A Probabilistic System for Automatic Transcription of Monophonic Music

HARJYOT SINGH

4th Year Project Report

Artificial Intelligence and Computer Science

SCHOOL OF INFORMATICS

UNIVERSITY OF EDINBURGH

2018

Abstract

Automatic Music Transcription is one of the biggest unsolved problems in the field of Music Information Retrieval. In the last 3 decades, there has been a lot of interest in using principles of probability to solve this problem.

This report chronicles the development of a probabilistic system for converting monophonic music recordings into staff notation. The problem is broken down into sub-problems, namely pitch detection, rhythm detection, and key detection. The report looks at each subproblem in depth, building up the required fundamental principles of music theory, exploring relevant past works, implementing models for each, training these models on a corpus, and evaluating their performance.

The report then demonstrates the process of unifying all these models to create a whole system and finally compares it to a state-of-the-art transcription system and evaluates its performance on an unseen set of live recordings.

Acknowledgements

I would like to thank my supervisor, Dr.Alan Smaill for his unrelenting dedication, expertise and guidance in making this project see the light of day.

I would also like to extend my sincerest gratitude David Temperley, for taking time out to explain his work to me and to all the members of the ISMIR Community, who have been astonishingly prompt and patient in introducing me to this field and showing me the ropes.

Last but not the least I would like to thank my family and my partner for their never ending support and motivation.

CONTENTS

1 INTRODUCTION

"Music transcription can be seen as discovering the 'recipe', or reverse-engineering the 'source code' of a music signal"[1]

[1] Anssi Klapuri and Manuel Davy. *Signal processing methods for music transcription*. Springer Science & Business Media, 2007

Automatic Music Transcription (AMT) is defined as the process of converting a digital audio signal into symbolic notation. The symbolic notation in this case usually refers to western music notation (commonly known as staff notation, or score), however in other literature it might refer to alternative forms of representation like piano-roll notation (a two-dimensional representation of musical notes across time), guitar tablature or MIDI. The process involves gathering information like pitch, pitch onsets and offsets, beat and tempo.

Music transcription can be categorised into two categories based on the type of input audio:

- *Monophonic* - where a single instrument plays only one pitch at a given time (eg. A guitar solo, flute solo, human voice singing etc.)
- *Polyphonic* - where multiple instruments can be playing multiple notes at a given time (orchestras, piano playing a musical piece with chords, a duet etc.)

In the last 3 decades the field has received a great interest, with a focus on solving transcription for polyphonic audio, while the older models focused on using probabilistic[2,3] or signal processing methods, there has been a boom in models[4,5] using deep learning in recent years to achieve what the older models couldn't.

[2] Ali Taylan Cemgil. *Bayesian music transcription*. [Sl: sn], 2004

[3] Christopher Raphael. Automatic transcription of piano music. In *ISMIR*, 2002

[4] Sebastian Böck and Markus Schedl. Polyphonic piano note transcription with recurrent neural networks. In *Acoustics, speech and signal processing (ICASSP), 2012 ieee international conference on*, pages 121–124. IEEE, 2012

[5] Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(5):927–939, 2016

The most immediate application of an AMT system is to provide musicians a way to record improvised performances and reproduce it, also it has applications in musical styles that don't necessarily have scores eg. oral folk traditions, jazz, pop etc. It can also be applied to further development of interactive musical systems such as score following, teaching instruments etc.

In this report, we propose a system for converting monophonic audio to common music notation using probabilistic models.

While some people consider the problem of transcription of monophonic audio solved, its not so because of the difference in the definition they use - from a scientific perspective, a correct estimation of pitches and their durations from audio might be considered 'successful' monophonic transcription, whereas the level of transcription we are trying to achieve with our system goes more in depth, looking at things like key signatures, time signatures, bar locations etc. which are actually required for the way a musician would think of music transcription. The AMT model in this report has been broken down into 3 sub-models or tasks, namely *pitch extraction, key detection and rhythm detection*. It was decided to use *Python 3*[6] for the project, mostly due to the rich ecosystem of libraries and tools that would be beneficial in achieving the requirements of the system.

[6] Guido Van Rossum and Fred L Drake. *The python language reference manual.* Network Theory Ltd., 2011

In the chapter *The Pitch*, we define the problem of pitch extraction, look at some of the most successful methods, we also illustrate how machine learning can be used to optimise the parameters for one of these popular models of pitch extraction, further improving its performance and how we utilise the optimised model in our AMT system.

In the chapter *The Key*, we walk through the fundamentals of music in terms of keys and scales whilst looking at important models proposed in the literature, the importance of key in an AMT system, propose an implementation of a probabilistic key finding model, optimising its parameters using a training corpus, and onboarding it on our AMT system.

In the chapter *The Beat*, we begin by explaining the fundamentals of rhythm like time signature, tempo. We then demonstrate the need to incorporate rhythm information into our AMT system, whilst also reviewing some of the more predominant work in solving the problem. We then illustrate how we extend a well known model to be used in our AMT system, and discuss the major problems in the field that still need to be solved.

In the chapter *The Score*, we introduce the fundamentals of the staff notation, demonstrate how the pitch, key and beat models are connected to create a whole AMT system, describe the process of creating a score from the combined output of these models, and talk about the simple web interface for the system.

In the chapter *The Finale*, we compare our AMT system against other well known commercial systems of music transcription and evaluate its performance. We then propose future work that could be done to solve the system's drawbacks.

2

THE QUEST FOR DATA

This chapter looks at finding the right dataset for our system and how it is onboarded.

2.1 *Requirements*

a As is the first step of every Machine Learning problem, We had to begin by finding the right dataset, the candidates had to meet the following requirements:

- *Size* - Should be quite substantial to be able to train a probabilistic model.
- *Monophonic* - Should be monophonic in nature, which is tricky because most large music datasets are polyphonic in nature.
- *Annotated* - Should be annotated in terms of information like Pitches, Key Signature, Time Signature.
- *Machine Readable & Convertable* - Should be in one of the popular formats that are viable to parse (eg. **kern[1], MusicXML[2]) and convert to audio formats.

