and prediction too close
33:    end for
34:    $A_{\text{best}} = \underset{A}{\operatorname{argmax}}\, A$.score
35: end procedure
```

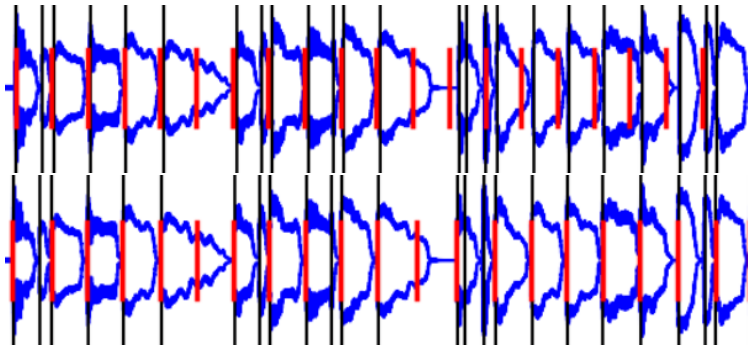


Figure 12: Above are two copies of the wavelength of a musical excerpt shown with two pulse schemas (shown in red) generated by the implemented system. Both schemas start with the same pulse size but only one has pulse tracking enabled. Above, after 10 or so notes, the tempo pulses (red) become out of sync with the note onsets (in black). The second is adjusted using the process outlined above, and as can be seen, stays in sync with the note onsets the whole way through. Excerpt is first eight bars of *Happy Birthday*, played on Trumpet.

4.8 Quantization

Once the pulse of the excerpt has been determined, it becomes possible to convert the absolute timing information deduced about the notes in the excerpt to the relative timing information required to notate it as sheet music. In western notation, note durations are always given in terms of some relatively simple fraction of the pulse length (e.g. there are exactly two eighth notes in one beat interval, an eighth note triplet has duration equal to one third of the pulse length, etc.). The process of converting from absolute duration to these discrete fractional relative durations is called quantisation. Two approaches were investigated for performing this task. Like with tempo estimation, the first method will be referred to as the "naive" method as it takes a relatively straight-forward approach to the problem but its simplicity leads to certain disadvantages. The naive method effectively consists of subdividing the pulse schema into discrete intervals and "snapping" the note onsets to the grid point they are closed to. By choosing a variety of different subdivisions and scoring them based on certain criteria, the "best" quantisation can be found. Details of how different grid subdivisions. An issue which becomes clear when using this naive approach is that the quantisations this method produce will often be "accurate" but will look odd to the human eye, especially for someone who has experience reading sheet music. The biggest challenge when quantising notes is to strike a balance between accuracy and human readability, which is what the second method investigated attempts more systematically to do.

The first task both quantization methods must do is split the list of unquantized onsets into a discrete list of sections based on where the tracked pulses in the system lie. The notes within each section can then be quantized and recombined to give a complete sequence of quantized notes.

Maximum a-posteriori (MAP) approach

We have already discussed the basic logic underlying MAP estimation. In order to fully implement a MAP solution to the problem of quantization, it is necessary to more precisely define the statistical distributions we are concerned with. In this scenario, it is impossible to empirically gather the information about the distributions involved in our problem by searching in the real world. Instead we must make assumptions about these distributions and how they behave. Firstly, let us break down exactly what it is we need to more precisely define.

The prior $p(\mathbf{c})$ - The prior indicates how "likely" it is that our quantisation c would actually appear in a written score and can be seen as a quantification of how complex the quantization we choose is. We define the formula for the prior as:

$$p(\mathbf{c}) = \sum_{S} p(\mathbf{c}|S)p(S)$$

Which follows from the *law of total probability*, where S is a subdivision schema for our given interval. A subdivision schema is a list of small primes which we use to subdivide the interval being quantized into a discrete grid. In our implementation, we use two subdivision schemas, $S_1 = [2, 2, 2], S_2 = [2, 3]$. The prior of each schema is defined as:

$$p(S) = \exp(-\zeta \sum_i w(s_i))$$

Where w is a simple weighting function defined as $w(s) = 0$ if $s = 2$ and $w(s) = 1$ if $s = 3$. And $p(\mathbf{c}|S)$ is defined as:

$$p(\mathbf{c}|S) = \exp(-\gamma \textsc{Depth}(c, S))$$

Where $\textsc{Depth}(\mathbf{c}, S)$ defines the depth of $\mathbf{c}$ with respect to the division schema S. The depth is calculated using the following algorithm.

Algorithm 5 Depth algorithm

```
1:  procedure DEPTH(c, S)
2:      iterations = 0
3:      depth = 0
4:      prime_product = 1
5:      for $s_i$ in S do
6:          iteration = iteration + 1
7:          prime_product = prime_product $*s_i$
8:          for $i \in \{0, 1, ..\text{prime\_product} + 1\}$ do
9:              for $c_j$ in c do
10:                 if $\exists n : c_j * n = i/\text{prime\_product}$ then
11:                     depth = depth + iteration
12:                     remove $c_j$ from c
13:                 end if
14:             end for
15:         end for
16:     end for
17:     depth = depth + (iterations + 1) * # onsets remaining in c
18: return depth
19: end procedure
```

Combining what we have outlined here allows us to determine the prior for a given quantization, $p(\mathbf{c})$.

The conditional probability $p(\mathbf{t}|\mathbf{c})$ - The conditional probability $p(\mathbf{t}|\mathbf{c})$ can be seen as a measure of how closely the quantization $\mathbf{c}$ matches $\mathbf{t}$. We will assume that the fluctuations from the true pulse in the unquantized onsets are caused by a normal distribution with average zero, meaning for the case where we have just one point, $p(t|c)$ can be taken to be:

$$p(t|c) = \frac{1}{\sqrt{2\pi\delta}} \exp\left(- \frac{(t-c)}{2\delta^2} \right)$$

However since within a given interval there will be multiple onsets, and the errors $|t_i - c_i|$ will be covariant, we must generalise our normal distribution to the case where $\mathbf{c}$ and $\mathbf{t}$ are vectors. We take the covariance matrix to be:

$$\Sigma = \delta^2 \begin{pmatrix} 1 & \rho_{1,2} & \cdots & \rho_{1,k} \\ \rho_{1,2} & 1 & \rho_{n,m} & \vdots \\ \vdots & \rho_{n,m} & \ddots & \vdots \\ \rho_{1,k} & \dots & \dots & 1 \end{pmatrix}$$

Where

$$\rho_{n,m} = \eta(\frac{\lambda^2}{2}c_n - c_m)^2)$$

This then allows us to generalise our normal distribution to the case where we have v, s as vectors. In that case:

$$p(\mathbf{t}|\mathbf{c}) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}}\exp\Big(-\frac{1}{2}(\mathbf{t}-\mathbf{c})^T\Sigma^{-1}(\mathbf{t}-\mathbf{c})\Big)$$

We have now seen how it is possible to computer $p(\mathbf{c}|\mathbf{t})$ and $p(\mathbf{c})$, thus these two parts can be combined to allow us to find the posterior probability $p(\mathbf{t}|\mathbf{c})$. To find our optimum quantization, it is now a case of finding $\underset{\mathbf{c}}{\operatorname{argmax}}\, p(\mathbf{t}|\mathbf{c})$.