[1] The humdrum **kern representaton. `https://musiccog.ohio-state.edu/Humdrum/representations/kern.html`

[2] The musicxml format. `https://www.musicxml.com/`

2.2 *Essen Folk Collection*

Through the recommendation of my supervisor, and other members of the ISMIR Community[3], the obvious choice which met all the above criterion was The Essen Folksong Collection[4]. The Essen Folksong Collection provides a dataset of *8473* folksongs that have been collected and encoded under the supervision of Helmut Schaffrath at the Gesamthochschule of Essen University from 1982 until 1994. Originally encoded in the Essen Associated Code (ESAC) format[5], the **kern version of the dataset was produced by automated translation followed by hand editing under the supervision of Dr.

[3] ISMIR - http://www.ismir.net/

[4] Helmut Schaffrath and David Huron. The essen folksong collection in the humdrum kern format. *Menlo Park, CA: Center for Computer Assisted Research in the Humanities*, 1995

[5] more information - http://www.esac-data.org/

David Huron. Table 2.1 shows a breakdown of the number of songs by geographic region

Table 2.1: Distribution of songs in Essen Collection by Country

Country	Number of Songs
Germany	5,260
China	1,972
France	297
Yugoslavia	115
Austria	114
Switzerland	105
Netherlands	102
Romania	37
Russia	20

Collecting the data

What seemed to be straight forward task, turned out to be a little bit more tricky. The Essen dataset is hosted on KernScore[6] however, it is hosted as a directory architecture with no clear way to download all the songs together in one go, also each song is available in 2 different formats - *.midi, .kern* and we needed both the files for each song.
A web scraper was written using *BeautifulSoup*[7] which is an HTML Parser for Python. It operated by essentially making a search request to the KernScore Website with Essen as the query, and pulling out all the ***kern* and *midi* file pair links. Once all the links (matched them against the total number of Essen Files available in the specification) were collected, Python's inbuilt *urlib.request* was used to download the files.

[6] http://kern.humdrum.org/cgi-bin/browse?l=/essen

[7] https://launchpad.net/beautifulsoup

Converting to Audio

Now that all the **kern and midi files were gathered, the next step was to convert the midi files into audio files. It was decided to go ahead with *.wav* as the audio file format because its *lossless* and *uncompressed*, which essentially means that a file encoded into *wav* will almost always be a 100% accurate and bit-for-bit reproduction of the source material.

To convert *MIDI*[8] data to audio, there's a requirement for a virtual synthesizer. FluidSynth was just right for the job as it converted MIDI data into an audio signal using SoundFont[9] technology without need for a SoundFont-compatible soundcard.

[8] MIDI (Musical Instrument Digital Interface) is a protocol which allows electronic instruments and other digital musical tools to communicate with each other. MIDI itself does not make sound, it is just a series of instructions like "note on" "note off", These messages are interpreted by a MIDI instrument to produce sound. Computers require a soundcard to play MIDI as audio.

What is midi? `http://www.instructables.com/id/What-is-MIDI/`

[9] https://en.wikipedia.org/wiki/SoundFont

After search through different Open-Source SoundFonts and three versatile ones were picked as they did not have too much of overlay effects like delay or reverb, each SoundFonts was a different instrument - namely, *Upright Piano, Acoustic Guitar and Flute*. Using *midi2audio*[10] which is a python package providing an interface to FluidSynth, The entire dataset was randomly divided into three equal sets, and converted each set with one of the SoundFonts into a *.wav* file. This was done to to add variation in the sound signals which was going to be needed in training the Pitch model (Chapter 3).

[10] https://github.com/bzamecnik/midi2audio

*Parsing **kern humdrum*

The ***kern* is used to represent basic or core information for period-of-common-practice Western music. The ***kern* scheme allows the encoding of pitch and duration, as well as accidentals, articulation, ornamentation, ties, slurs, etc.[11]

[11] The humdrum **kern representaton. `https://musiccog.ohio-state.edu/Humdrum/representations/kern.html`

Given the ***kern* files, the next step was to be able to retrieve information from the file - namely, *Time Signature, Key Signature, Notes and Rests, Note durations and Bar locations*. There were a few possible choices - first was The Humdrum Toolkit[12] itself, however since the library is in C++ and since the requirement was a way to encapsulate the data in Python, a decision was made against using it. Next up was music21[13], a robust Python library that is used quite a bit in the later stages of the project, it provided easy to use parsers, however when parsing all *8473* files, the performance was quite poor - *approx. 180 seconds for the entire dataset* due to the heavily modular, but unnecessary for the task, underlying boilerplate.

[12] https://musiccog.ohio-state.edu/Humdrum/

[13] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010

The last package that was found was a personal project called humpy[14] of Yuri Broze[15], which had the skeleton code which seemed to be a good candidate but required some work on top of it. Since there was no license mentioned for the package, I got in touch Yuri, who after a brief Email exchange was more than happy for me to proceed and modify the repository. The package worked by pulling in a ***kern* file and using regex to parse different elements. Following is a list of changes that were made to the repository:

[14] https://github.com/ybroze/humpy

[15] https://github.com/ybroze

- Octave processing - Octaves are quite convoluted in ***kern representation*[16], so the right functions encode pitch and the correct corresponding octave were added.
- Note to *MIDI* Number - A simple utility function to convert a note in the form of *C5* (Where the number is the octave, and letter the root note) to it's *MIDI* equivalent *72*
- Note & Rest durations - A parsing function to encode duration as a fraction of a whole note, *eg. Quarter note/rest = 0.25*
- Getters - Getter functions to retreive *Key Signature getkeysignatues(), Time Signature , Notes with duration* and *Notes (MIDI)*

[16] Middle C (C4) is represented using the single lower-case letter "c". Successive octaves are designated by letter repetition, thus C5 is represented by "cc", C6 by "ccc" and so on. For pitches below C4, upper-case letters are used: "C" designates C3, "CC" designates C2.