It is not feasible to perform this search over the space of all quantizations, as this is an extremely large set. Instead we narrow our search The algorithm quantises the notes in the input by breaking down the input rhythm into sections of a given size based on the tempo pulse derived from the tempo tracking part of the system. For each section, the interval is then subdivided into a grid using two different schemas (one of which allows for the possibility of triplet rhythms occurring). For the first note onset in the unquantized rhythm, we find the n grid points it is closest to ($n = 2$ proved a satisfactory value), each of the n gridpoints chosen represent the first quantized onset in a quantization candidate. For each of the other unquantized notes chosen, we iterate through all the candidates we have we create n copies, and add one of the n gridpoints closest to the note's onset. Since we know the rhythm represents a monophonic melody, we consider the n closest grid points *not "already taken"* by another note - since two notes shouldn't be mapped to the same grid point as that implies polyphony. Once this has been done for all notes in the interval, we have a set of quantization candidates which is large enough to include all "reasonable" candidates which we should consider, but small enough to be analysed in a reasonable amount of time. Once these candidates have been found, it is simply a case of using the MAP approach to score each one and then choose the one with the best score. A variety of parameters are employed by the MAP based algorithm. The parameters used were those found to be optimal in the paper outlining the method [CDK00], with the exception of γ. A high γ can be seen as indicating preference for simpler quantizations, and it was found that increasing the value from that given in the paper to $\gamma = 1.0$ provided more appropriate quantization results. The other parameter values were: $\zeta = 1.43$, $\eta = 0.83$, $\lambda = 3.07$, and $\delta = 0.053$

Naive method

The naive method takes the approach of simply "snapping" each note to the closest grid point for a variety of grid resolutions, and then scoring each possible quantization based on how closely it matches the unquantized rhythm. This method is susceptible to prioritising accuracy by choosing very complex notations so an effort is made to balance the effects of this by penalising the use of grids with higher resolutions, so as to attempt to balance out the algorithms weighting of accuracy versus complexity. This is done by taking the sum of the distance between each of the quantized onsets and their unquantized counterparts, and multiplying it by the number of gridpoints in the grid chosen for example, if the interval was split into 16 subdivisions, then the final score would be multiplied by 16.

For a given rhythmic block (blocks of size 2 beats were considered to be a good size), we overlay a grid on the box. We choose grids containing a number of grid points which is the product of a small set of small primary numbers (i.e. 2 and 3), so we experiment with block size 4 (primes: $(2, 2)$), 12 (primes $(2, 2, 3)$, 3 (primes: (3)), etc. To generate the quantization associated with the grid, we "snap" each onset to the closest grid point, assuming there is no onset already snapped there.

Algorithm 6 Naive quantization

```
1: procedure QUANTIZE(onset, grid_resolution)
2:     quantized_onsets = [ ]
3:     for each onset $O_i$ do
4:         find $\underset{n}{\text{argmin}}(|O_i - (n/\text{grid\_resolution})|)$ where $n$/grid_resolution not in quantized_onsets
5:         quantized_onsets.add($n$/grid_resolution)
6:     end for
7: end procedure
```

We then score a given quantization as:

Algorithm 7 Naive quantization scoring

```
1: procedure SCOREQUANTIZE(onset, grid_resolution)
2:     quantized_onsets = QUANTIZE(onsets, grid_resolution)
3:     score = 0
4:     for $i \in \{0, 1, ...\texttt{len}(\text{onsets})\}$ do
5:         score = score + |onsets[i] - quantized_onsets[i]|
6:     end for
7:     score = score $*$ grid_resolution
8: end procedure
```

So the to find the optimal quantization $\mathbf{O}_q$ for a list of onsets $\mathbf{O}$ using the naive method, we find:

$$\mathbf{O}_q = \text{QUANTIZE}\big(\mathbf{O}, \underset{n}{\text{argmin}}(\text{SCOREQUANTIZE}(\mathbf{O}, n))\big)$$

Where $n \in \{1, 2, 3, 4, 6, 8, 12, 16\}$.

4.9 Determining the key signature

Although there do exist algorithms and methods for estimating the key signature of pieces of music which make use of considerably advanced techniques from statistics, machine learning, etc. [Tem02] [NLAF19] it was found that a relatively simple approach to determining the key signature of the piece of music in question was more than sufficient for the purposes of this project. The more advanced methods are often employed to determine the key signature of an excerpt directly from the source audio, however having already derived the notes present in the excerpt, the task is a lot easier.

We consider twelve standard key signatures in western music (we do not consider enharmonically equivalent keys), and each key signature corresponds to a major (and relative minor) scale which contains seven notes from the standard twelve notes of the octave. (e.g. the key signature F corresponds to the F major scale which contains the notes F, G, A, B♭, C, D, E). Each key signature is uniquely determined by their seven constituent notes. The key signature of the piece of music in question was determined by counting how many of the notes in the excerpt "belonged" to each of the twelve standard musical key signatures (e.g. B♭ belongs to the key of F, but G♯ does not). The "correct" key signature was taken simply to be the one which fitted the most notes.

The only non-trivial difficulty encountered in this section was as a result of enharmonics: musical notes which are notated differently but which sound the same, such as the notes A♯ and B♭. The original implementation of the algorithm used note names as the basis for the pattern matching approach taken, however this had to be replaced with using MIDI note numbers so as to avoid problems stemming from the occurrence of enharmonics.

Algorithm 8 Key signature algorithm

```
1: procedure KeySignature(Notes)
2:     for each note $N_i$ do
3:         for each key signature $K_j$ do
4:             if $N_i$.pitch $\in K_j$.notes then
5:                 $K_j$.score = $K_j$.score + 1
6:             end if
7:         end for
8:     end for
9:     $K_{\text{best}} = \underset{K}{\operatorname{argmax}} K$.score
10: end procedure
```

4.10 Determining the time signature

A more abstract characteristic which must be deduced in order to transcribe a musical recording is the recording's time signature. In essence, a piece's time signature represents the piece's rhythmic structure at a level higher than that of the pulse, this higher level is often referred to as the *meter* of the piece. A piece in $\frac{3}{4}$ time will generally be perceived by listeners as having every third pulse stronger than the rest. Similarly in $\frac{4}{4}$ it would be every fourth pulse. Two methods were investigated for determining the time signatures of excerpts. As discussed already, the candidate time signatures considered by the system are $\frac{4}{4}$, $\frac{3}{4}$, and $\frac{6}{8}$

Salience method

The first method analysed meter through the lens of musical *salience*. To determine which time signature is most appropriate for a piece, we iterate through each time signature and find the average salience of each of the notes the time signature predicts should be strong beats (e.g. for $\frac{3}{4}$ we find the average of the saliences of every sixth eighth note). We do this not just starting on the first note but also for each note in what would be the first bar of music, this means that if the first note in the piece does not actually begin a bar (an anacrusis),and hence do not correspond to strong beats, the average salience found starting on other notes in the first bar may still indicate the signature in question is still the most appropriate. The time signature with the highest average salience is taken to be the time signature to be used for the transcription. We can take a simplified view of the three considered time signatures as defining the number of eighth notes which occur between successive strong beats.