The humdrum **kern representaton. `https://musiccog.ohio-state.edu/Humdrum/representations/kern.html`

The new parser takes 2 *seconds* for the entire dataset.

Essen Dataset Class

What remained was a way to bring it all together as one class object that could be manipulated and used in the scripts easily,this was accomplished by creating a general *DataObject* Class which stored the following for each song:

- *wav file location*
- *MIDI file location*
- ***kern file location*
- *Parsed Kern file object*

To further facilitate the use, *Essen* Class was created as a dataset provider wrapper, which walked the directory of all the song files and creates the *Dataset Object* for each song, it also provides a *files_test* property which returns 100 files that were randomly picked from the 8473 files, they're not used for any of the training of the models and instead used for evaluations.

An example is presented below on how the class is used.

```
>>> from autoscore import datasets
>>> data = datasets.Essen()
>>> len(data.files)
8373
>>> len(data.files_test)
100
>>> data.files[0].wav
'/Users/harjyot/datasets/essen/1069/1069.wav'
>>> data.files[0].kern_score
<autoscore.score.Score object at 0x106a785f8>
>>> data.files[0].kern_score.key()
'F'
>>> data.files[0].kern_score.timesignatures()
'2/4'
>>> data.files[0].kern_score.get_notes(mono=True)[0]
{'type': 'note', 'note': 'd', 'octave': 5,
'naturals': 0, 'recip': '16' 'duration': 0.25,
'beat': 0, 'sharps': 0, 'flats': 0}
>>> data.files[0].kern_score.get_notes_midi(mono=True)[0]
74
```

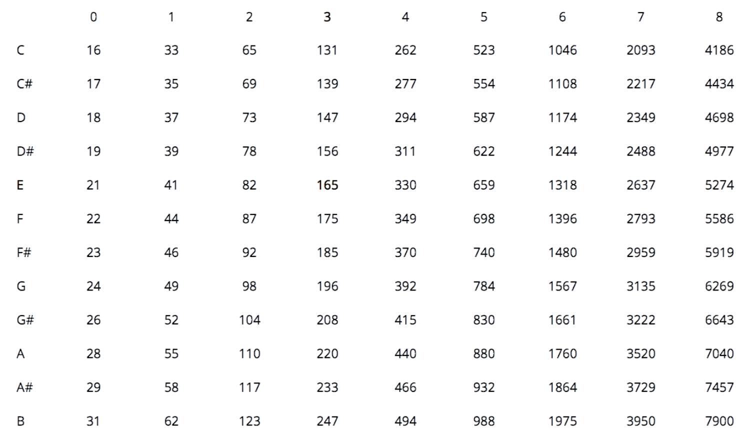
3 THE PITCH

The first step in the system is to have a model that extracts a sequence of notes with the respective onset, offset times from a digital recording. This chapter explores the field of pitch extraction, some popular methods in the literature, and the how one of these models has been optimised to be used in our system.

3.1 Fundamentals

Pitch or the *fundamental frequency (fo)* is defined as the frequency at which a sound wave oscillates (this differs from the definition in physics, but it suffices for the current requirement). *Octave* is defined as the interval between two music pitches, where one is double or half the other's frequency (For example A4 440Hz and A5 880Hz).

A musical melody comprises of a sequence of pitches played for a particular duration. The most commonly recurring frequencies are given note names ie. C, D, E etc. The chart in figure 3.1 illustrates which note in a given octave is associated with which frequency.



	0	1	2	3	4	5	6	7	8
C	16	33	65	131	262	523	1046	2093	4186
C#	17	35	69	139	277	554	1108	2217	4434
D	18	37	73	147	294	587	1174	2349	4698
D#	19	39	78	156	311	622	1244	2488	4977
E	21	41	82	165	330	659	1318	2637	5274
F	22	44	87	175	349	698	1396	2793	5586
F#	23	46	92	185	370	740	1480	2959	5919
G	24	49	98	196	392	784	1567	3135	6269
G#	26	52	104	208	415	830	1661	3222	6643
A	28	55	110	220	440	880	1760	3520	7040
A#	29	58	117	233	466	932	1864	3729	7457
B	31	62	123	247	494	988	1975	3950	7900

Figure 3.1: Music Notes and their corresponding Fundamental Frequencies

3.2 *Prior Work*

A pitch extraction method is one that correctly estimates the pitches in a digital recording and their durations, most of the pitch extraction methods can be segregated into three categories, each based on the domain they work on:

- Time domain methods - consists of looking at the input signal as a fluctuating amplitude in the time domain and to try and find repeating patterns in the waveform. Some methods are Zero crossing, Autocorrelation, Maximum likelihood, Adaptive Filtering.
- Frequency domain methods - consists of breaking the input signal into small frames, getting the short time Fourier transform (STFT) of each frame. For periodic, the STFT shows peaks where the fundamental frequencies in each frame are. Some methods for detecting these fundamental frequencies are - Harmonic Product Spectrum, Cepstrum.
- Mixed methods (utilising both time domain and frequency domain)

There have been hundreds of algorithms and methods proposed for the task of pitch extraction, while some work better for speech, others work better for musical inputs, however listing them all out is beyond the scope of this report - there are quite a few comparative studies that explore these methods[1,2].

[1] Onur Babacan, Thomas Drugman, Nicolas d'Alessandro, Nathalie Henrich, and Thierry Dutoit. A comparative study of pitch extraction algorithms on a large variety of singing sounds. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7815–7819. IEEE, 2013

[2] David Gerhard. *Pitch extraction and fundamental frequency: History and current techniques*. Department of Computer Science, University of Regina Regina, 2003

3.2.1 *Autocorrelation*

Autocorrelation is a common method in signal processing, its main goal is to find similarity between a signal and a shifted version of the given signal. The autocorrelation function for harmonic signals has peaks in multiples of the fundamental frequency of the signal. This forms the basis of it being used as a base for many pitch extraction methods.

3.2.2 *YIN*

The YIN algorithm[3] , developed by Alain de Cheveigne and Hideki Kawahara, is an algorithm to estimate the *fundamental frequency* for speech and monophonic music based on the autocorrelation method with number of modifications to mitigate error, it is named after the oriental yin-yang philosophical principal of balance, representing the authors attempts to balance between autocorrelation and cancellation in the algorithm. It has widely been accepted in the community and research as one of the most successful models (error rates are 3 times lower than competing models) for monophonic audio, with quite

[3] 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. URL `http://recherche.ircam.fr/equipes/pcm/cheveign/pss/2002_JASA_YIN.pdf`

desirable features like few parameters to tune, no upper bound on the frequency search (making it very suitable for high pitched voices and music).

3.2.3 pYIN - Probabilistic YIN

pYIN model[4] aims to correct the failings of the YIN algorithm, namely, in YIN for each frame only one fundamental frequency estimate is selected, so during the phase of generating a smoothed pitch track, the method can't fall back on alternative estimations. pYIN corrects that by outputting multiple fundamental frequency candidates (*f0*) per frame, with probabilities associated with each, and then utilising these probabilities as observations in a hidden Markov model, which is Viterbi-decoded to produce an improved pitch track. The model demonstrates a much higher recall and accuracy over original YIN. Figure 3.2 illustrates the difference.

[4] Matthias Mauch and Simon Dixon. pyin: A fundamental frequency estimator using probabilistic threshold distributions. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014)*, 2014. in press

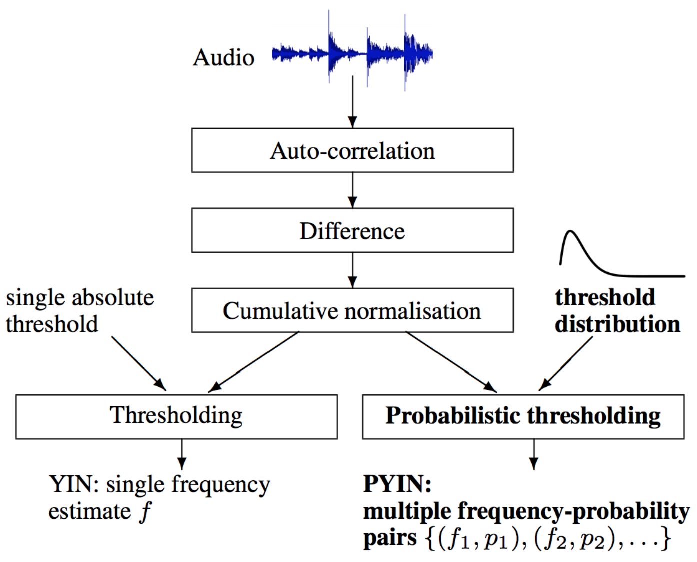

Audio
Auto-correlation
Difference
Cumulative normalisation
single absolute threshold
threshold distribution
Thresholding
Probabilistic thresholding
YIN: single frequency estimate f
PYIN: multiple frequency-probability pairs $\{(f_1, p_1), (f_2, p_2), \ldots\}$

Figure 3.2: Difference between pYIN and YIN, taken from the original pYIN paper

3.3 Optimising pYIN

The aim of the model is to find the most optimal parameters of pYIN that give the highest accuracy.

3.3.1 pYIN Parameters

The following are all the possible parameters for pYIN:

- Threshold distribution - Uniform, Beta (mean 0.10), Beta (mean 0.15), Beta (mean 0.20), Beta (mean 0.30), Single Value 0.10, Single

Value 0.15, Single Value 0.20

- Low amplitude suppression - 0 to 1 range.
- Onset sensitivity - 0 to 1 range.
- Duration Pruning threshold - 0 to 0.2 range.

3.3.2 *Input*

The input to the model is the Essen Corpus dataset.

3.3.3 *Output*

The output of the model are the best combination of the parameter values.

3.3.4 *Evaluation Metrics*

MIREX[5] provides well established metrics for estimating the performance of a pitch detection algorithm, they focus on both voicing[6] as well as pitch estimation

[5] `http://www.music-ir.org/mirex/wiki/2015:Audio_Melody_Extraction`
[6] Whether a pitch has been found in a given frame

Since an algorithm may report an estimated melody pitch even for a frame which is considered unvoiced in the actual file, the evaluation of voicing and pitch estimation must be done separately. Voicing detection is evaluated using metrics from detection theory. In the equations below, n is the vector of voicing decisions by the algorithm (1 when pitch found for a frame, 0 when not), n^* is the corresponding vector generated from the input file. Similarly, $\overline{n}$ and $\overline{n}^*$ are the complementary vectors of voicing (1 when unvoiced, 0 when voiced).

Voicing recall is defined as the proportion of frames marked as voiced by the algorithm which are also voiced in frames of the input file.

$$Recall = \frac{\sum_{f \in frames} n_f n_f^*}{\sum_{f \in frames} n_f^*} \tag{3.1}$$

Voicing false alarm rate is defined as the proportion of frames marked voiced by the algorithm which are not voiced in the frames of the input file.

$$FalseRate = \frac{\sum_{f \in frames} n_f \overline{n}_f^*}{\sum_{f \in frames} \overline{n}_f^*} \tag{3.2}$$

Pitch accuracy is estimated using pitch vectors, $freq$ for the vector of frequencies per frame generated by the algorithm, $freq^*$ for the vector of frequencies from the input file.