Time signature	eigth notes between strong pulses
4/4	8
3/4	6
6/8	3

Pseudocode for for the salience-based time signature algorithm can be found below:

Algorithm 9 Salience time signature algorithm

```
procedure TimesignatureSalience(Notes)
    for each $k$ in $\{3, 6, 8\}$ do
        for each note in first bar $N_i$ do
            $n = 0$
            while $N_i$.start+$k \cdot n < N_{\text{last}}$.start do
                if $\exists N_j$ where $N_j$.start $= N_i$.start+$k \cdot n$ then
                    $\text{score}_{N_i}$ += Salience($N_j$)
                end if
                $n+ = 1$
            end while
            $\text{score}_{N_i} = \text{score}_{N_i}/n$
        end for
        $\text{score}_k = \max(\text{score}_{N_i})$
    end for
    $k_{\text{best}} = \underset{k}{\text{argmax}}\, \text{score}_k$
end procedure
```

This algorithm returns the eighth note increment which gives the highest average salience, this can then be converted to the corresponding time signature using the table already given above.

Rhythmic similarity method

The second method investigated compared rhythmic similarity of consecutive measures. In most forms of music, rhythmic ideas will be repeated or restated in accordance with the meter of the piece, e.g. a piece in $\frac{3}{4}$ will use rhythmic phrases lasting 3, or 6, (or 9, etc.) quarter notes, while a piece in 4/4 will have phrases lasting for multiples of four quarter note. By experimenting with different bar lengths and observing how rhythmically similar consecutive bars are to each other, it is possible to gauge how likely different time signatures are to corresponding to a given piece of music. Rhythms can be represented in a large number of ways [T$^+$04]. Different ways of representing a rhythm entail different methods to measure how similar one rhythmic passage is to another. In this project, two representations, and thus two similarity measures were investigated. The first representation of rhythm considered is a basic representation of the rhythm as a sequence of binary digits.

The length of the binary sequence corresponds to the rhythmic resolution of the sequence, so for comparing passages of notes four quarter notes long who's smallest rhythmic denomination is a sixteenth note, we would use binary sequences of length $4 * 4 = 16$. The binary sequence encodes which times onsets occur with respect to the underlying rhythmic grid. Below is an illustration of how the vector representation encodes the onsets times using the example of a grid with 8 subdivisions.

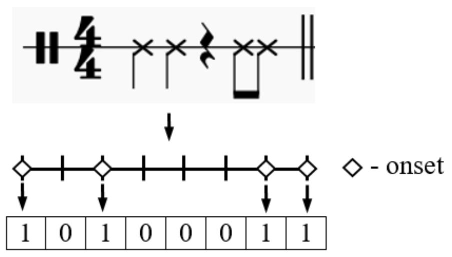


Figure 13: Illustration of how the vector for a given rhythm is generated, showing the notated rhythm (top), a symbolic representation of how the onsets fall on a grid (middle) and the corresponding vector (bottom).

To compare the rhythmic similarity between two of these rhythmic binary sequences, we consider the sequences as vectors with entries from the set $\{0, 1\}$ and consider the euclidean distance between them. Recall the euclidean distance between two vectors **u** and **v** is defined as:

$$\textsc{Sim}_1(\mathbf{u},\mathbf{v}) = |\mathbf{u}-\mathbf{v}| = \sqrt{\sum_{i=1}^{n} |u_i - v_i|^2}$$

That is to say we consider vectors which are close to each other as more similar than vectors which are farther apart.

We also consider a second representation of rhythm and a corresponding separate measure of similarity, first described in [Gus87], the so called Temporal Elements Displayed As Squares (TEDAS) representation of a rhythm. This representation is related to, and can be derived from the binary representation given above. The TEDAS representation of a rhythm can be seen geometrically as the union of boxes with side length corresponding to the duration of each rhythm in a passage. Below can be seen the TEDAS representation of the common *son clave* latin rhythm.

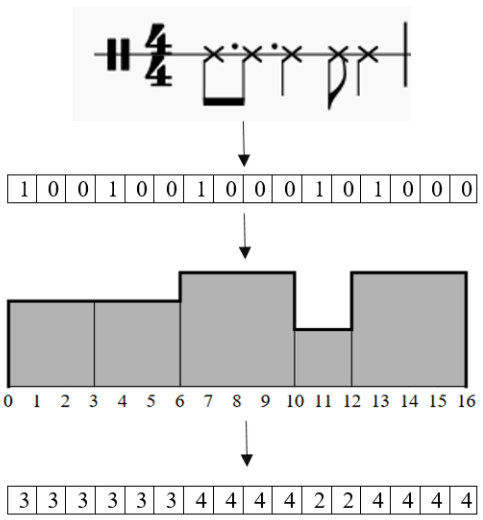


Figure 14: Derivation of TEDAS representation of *Son clave* rhythm (geometric visualisation taken from [T$^+$04])

To measure the similarity between two TEDAS representations of rhythms, we take as inspiration the measure outlined by Hoffman-Engl in [HE02]. For two vector representations of a TEDAS rhythm **v** and **u** of length n, their similarity is computed as:

$$\textsc{Sim}_2(\mathbf{v},\mathbf{u}) = \sqrt{\frac{\sum_{i=1}^{n} \exp\left(\frac{-1}{n}|v_i - u_i|^2\right)^2}{n}}$$

The use of the exponential operator makes sure notes of longer duration do not have a disproportionately large effect on the similarity measure. The closer $\textsc{Similarity}(\mathbf{v},\mathbf{u})$ is to 1, the more similar the rhythms. For determining the time signature, we want to be able to compare the average similarity of different lengths of rhythmic phrase, so we also normalise the similarity measure like so:

$$\tilde{\textsc{Sim}_2}(\mathbf{v},\mathbf{u}) = 1 - \frac{\textsc{Sim}_2(\mathbf{v},\mathbf{u}) - \max\limits_{\mathbf{x},\mathbf{y}} \textsc{Sim}_2(\mathbf{x},\mathbf{y})}{1 - \max\limits_{\mathbf{x},\mathbf{y}} \textsc{Sim}_2(\mathbf{x},\mathbf{y})}$$

This also has the advantage of making this measure more similar to the Euclidean distance measure where a smaller output indicates more rhythmically similar phrases. To determine which time signature leads to the best similarity measure we apply the following algorithm.

Algorithm 10 Rhythmic similarity algorithm

```
1: procedure TIMESIGNATURESIMILARITY(Notes)
2:     for each $k$ in {3, 6, 8} do
3:         counter = 0
4:         for each consecutive rhythmic phrase of length $k$ 16th notes, $\mathbf{v}_i$ do
5:             for each consecutive rhythmic phrase of length $k$ 16th notes, $\mathbf{u}_j$ do
6:                 $\text{score}_k = \text{score}_k + \textsc{Sim}(\mathbf{v}_i, \mathbf{u}_j)$
7:                 counter = counter + 1
8:             end for
9:         end for
10:        $\text{score}_k = \text{score}_k/\text{counter}$
11:    end for
12:    $k_{\text{best}} = \operatorname{argmax}_k \text{score}_k$
13: end procedure
```

In the algorithm above, SIM refers to either of the two similarities measures discussed, with $\mathbf{v}_i$ and $\mathbf{u}_j$ being the appropriate vector representations of the rhythmic passages for the chosen similarity measure.

One interesting thing to note is these two different ways of measuring rhythmic similarity provide different answers to the question “what is the least similar pair of rhythms?”. For the euclidean vector distance representation, the two most different rhythms of length n are the rhythms corresponding to a vector of all 1s compared to a vector of all 0s. (i.e. an empty bar is most rhythmically different from a bar consisting of all 16th notes), while for the chronomtonic representation, the two most different rhythms are the empty vector, and the vector corresponding to a single note on the first beat of the phrase (i.e. an empty bar is most rhythmically different from a bar consisting of a single sustained note).

4.11 Typesetting

Once we have extracted all the necessary information from our excerpt, the final step is to actually typeset the result as sheet music. There are many tools available to assist with automated music typesetting so it wasn’t necessary to spent a great deal of time during the project working on this aspect of of the system. The system exported sheet music in two formats, as image files and also as MusicXML, an XML-based file format which can be read by many standard music notation programs (e.g. Musescore, Sibelius, etc.)

Typesetting was relatively straight forward, the `music21` python library provides a *stream* object which notes and other musical features can be added to (e.g. time signature, key signature, etc.), which can then be used to export an image or musicXML file containing the information loaded into it. Therefore typesetting was simply a case of loading all the information which had been derived: Notes, tempo, key signature, time signature, into a stream object which was then used to export to the appropriate format. `music21` utilises the open source musical typesetting program Lilypond in order to create the final sheet music images. Lilypond automatically determines the clef to notate the transcription using based on which clef will have the most notes within the five normal stave lines.

The algorithm for exporting the transcription was straightforward.

Algorithm 11 Typeset music

```
1: procedure TYPESET(Notes)
2:     create music21 stream $S$
3:     for each note $N_i$ in notes do
4:         convert $N_i$ to music21 note object $N'_i$
5:         $S$.add($N'_i$)
6:     end for
7:     Set $S$ time signature
8:     Set $S$ key signature
9:     Set $S$ tempo
10:    generate image from $S$ using Lilypond
11:    Export $S$ as MusicXML
12: end procedure
```

5 Artefact evaluation and case studies

This section discusses the methodology taken to quantify the robustness and quality of the different solutions implemented to the variety of problems tackled throughout the project, and holistically evaluate the system as a whole. For each part of the system analysed, a discussion is given regarding what is being tested and relevant data is reported.

Due to the number of different working parts which constitute the system as a whole, and due to the difficulty associated with systematically evaluating how "close" two pieces of sheet music (e.g. the original and that generated by the program) are to each other, the overall robustness of the system is demonstrated by showcasing a selection of example transcriptions. Each example includes discussion of what the system gets right and what the system gets wrong in the description, along with discussion of potential explanations for unexpected behavior.

5.1 Quantisation

evaluation methodology

To compare the two quantization algorithms implemented in the project, both algorithms were run for a number of simulated rhythmic performances. The performances were generated by looping a rhythmic phrase and adding random noise to the onsets so as to simulate the errors and idiosyncrasies present in human performance. The output of both quantization measures were evaluated in two ways. Firstly the accuracy of the quantization was considered. How close was the average quantized beat to an actual beat in the "performed" rhythm. This is given in the table below as the average distance between the quantized note and the actual note, measured in quarter-notes (i.e. 0.3 means that quantization moved each note onset by 0.3 quarter notes on average). The second measure measured how complex the quantized rhythms ended up being. This was measured using the prior $p(c)$ defined in [CDK00]. The rhythmic motifs were repeated across a broad range of tempi, from 65 to 115bpm, and the values given in the table below represent the average values for each part of the system. Each algorithm was also given a correct pulse to quantize the input to.

(A = average difference between quantized and non-quantized onsets, in quarter notes, B = $p(c)$ of quantized rhythm.)

		Standad Deviation (ms)							
		0		15		25		35	
Rhythm	Algorithm	A	B	A	B	A	B	A	B
Son Clave	Naive	0.0131	0.0209	0.1788	0.0244	0.2194	0.0244	0.1981	0.0374
	MAP	0.0	0.0123	0.0121	0.0133	0.016	0.0202	0.0184	0.0296
Waltz	Naive	0.0049	0.0364	0.1965	0.0643	0.2643	0.0813	0.2254	0.076
	MAP	0.0	0.0201	0.0126	0.0222	0.0163	0.0313	0.0182	0.0378
8th notes	Naive	0.0031	0.0034	0.1784	0.0147	0.1409	0.012	0.1007	0.0133
	MAP	0.0	0.003	0.0118	0.0031	0.0163	0.0031	0.0183	0.0038
Bossa Nova	Naive	0.0031	0.0122	0.2942	0.0152	0.2155	0.0162	0.2716	0.0289
	MAP	0.0	0.0125	0.0121	0.0125	0.0165	0.0124	0.0184	0.0132
Mission Impossible	Naive	0.0207	0.1689	0.3099	0.196	0.2497	0.1952	0.2523	0.1916
	MAP	0.0	0.1386	0.012	0.1398	0.0168	0.1501	0.0184	0.1512
Overall	Naive	0.009	0.0484	0.2316	0.0629	0.218	0.0658	0.2096	0.0694
	MAP	0.0	0.0373	0.0121	0.0382	0.0164	0.0434	0.0183	0.0471

Remarks

We can observe that the MAP estimation method consistently out performs the naive method in regards to both accuracy and simplicity which is somewhat surprising. We would expect that the naive method might prioritise accuracy at the expense of simplicity, however it is outperformed by the MAP method in both regards. This indicates that overall, unsurprisingly, the more complex MAP estimation method for quantization provides a higher quality result than alternative, naive method.

5.2 Key signature

evaluation methodology

To evaluate the algorithm implemented for detecting the key signature of a piece, data was gathered consisting of MIDI versions of pieces not containing any modulations (changes of key) whose key was known. The note pitches were then extracted from the MIDI files and fed to the algorithm. The algorithm's output was then compared to the known key.