Raw pitch accuracy (RP) is defined as the proportion of frames with the correct pitch estimation (within half a semitone of the corresponding pitch of the input file) found by the algorithm.

$$RP = \frac{\sum_{f \in frames} n_f^* \ \tau \ [M(freq_f) - M(freq_f^*)]}{\sum_{f \in frames} n_f^*} \tag{3.3}$$

Where,

$$\tau[x] = \begin{cases} 1, & \text{if } |x| < 0.5 \\ 0, & \text{otherwise} \end{cases}$$

and

$$M(freq) = 12 \log_2(freq)$$

Raw Chroma accuracy (RC) is a measure of pitch accuracy, in which both estimated and input file pitches are mapped into one octave, thus ignoring the commonly found octave errors.

$$RC = \frac{\sum_{f \in frames} n_f^* \ \tau \ [|| \ M(freq_f) - M(freq_f^*) \ ||_{12}]}{\sum_{f \in frames} n_f^*} \tag{3.4}$$

where,

$$|| \ a \ ||_{12} = a - 12 \lfloor \frac{a}{12} + 0.5 \rfloor$$

Overall Accuracy measures the proportion of frames that were correctly labelled in terms of both pitch and voicing by the algorithm.

$$OA = \frac{\sum_{f \in frames} (n_f^* \ \tau \ [\ M(freq_f) - M(freq_f^*)] \ + \ \overline{n}_f \overline{n}_f^*)}{numberOfFrames} \tag{3.5}$$

For our metric, a decision was made to use the overall accuracy as the main criteria, with the net overall accuracy for a given run of the algorithm as:

$$AverageOA = \frac{\sum_{f \in files} OA_f}{NumberOfFiles} \tag{3.6}$$

3.3.5 Method

Rather than implementing our own version of pYIN we went with using the Vamp[7] plugin[8] developed by Matthias Mauch and Simon Dixon. The method for the model is outlined as follows:

[7] https://www.vamp-plugins.org/ - Vamp is an audio processing plugin system for plugins that extract descriptive information from audio data. We used the python wrapper around this, `https://github.com/c4dm/vampy-host`

[8] `https://code.soundsoftware.ac.uk/projects/pyin`

- A list of combinations of the above mentioned parameters are created.

- For each combination of parameters, the Essen corpus is loaded, and for each of the files in the corpus, the wav file is loaded using librosa[9], with a frame rate of *44100*, then the loaded file is fed through the Vamp implementation of pYIN, which returns the

[9] https://librosa.github.io/librosa/

fo candidates. We convert the parsed kern version of the file to get the pitches per frame, and the overall accuracy for the file is calculated using Equation 3.5. The *average overall accuracy* is calculated for all the files with the Equation 3.6

- The combination of parameters with the maximum average overall accuracy is picked.

3.3.6 Optimal Parameters

The Parameter values picked are given in the table 3.1. The overall accuracy achieved by using these parameters was *94.3*% on the training corpus.

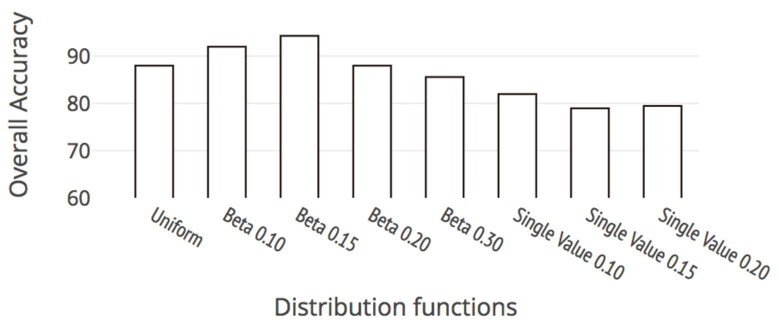


Figure 3.3: Average overall accuracy for different distribution functions, other parameters fixed

Table 3.1: Parameters Generated for pYIN

Parameter	Value
Threshold distribution	Beta (0.15)
Low amplitude suppression	0.3
Onset sensitivity	0.05
Duration Pruning threshold	0.03

3.4 The Pitch evaluator

Using the pipeline we had for evaluating the most optimal parameters, a *melody.evaluate* function was created that given a wav file, loads it through librosa, and then put it through the pYIN estimator using the optimised parameters and returns a list of pitches with their respective onset and offset times.

The pitches are then converted into their respective MIDI Numbers using the formula:

$$MIDInumber = [69 + 12\log_2(\frac{freq}{440})] \tag{3.7}$$

where, $[a]$ is the nearest integer function. The $\log_2 \frac{freq}{440}$ value is the number of octaves the frequency is above concert A4 pitch[10], multi-

[10] 440 Hz, the standard pitch of reference most commonly used

plying it by 12 gives the number of steps that the given pitch is above or below 69, which is the MIDI Number for A4.

3.5 *Evaluation*

Using the metric defined in Equation 3.5, the Essen corpus test set of 100 files was evaluated using the model, returning an Overall accuracy of *91.45*%.

It should be noted however, that since the audio files of the test set and training set are both synthesised from MIDI, the accuracy in duration of each pitch in the audio file is quite high, in a live performance however, its almost impossible for musicians to adhere to such accurate duration of pitches, hence the model might overestimate the accuracy for a real world scenario. In future work, the parameters of the model can be fine-tuned much more using live performance recordings in the training phase, making it more robust and versatile.

4 THE KEY

This chapter explores the field of Key detection, some popular methods in the literature, and the how one of these models has been implemented to be used in our system.

4.1 *Fundamentals*

To begin the explanation of Keys, one has to begin with the concept of a Scale. Scales are defined as an organised sequences of notes that generally span over an octave. In Western music, they are usually categorised by the number of pitches they comprise of - the most commonly used are *diatonic*[1] and *pentatonic*[2] scales. Of the diatonic scales, there are two that are of main importance, the *Major scale* and the *Natural Minor scale*, the way the Major[3] and Natural Minor[4] scales are different are by the respective formula of intervals[5,6] between the the consecutive notes.