Two quantities were tallied. Firstly, how often was the key signature correct, and secondly, how often the key signature predicted was a perfect fifth/perfect fourth away from the correct key signature. Key signatures differing by only a single note are separated by perfect fifths/perfect fourths (e.g. F contains 1 flat, C contains 0 flats or sharps, and G contains one sharp). If the algorithm is able to determine the key to within a perfect fifth/fourth, it indicates the algorithm is close to agreeing with the known correct key. Relatedly, transcribing a piece in a key a perfect fifth/fourth from the correct key will still result in a piece of sheet music which would be relatively easy to read for a musician, as opposed to if a key was chosen at random, in which case, the sheet music would likely be very confusing and difficult to read. Hence, it is valuable to know the frequency of these "near misses".

Data set	# of pieces	Correct Key (%)	Fifth/fourth away from correct key (%)
Bach	31	96.8	100.0
Mozart	17	100	100.0
Beethoven	18	66.7	94.4
Total	66	89.4	98.5

Remarks

The basic algorithm implemented performs well, almost without fail it is able to determine the key of the piece to within a perfect fifth/fourth, and for the vast majority (almost 90%) of cases, it is able to determine the key correctly. This is impressive since the key signature of a piece is a characteristic not objectively determined just by the notes in the piece, and due to the relationship between western music theory and key signatures, there are pieces of music which don't contain clear transpositions but for which it can be debated what key is most "correct" or "appropriate" [Man02].

5.3 Time signature

evaluation methodology

To test the robustness of the time signature detection capabilities of the system, a similar approach was taken as to with the key signature. A selection of MIDI files containing the note information for a variety of works with a variety of time signatures was fed into the algorithms and the rate of success for each algorithm was recorded. The tempi of the pieces in question was known and since the notes came from a midi recording, they were already quantized.

		Time signature correctly detected (%)			
		Rhythmic Similarity		Salience	
Time signature	# of pieces	Chronotonic	Euclidean	Linear	Nonlinear
4/4	23	47.8	13.0	65.2	60.9
3/4	25	48.0	20.0	40	24.0
6/8	3	0.0	100.0	0.0	0.0
total	51	45.1	21.6	49.0	39.2

Remarks

The results for the time signature experiment are somewhat inconclusive, partially this may be to do with the limited test data which was found to be available (although there are thousands of free MIDI files available, very few are monophonic and have the note onsets defined in an easy to parse manner). From the results obtained, we observe that neither algorithm performs exceedingly well in this task. Although the test data does contain a few pieces in $\frac{6}{8}$, the majority are in $\frac{4}{4}$ or $\frac{3}{4}$, so if we wanted to have confidence that our methods worked, it would be encouraging if they had overall success rates better than 50%. Another potential issue may be the test data, which consisted wholly of solo works by Bach. Many of these pieces incorporate long streams of sixteenth notes rather than more rhythmically varied phrases. For the rhythmic similarity approach, without much rhythmic information to go on, it would be unsurprising for the accuracy of the algorithm to be significantly reduced.

5.4 Tempo detection

One of the most important aspects of the system was the ability to detect and track the tempo of a recording. To test the tempo detection ability of the system. Firstly we will examine the tempo detection system

Tempo detection

The tempo tracking algorithms implemented take as input a list of note onset times in milliseconds (ms). To evaluate them, a set of lists of simulated note onset times was generated. The onset lists were designed to simulate rhythms played by human beings and ranged in how accurate the beat was followed and how much the overall tempo fluctuated.

In order to accurately simulate a rhythm source performed by a human, two sources of controlled random variation were added. Firstly, the onset of each note was displaced by an amount ε based on a normal distribution, secondly, the overall tempo fluctuated with respect to a randomly generated signals have a frequency spectrum such that the intensity of a frequency, $S(f)$ is inversely proportional to f itself.

$$S(f) \propto \frac{1}{f^\alpha}$$

This is in contrast to white noise where $S(f)$ is constant. Pink noise appears very frequently in natural systems and systems involving the human body, including fluctuations in tempo tapping without a metronome [RPV+15][SVS01]. For the experiments below, the pink noise generated took $\alpha = 0.7$. To test the tempo detection system, the standard deviation of the onset times was varied and tests were performed twice, once with no variation in tempo, and once with pink noise variation in tempo, the pink noise generated had mean zero and standard deviation 1.5, which was found as a good amount to simulate human like tempo fluctuation.

(Note A = correct tempo is first estimate (%), B = correct tempo one of top 3 estimates (%), C = First guess simple ratio of correct tempo, where a simple ratio is either $\frac{1}{1}$ $\frac{1}{2}$, $\frac{2}{3}$, $\frac{1}{3}$, or the inverse of one of these fractions (%)).

No tempo noise:

		Standard deviation (ms)											
		0			15			25			35		
Pulse type	Algorithm	A	B	C	A	B	C	A	B	C	A	B	C
Constant 8th notes	Dixon	100.0	40.7	100.0	100.0	40.7	98.1	80.2	33.3	85.2	65.7	25.9	69.4
	Naive	59.3	40.7	100.0	59.25	42.59	100.0	54.3	40.7	100.0	50.0	37.0	97.2
Son clave	Dixon	85.2	18.5	100.0	66.7	18.5	81.5	42.6	13.0	57.4	35.8	11.1	49.4
	Naive	63.0	37.0	92.6	59.3	40.7	77.8	50.0	35.2	66.7	44.4	33.3	61.7
Bossa Nova	Dixon	85.2	18.5	100.0	72.2	16.7	96.3	53.1	13.6	76.5	42.6	11.1	62.0
	Naive	74.1	40.7	66.7	66.7	38.9	59.3	53.1	32.1	51.9	45.4	28.7	48.1
Mission Impossible	Dixon	100.0	40.7	100.0	96.3	38.9	96.3	74.1	27.2	76.5	62.0	22.2	61.1
	Naive	66.7	29.6	88.9	62.0	35.2	94.4	55.6	34.6	93.8	50.0	30.6	89.8
Waltz	Dixon	100.0	40.7	100.0	98.1	38.9	96.3	84.0	34.6	82.7	71.3	30.6	70.4
	Naive	70.4	40.7	96.3	63.0	40.7	98.1	54.3	38.3	95.1	47.2	35.2	88.9
Overall	Dixon	94.1	31.8	100.0	86.7	30.7	93.7	66.8	24.3	75.7	55.5	20.2	62.5
	Naive	66.7	37.7	88.9	62.0	39.6	85.9	53.5	36.2	81.5	47.4	33.0	77.1

Tempo noise:

		Standard deviation (ms)											
		0			15			25			35		
Pulse type	Algorithm	A	B	C	A	B	C	A	B	C	A	B	C
Constant 8th notes	Dixon	48.1	29.6	48.1	48.1	20.4	38.9	42.0	18.5	35.8	38.0	15.7	32.4
	Naive	44.4	33.3	59.3	59.25	42.59	100.0	35.8	25.9	46.9	33.3	23.1	44.4
Son clave	Dixon	29.6	14.8	33.3	24.1	11.1	29.6	21.0	8.6	27.2	18.5	7.4	25.0
	Naive	40.7	37.0	59.3	35.2	27.8	46.3	30.9	23.5	39.5	28.7	19.4	34.3
Bossa Nova	Dixon	18.5	3.7	22.2	14.8	3.7	24.1	13.6	2.5	24.7	15.7	2.8	24.1
	Naive	33.3	18.5	29.6	14.8	3.7	24.1	24.7	13.6	27.2	22.2	10.2	23.1
Mission Impossible	Dixon	18.5	18.5	25.9	20.4	11.1	20.4	22.2	11.1	25.9	21.3	9.3	24.1
	Naive	33.3	22.2	44.4	22.2	11.1	33.3	22.2	11.1	33.3	18.5	8.3	30.6
Waltz	Dixon	37.0	18.5	40.7	33.3	18.5	33.3	29.6	17.3	28.4	28.7	16.7	27.8
	Naive	25.9	14.8	33.3	20.4	14.8	33.3	22.2	14.8	32.1	21.3	13.0	27.8
Overall	Dixon	30.3	17.0	34.0	28.1	13.0	29.3	25.7	11.6	28.4	24.4	10.4	26.7
	Naive	35.5	25.2	45.2	30.4	20.0	47.4	27.2	17.8	35.8	24.8	14.8	32.0

Remarks

Somewhat surprisingly, the performance of the two algorithms is relatively similar. For less noisy signals it seems that Dixon's algorithm outperforms the naive approach somewhat, however especially for noisier signals, the naive approach actually performs slightly better than Dixon's. One thing not considered here however is the efficiency of the two algorithms. Performing the differential evolution optimisation step in the naive algorithm means it is significantly slower than Dixon's algorithm.

5.5 Tempo tracking

In order to test the system's tempo tracking ability, rhythmic pulses were generated as before, with pink noise again being used to generate tempo variation over time. For each onset which lay on a strong beat with respect to the pulse of the rhythm, it was measured whether the pulse generated by the system coincided with it (within a 3% tolerance). The percentage of strong beats which aligned with the calculated pulse is recorded in the table below.

	Standard deviation of pink noise			
Rhythm type	0.0	1.0	2.0	3.0
Waltz	100.0	77.9	62.2	54.7
Constant 8th notes	100.0	85.3	70.5	55.2
Bossa Nova	50.0	47.8	52.2	56.5
Mission Impossible	100.0	84.8	74.4	60.9
Son Clave	68.5	76.4	65.1	58.3
Overall	83.7	74.4	64.9	57.1

remarks

Unsurprisingly, overall as the tempo deviation increased, the performance of the algorithm gradually worsened. This was the expected behavior of the algorithm, and even when the intensity of the pink noise was increased, the algorithm was still able to stay in time a relatively high percentage of the time

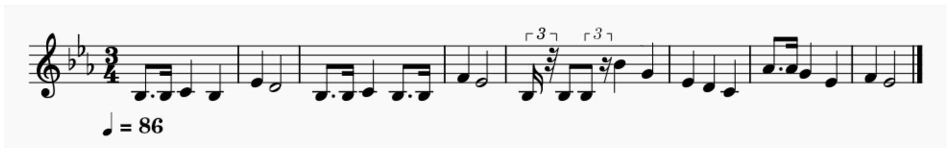
5.6 Case studies

In order to more comprehensively evaluate the robustness of the system, a set of 5 example transcriptions are discussed in this section. For each section we evaluate the subjective quality of the transcription and discuss possible explanations for mistakes or discrepancies. The five excerpts were taken from a variety of places. Each excerpt was processed using *Audacity*, an open source basic audio editing suite to normalise the audio level to a suitable level and export the audio in the correct format to be processed by the system (*.wav* files with a 48kHz sample rate).

1. Happy Birthday

Background: The first excerpt we will look at is a performance of a trumpet player playing the well-known traditional tune Happy Birthday. The recording was found on Youtube.

Transcription



Settings used to generate transcription

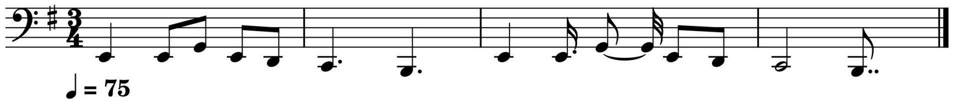
Sub-task	Approach
Time Signature	Rhythmic similarity
Quantization	MAP
Tempo	Dixon algorithm

Transcribed Key: E♭ (Correct)
Transcribed Time Signature: $\frac{3}{4}$ (Correct)
Discussion: The system manages to product a good transcription of Happy Birthday. The most obvious mistake is the awkward triplet-based rhythm observed in measure 7. This can be traced to the detection of a false positive onset time. Since the false positive appears at an essentially random moment (it does not relate to the rhythmic structure of the other correctly detected notes), it unsurprisingly results in an awkward quantization for that section. Another false positive onset is present in the fifth bar however may be harder to notice as it coincidentally gets quantised to the dotted-eighth-then-16th-note pattern which is a reoccurring rhythmic motif in the melody of Happy Birthday. Both the time signature and key signature of the piece are transcribed correctly and the tempo has been chosen so as to allow the notated notes to be of sensible durations. Overall this is quite a strong transcription with only a couple of minor mistakes.

2. The White Stripes - Seven Nation Army (intro)

Background: The next excerpt is taken from the first few measures of the White Stripes song "Seven Nation Army", it is played on an overdriven bass guitar.

Transcription



Settings used to generate transcription

Sub-task	Approach
Time Signature	Rhythmic similarity
Quantization	MAP
Tempo	Dixon algorithm

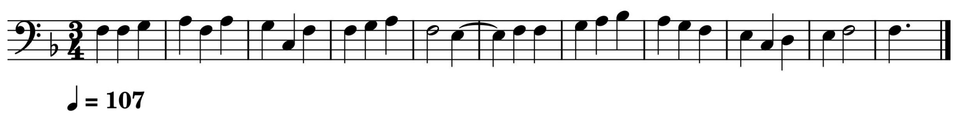
Transcribed Key: G (Correct)
Transcribed Time Signature: $\frac{3}{4}$ (Incorrect)
Discussion: The system produces a satisfactory transcription of the introductory riff to Seven Nation Army. The time signature is not correctly detected, however the time signature chosen does make sense in the context of the pulse which was detected and quantization chosen. The underlying pulse detected by the system appears to be a factor of $\frac{3}{4}$ slower than the actual tempo; in a correct transcription the second measure would consist of two half notes, whereas in ours it consists of two dotted quarter notes

(which have $\frac{3}{4}$ the duration of a half note). In the context of this incorrect pulse then it does make sense to notate the excerpt in $\frac{3}{4}$ since this will place the bar lines between the same notes as they would be if the pulse was detected correctly. Since the detected pulse is related what would be the correct pulse by a relatively simple ratio, the result is still a fairly readable transcription. The other mistakes which we can observe are some slightly awkward quantization choices: In the third bar, the second and third notes are off by a 16th note from what would be the correct rhythm, and similarly the final bar has a doubly dotted eighth note as the final note of the excerpt, which would be an uncommon duration to conclude a bar on, instead the more sensible choice would be to notate it as just a quarter note.