[1] Diatonic Scale - Scale with 7 unique pitches

[2] Pentatonic Scale - Scale with 5 unique pitches

[3] Major scale intervals - W W h W W W h

[4] Natual Minor scale intervals - W h W W h W W

[5] Semitone or Half Step (h): Shortest interval between two notes in Western Music. eg. C to C# is one semitone

[6] Whole Tone or Whole Step (W): Two Semitones

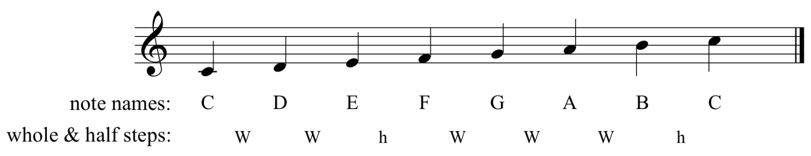


Figure 4.1: C Major Scale with intervals

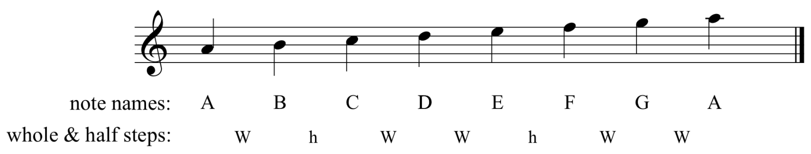


Figure 4.2: A Minor Scale with intervals

Scale degree refers to the position of a particular note on the scale relative to the main note. For a Scale of C, the scale degrees are: C - 1, C♯ - 1♯, D - 2, E♭ - 3♭, E - 3, F - 4, F ♯ - 4♯, G - 5, G♯ - 5♯ , A - 6, B♭ -

7♭, B - 7.

Hence we can define major or minor scale in terms of the notes of scale degrees that belong in them.[7,8]

[7] Major Scale consists of notes of degrees *1, 2, 3, 4, 5, 6, 7*

[8] Natural Minor scale consists of notes of degrees *1, 2, 3♭, 4, 5, 6♭, 7♭*

A *Key*, is a set of pitches that corresponds to a Scale, it provides the framework around which most of the chords and notes of the musical piece revolves. For example, a Key of C Major means that a musical piece in that key uses mostly (or only) notes from the C Major Scale. As a direct parallel to scales, the Key can be either Major or Minor. In Western Music, there are 24 Keys in total - 12 Major and 12 Minor.

Key Signature is a set of accidentals - sharps (♯) or Flats (♭) that is used in staff notation to signify which particular natural notes[9] are altered in the musical piece. This is directly used to find the Key of the musical Piece. A common tool used by most musicians is the circle of fifths (Figure 4.3) which shows each of the Major Key (Outer Circle) and Minor Keys (Inner Circle) with the responding Key Signature, it can be seen that every Major Key has a corresponding Minor Key that shares the same Key Signature, these Minor Keys are called *Relative Minor Key*[10].This leads to an interesting side effect of using the *Key Signature* - without knowing the chords of a particular piece its hard to decide whether the Key of the piece is Major or Minor.

[9] Natural Notes are C D E F G A B

[10] Relative Minor Key - The minor key which shares the same key signature as the Major Key but a different root note

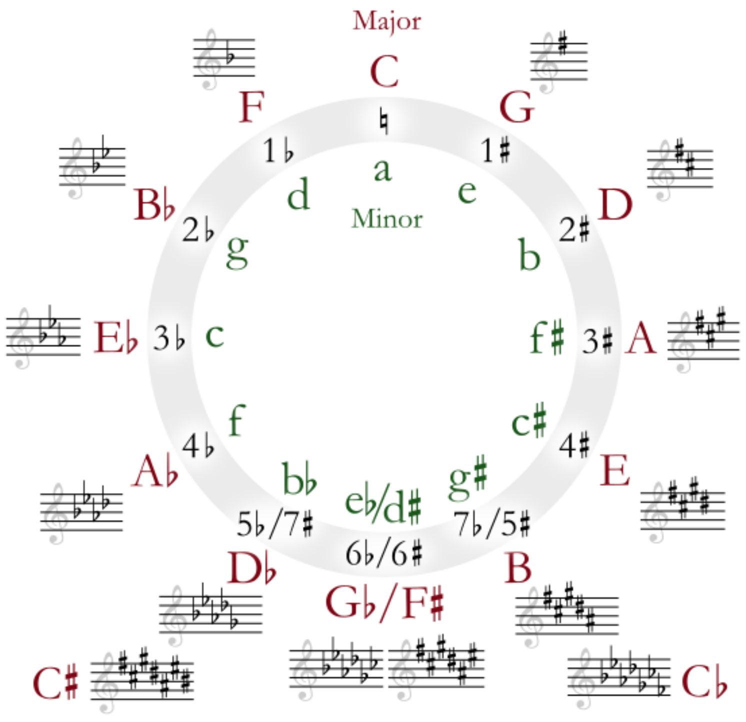


Figure 4.3: Circle of Fifths - Key Signatures of Major & Minor Keys

4.2 *Prior Work*

The key is a core element in writing a music piece in Western music, this is pretty evident from the literature, where discussions about the music piece, its notes and chords are usually done in relation to the key. It's no surprise then that a lot of research has been done on the question of *inferring a key*. Here I will list a few of the most important models that have been put forth.

Longuet-Higgins and Steedman (1971)

One of the first models to find Key in Monophonic audio, Longuet-Higgins and Steedman's model[11] is based on the relationship between Keys and Scales. The model works by analysing notes left to right, iteratively removing keys which do not contain that note, until only one Key is remaining, which is chosen as the Key for the melody. In cases when all keys have been eliminated or more than one Key remains, the Key where the root note is the first note of the melody is selected.