3. Yankee Doodle

Background: The next excerpt is a well known traditional melody and was recorded by the author, played on an acoustic guitar. There is no standard key to play Yankee Doodle in, in the recording it was played in F.

Transcription



Settings used to generate transcription

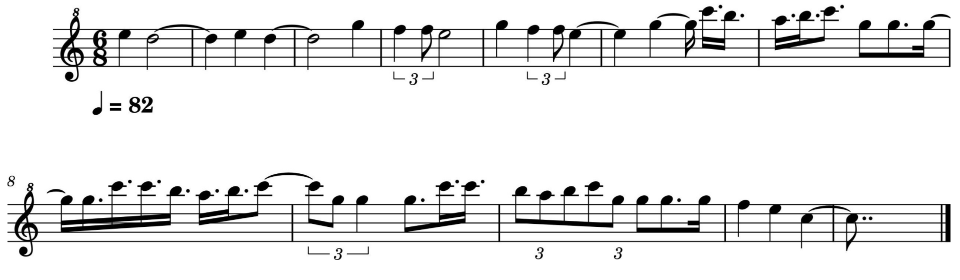
Sub-task	Approach
Time Signature	Rhythmic similarity
Quantization	MAP
Tempo	Dixon algorithm

Transcribed Key: F (Correct)
Transcribed Time Signature: $\frac{6}{8}$ (Incorrect)
Discussion: The system produces a good transcription of Yankee Doodle. The only mistake made was in the time signature, however apart from that, the transcription does not make any mistakes. The fact that the melody consists almost solely of quarter notes with just a couple of half notes as well means that the tempo detection system has a relatively easy time “locking in” to the correct pulse, since there aren’t any shorter subdivisions to throw off the system.

4. Three Blind Mice

Background: The next excerpt is a performance of three blind mice played on a toy glockenspiel, the recording was found in a youtube video.

Transcription



Settings used to generate transcription

Sub-task	Approach
Time Signature	Rhythmic similarity
Quantization	MAP
Tempo	Dixon algorithm

Transcribed Key: C (Correct)
Transcribed Time Signature: $\frac{6}{8}$ (Correct)
Discussion: The system struggled in producing a transcription of three blind mice. Despite the fairly clear transients of the glockenspiel, several note onsets were not detected by the onset detector, this is most noticeable in the first few bars. The sections containing notes played quite quickly challenged the pulse detection system since it was relatively easy for agents to get swayed by an onset *close to* the where the pulse actually lay. This had the knock on effect of making some of the quantisation slightly awkward since the system was trying to quantize notes over an awkward time interval. despite this, the system managed to correctly identify both the key and time signature of the excerpt and it is easy to see the general shape and contour of the melody being played. Despite the overall quality of this transcription not being extremely high, the outputted transcription still demonstrates the systems knowledge of certain musical features of the excerpt.

6 Critical appraisal

The main goal undertaken in this project was to create a full audio-to-sheet transcription system which could transcribe simple monophonic musical recordings and export them as typeset sheet music. The initial goals outlined for the project were changed as more understanding about nature of the problem and the context surrounding it was learned. The updated goals for the project provided a rewarding challenge, with many different techniques and approaches required to build and implement a system to meet them.

One of the main challenges faced throughout this project was that of time allocation. The goal to build a fully functional audio-to-sheet system, even with the restrictions placed on it in this case, was a fairly ambitious one. Certain aspects of the project were necessarily limited by the amount of time which could be dedicated to working on them. The task of creating an audio-to-score system is broad enough that it would be possible to approach completing this tasks from a vast number of perspectives, it is easy to imagine a Senior Honours or even Masters project which focused on the same goals, only taking a more intense and longer approach to solving them. As noted at the start of this project, the aim of this project was not to implement a state of the art system to rival the newest and best MIR technology or recent deep learning approaches which have been taken to the problem, but to balance complexity with attainability and first and foremost, develop a complete working system, and this goal has been achieved.

Another aspect of the project which proved challenging was testing the implemented system. Since the system does so many things, it proved difficult to develop a systematic way of testing the entire system. Instead a hybrid approach was taken, where the most important parts of the system were tested systematically and objectively while a more holistic evaluation was also undertaken by looking at a variety of example transcriptions produced by the system.

With regards to the specific goals given for this project, all the basic goals were met and the extended goal of providing support for a variety of instruments was also met, although this actually turned out to be a relatively simple goal to fulfill, since pitch and onset detection systems generally do not discriminate between different instruments greatly. Although the goals accomplished chiefly consisted of primary goals, the primary goals required to build a full audio-to-score system were relatively demanding and based on how time was allocated throughout the project, it seems that to meet any further goals than those listed as primary ones would be somewhat unrealistic for a minor project.

As we have already discussed, although there exist publicly available audio-to-score systems, it is difficult to compare the specification of the system developed in this report to such systems since these public systems are proprietary in general and require payment or subscription to use.

Compared to contemporary MIR research, the system outlined in this report does not demonstrate as robust or broad a set of features as to what is available, however it is difficult to make a direct comparison between these contemporary systems and the one discussed in this project, since these contemporary projects are undertaken over longer periods of time and by individuals with far more expertise in the field of MIR than the author did at the beginning of undertaking this project.

7 Conclusion

The chief goal of this project was to develop a full audio-to-score automatic transcription for simple monophonic instruments. This goal was achieved, and a system able to transcribe melodies played by a variety of instruments was developed. A variety of techniques and approaches to the problem presented were investigated over the course of the project and the final system produced represented the culmination and combination of a diverse set of procedures and algorithms.

There are many ways the functionality of the system could be developed given more time to work on it. The system developed would provide a strong basis for developing a more robust automatic transcription system, or a system with less restrictions on the types of recordings it could process. For example it was already noted that the system will only transcribe pieces in $\frac{4}{4}$, $\frac{3}{4}$, or $\frac{6}{8}$, and despite these time signatures covering a vast majority of western music, they are not exhaustive. Furthermore, a more fully featured transcription system would probably support polyphonic transcription, the main challenge in this regard would likely be developing accurate polyphonic pitch detection methods, which is still a challenging problem in MIR. it would be possible to generalise many of the techniques and algorithms from this project to work with data containing simultaneous notes, and some problems might actually be made easier, for example a more well motivated model for salience could possibly be developed by considering the consonance and dissonance of the harmonies present and the number of notes being played at a given time simultaneously. Overall the system developed over the course of this project represents a full monophonic audio-to-score which could provide a strong basis for developing more powerful transcription systems with more fully featured specifications.