[11] H Christopher Longuet-Higgins and Mark J Steedman. On interpreting bach. *Machine intelligence*, 6:221–241, 1971

The model had some obvious loopholes -

- Since Minor scales can be either Harmonic, Melodic or Natural, the relationship with the Key is less well defined.
- Accidental Notes couldn't be taken into account.
- It had trouble differentiating between Major and Relative Minor Keys (They have the same notes)

Krumhansl-Schmuckler (K-S) key-finding algorithm

A breakthrough model based on the principle of *Key Profiles* was proposed by Krumhansl and Schmuckler in the paper[12]. Key Profiles are a 12 valued vector which shows the proportion of occurrence of certain scale degrees in Keys, The Key Profiles were generated by experimentation by by Krumhansl and Kessler[13] where they came up with Two Key Profiles, one for Major Keys and the other for Minor Keys.

[12] Carol L Krumhansl. Cognitive foundations of musical pitch. 1990

[13] Carol L Krumhansl and Edward J Kessler. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological review*, 89(4):334, 1982

Their model operated by converting a melody into a 12 valued vector, noting the occurrence of each scale degree, then calculating the correlation value (Equation 4.1) with the Major and Minor Key profiles they had calculated from experiments (mentioned above). The key with the maximum correlation value was picked as the most likely Key.

$$r = \frac{\sum (x - \mu_x)(y - \mu_y)}{(\sum (x - \mu_x)^2 \sum (y - \mu_y)^2)^{1/2}} \tag{4.1}$$

K-S Variants

The success of K-S model led to a lot of research into the usage of Key Profiles, with a lot of variants of the initial model popping up. Some tried to explore on how to handle modulation[14]. In the model[15] proposed by Shmulevich and Yli-Harja, the piece is divided into frames and the K-S method is used to find Keys for the particular frame, these frames are then grouped together to form sections and the average position of all frames in each section is mapped on a 4-dimensional Key space, the key closest to this average is chosen.

[14] Change of key in between parts of the melody

[15] Ilya Shmulevich and Olli Yli-Harja. Localized key finding: Algorithms and applications. *Music Perception: An Interdisciplinary Journal*, 17(4):531–544, 2000

The *CBMS Model*[16] proposed by Temperley, made alterations to the Key Profile values, it also worked by dividing a melody into segments (handling modulation) of input vectors, in which each pitch-class got a 1 if it was present and 0 if it was not. To calculate a match for each segment, those values in the key profile that scored a 1 were added. For each segment if the key varied, a penalty was added, these penalties were combined with the key profile scores to get an overall score, the Key with the highest score was picked as the most likely one.

[16] David Temperley. *The cognition of basic musical structures*. MIT press, 2004a

Other works and discussion

Most of the works above are distributional in nature - in the sense that they don't rely on time-based placements and patterns formed due to that, some have argued that taking those properties into account are important [17], also since a choice was made in the beginning of the project to not work with acoustic signals except in the pitch extraction phase, the models proposed that use such aren't mentioned.

[17] Helen Brown, David Butler, and Mari Riess Jones. Musical and temporal influences on key discovery. *Music Perception: An Interdisciplinary Journal*, 11(4):371–407, 1994

4.3 *Model*

The chosen model has been adapted from *David Temperley's*[18] book *Music and Probability*[19] and his paper *A Probabilistic Model of Melody Perception.*[20]. The model uses Bayesian reasoning to infer the most probable Key, given a sequence of pitches.

[18] http://davidtemperley.com/

[19] David Temperley. *Music and probability*. Mit Press, 2007

[20] David Temperley. A probabilistic model of melody perception. *Cognitive Science*, 32(2):418–444, 2008

Input

The Input to the model is the ordered sequence of pitches in the form of MIDI notes, without any duration, onset or offset information.

Output

The Most Probable Key for the melody.

Principles

Pitches used generally in most melodies tend to be quite confined in range, as can be seen from 4.4 most of the pitches form a bell-shaped curve around pitch *70 (MIDI Note)*. In addition for a given

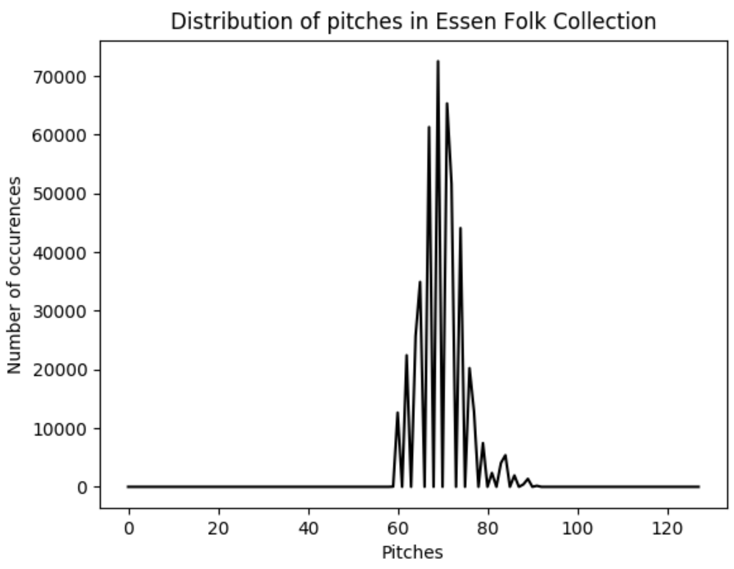


Figure 4.4: Distribution of Pitches in the Essen Folksong Collection

melody, most of the pitches tend to use notes out of a given Scale, that is, if one note is picked as the central note, the probabilities of occurrence of some notes of particular scale degrees outweigh the others. For example given a melody is in C Major, the probability of a note E♭ (Scale degree 3♭) occurring is much less than that of D (Scale degree 2), Whereas, for a melody in C Minor, the probability of a E♭ occurring is more than the that of D.

Also, an observation that has come up widely in literature is the phenomenon of intervals between adjacent notes of a melody tend to be quite small, ie. the concept of *Pitch Proximity*. It has been demonstrated as a statistical tendency in actual melodies[21] and as a preference in perception of audio[22,23].