8 Appendix: Running the program

The system can be run as a python script on the command line. The modules used throughout the system which are used by the system which may need to be installed were: `aubio, colorednoise, matplotlib, mido, music21, numpy, Pillow, PyAudio, pynput, scipy`

The script can then be run from within the project directory as

```
python command_line_tool.py -i [PATH TO INPUT FILE] -o [PATH TO OUTPUT DIRECTORY]
```

To change algorithms used for the different sub-tasks by the system (i.e. use naive quantization method instead of MAP method), options can be found at the top of the `command_line_tool.py` file, with directions on how to change options as required.

References

[Abl22] Ableton. Ableton reference manual version 11. 2022.

[BC02] Donald Byrd and Tim Crawford. Problems of music information retrieval in the real world. *Information processing & management*, 38(2):249–272, 2002.

[BDA+05] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on speech and audio processing*, 13(5):1035–1047, 2005.

[BKS12] Sebastian Böck, Florian Krebs, and Markus Schedl. Evaluating the online capabilities of onset detection methods. In *ISMIR*, pages 49–54. Citeseer, 2012.

[BMS00] Juan Pablo Bello, Giuliano Monti, and Mark Sandler. An implementation of automatic transcription of monophonic music with a blackboard system. In *Proceedings of the Irish Signals and Systems conference (ISSC 2000), Dublin, Ireland, June*, 2000.

[CDK00] Ali Taylan Cemgil, Peter Desain, and Bert Kappen. Rhythm quantization for transcription. *Computer Music Journal*, 24(2):60–76, 2000.

[CVG+08] Michael A. Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[DaT22] DaTuner. Guitar tuner, bass, violin, banjo more. `https://play.google.com/store/apps/details?id=com.bork.dsp.datuna&hl=en_GB&gl=US`, 2022. [Online; accessed 26-03-2022].

[DCK02] Alain De Cheveigné and Hideki Kawahara. Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.

[Dix01] Simon Dixon. Automatic extraction of tempo and beat from expressive performances. *Journal of New Music Research*, 30(1):39–58, 2001.

[Dix06] Simon Dixon. Onset detection revisited. In *Proceedings of the 9th International Conference on Digital Audio Effects*, volume 120, pages 133–137. Citeseer, 2006.

[DLCMS01] Patricio De La Cuadra, Aaron S Master, and Craig Sapp. Efficient pitch detection techniques for interactive music. In *ICMC*, 2001.

[Elo20] Anders Elowsson. Polyphonic pitch tracking with deep layered learning. *The Journal of the Acoustical Society of America*, 148(1):446–468, 2020.

[Fen22] Fender. Online guitar tuner. `https://www.fender.com/online-guitar-tuner`, 2022. [Online; accessed 26-03-2022].

[Foo97] Jonathan T Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II*, volume 3229, pages 138–147. International Society for Optics and Photonics, 1997.

[G+03] David Gerhard et al. *Pitch extraction and fundamental frequency: History and current techniques*. Department of Computer Science, University of Regina Regina, SK, Canada, 2003.

[GBS12] Emilia Gómez, J Bonada, and Justin Salamon. Automatic transcription of flamenco singing from monophonic and polyphonic music recordings. In *III Interdisciplinary Conference on Flamenco Research (INFLA) and II International Workshop of Folk Music Analysis (FMA)*, 2012.

[Get22] GetSongBPM. Mp3 to bpm (song analyser). `https://getsongbpm.com/tools/audio`, 2022. [Online; accessed 26-03-2022].

[Gus87] K Gustafson. A new method for displaying speech rhythm, with illustrations from some nordic languages, nordic prosody iv, editado por k. gregersen y h. basboll, pág. 105-114, 1987.

[HE02] Ludger Hofmann-Engl. Rhythmic similarity: A theoretical and empirical approach. In *Proceedings of the seventh international conference on music perception and cognition*, pages 564–567, 2002.

[Kla99] Anssi Klapuri. Sound onset detection by applying psychoacoustic knowledge. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 6, pages 3089–3092. IEEE, 1999.

[LS09] Mary Lourde and Anjali Kuppayil Saji. A digital guitar tuner. *arXiv e-prints*, pages arXiv–0912, 2009.

[Man02] Peter Manuel. From scarlatti to "guantanamera": Dual tonicity in spanish and latin american musics. *Journal of the American Musicological Society*, 55(2):311–336, 2002.

[MS00] Giuliano Monti and Mark Sandler. Monophonic transcription with autocorrelation. In *Proceedings of the COST G-6 Conference on digital audio effects (DAFX-00), Verona, Italy*, pages 257–260, 2000.

[NLAF19] Néstor Nápoles López, Claire Arthur, and Ichiro Fujinaga. Key-finding based on a hidden markov model and key profiles. In *6th International Conference on Digital Libraries for Musicology*, pages 33–37, 2019.

[RCRM76] L. Rabiner, M. Cheng, A. Rosenberg, and C. McGonegal. A comparative performance study of several pitch detection algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(5):399–418, 1976.

[RPCZ18] Miguel A Román, Antonio Pertusa, and Jorge Calvo-Zaragoza. An end-to-end framework for audio-to-score music transcription on monophonic excerpts. In *ISMIR*, pages 34–41, 2018.

[RPV$^+$15] Esa Räsänen, Otto Pulkkinen, Tuomas Virtanen, Manfred Zollner, and Holger Hennig. Fluctuations of hi-hat timing and dynamics in a virtuoso drum track of a popular music recording. *PLoS One*, 10(6):e0127902, 2015.

[Ryy04] Matti Ryynänen. Probabilistic modelling of note events in the transcription of monophonic melodies. 2004.

[S06] Dixon S. Simple spectrum-based onset detection. 2006.

[SVS01] P Szendro, Gy Vincze, and A Szasz. Pink-noise behaviour of biosystems. *European Biophysics Journal*, 30(3):227–231, 2001.

[T$^+$04] Godfried T Toussaint et al. A comparison of rhythmic similarity measures. In *ISMIR*. Citeseer, 2004.

[TA22] Carl Thomé and Sven Ahlbäck. Polyphonic pitch detection with convolutional recurrent neural networks. *arXiv preprint arXiv:2202.02115*, 2022.

[Tem02] David Temperley. A bayesian approach to key-finding. In *International Conference on Music and Artificial Intelligence*, pages 195–206. Springer, 2002.

[Tun22] Tunebat. Song key bpm finder. `https://tunebat.com/Analyzer`, 2022. [Online; accessed 26-03-2022].

[Wal09] Rob Walker. The song decoders at pandora. `https://www.nytimes.com/2009/10/18/magazine/18Pandora-t.html`, 2009. [Online; accessed 26-03-2022].

[You22] Yousician. Guitartuna. `https://yousician.com/guitartuna`, 2022. [Online; accessed 26-03-2022].