Parameters

These principles are incorporated in the model with the following parameters:

- *Central Pitch Profile* - A Normal Distribution[24] based on the most frequent pitches in the corpus. This is used to randomly pick a central pitch c for the melody.
- *Range Profile* - A Normal Distribution centred around the given central pitch (c) of the melody. A melody can be generated around

[21] Paul Von Hippel and David Huron. Why do skips precede reversals? the effect of tessitura on melodic structure. *Music Perception: An Interdisciplinary Journal*, 18(1):59–85, 2000

[22] George A Miller and George A Heise. The trill threshold. *The Journal of the Acoustical Society of America*, 22(5):637–638, 1950

[23] Diana Deutsch. Grouping mechanisms in music. In *The Psychology of Music (Third Edition)*, pages 183–248. Elsevier, 2013

[24] A normal distribution is a continuous probability density function where the probability of x given mean μ and standard deviation σ is,

$$P(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$$

a central pitch from the notes in its range profile.

- *Proximity Profile* - A Normal Distribution centred around a given pitch, indicating the probability of note that follows it.
- *Key Profiles* - A 12 valued vector, noting the probability of scale-degrees for a major scale and a minor scale. This is repeated over many octaves.
- *RPK Profile* - Product of Range, Proximity and Key Profiles.
- *Probability of Major or Minor* - Probability for Key to be Major or Minor

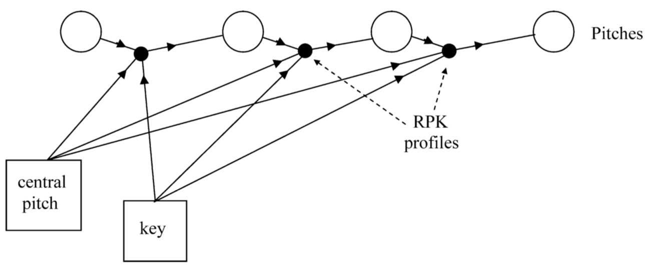


Figure 4.5: Graphical Representation of the Model [Temperley, 2008]

The Process

For each Key, k:

- For each value of Central Pitch, c.
- The probability of the melody given the k and c is the product of probability of k, probability of c and the probabilities of each note in the melody - the probability of each note is its *RPK* value, which is the normalized product of its *Range Profile value* (using c), its *Proximity Profile* (using the previous pitch) and its *Key Profile* value (given k).

The Probability of k is the sum of all probabilities for all value of c.

Mathematically the probability of k given a sequence of notes *pitchsequence* is,

$$P(k|pitchsequence) \propto P(k \cap pitchsequence)$$

$$P(k \cap pitchsequence) = \sum_c P(pitchsequence \cap k \cap c)$$

$$P(k \cap pitchsequence) = \sum_c (P(k)P(c) \prod_n RPK_n) \tag{4.2}$$

The key with the maximum probability is picked as the key for the melody ie,

$$Key = \arg\max_k P(k|pitchsequence) \tag{4.3}$$

4.4 *Implementation of the Model*

The Class *KeyModel* was implemented to be the trained model that we utilize in evaluating the Key for a given melody. It was given the *Essen object* as the training corpus, each melody's notes are used in the MIDI Number notation.

Estimation of Parameters

The *Central Pitch Profile* Normal Distribution required the *Mean* and *Variance* of notes which was found by iterating over all the notes of all the melodies in the Essen corpus.

Since the first note of each melody does not depend on the Proximity Profile, the Variance for the Range Profile is calculated as the *Mean of all the variances of the first note of the melody with the respective central pitch of the melody* ie

$$\sigma^2_{RangeProfile} = \frac{\sum_{melody \in Essen} (melody.firstnote - \mu_{melody})^2}{\sum_{melody \in Essen} 1}$$

The Estimation for the variances for the *Proximity Profile* Normal Distribution was done by the method outlined in the notes section of Music and Probability[25].

[25] David Temperley. *Music and probability*. Mit Press, 2007

- For each of the melodies, the variances of all those pitches in it where the previous note was the central pitch of the melody was taken ie, for each melody m its variance is given by:

$$\sigma^2_m = \frac{\sum_{n \in m, n-1=\mu_m} (n - \mu_m)^2}{\sum_{n \in m, n-1=\mu_m} 1}$$

- The mean of all those variances was taken, ie

$$\sigma^2_t = \frac{\sum_{m \in Essen} \sigma^2_m}{\sum_{m \in Essen} 1}$$

- The Proximity Profile variance was estimated by the formula:

$$\sigma^2_{ProximityProfile} = \frac{\sigma^2_{RangeProfile} \sigma^2_t}{\sigma^2_{RangeProfile} - \sigma^2_t}$$

For the *Key Profiles* The occurrences of each scale-degree in each melody of the Corpus was counted; Then summed these counts over all melodies, grouping Major Keys and Minor Keys separately, and expressed the totals as proportions. The outputs are shown graphically Figure 4.6 and Figure 4.7, it illustrates the relationship tendencies as expected in major and minor keys.

Figure 4.6: Key Profile for Major Keys

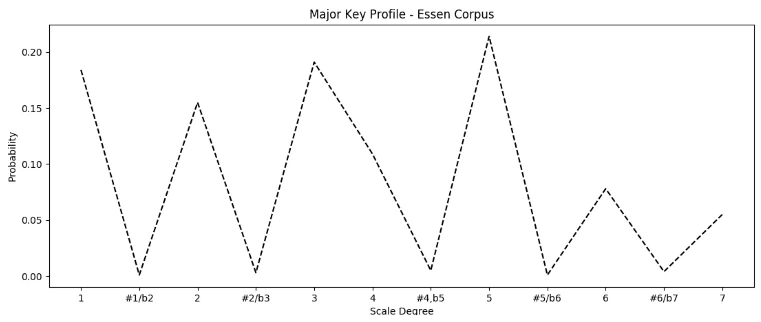
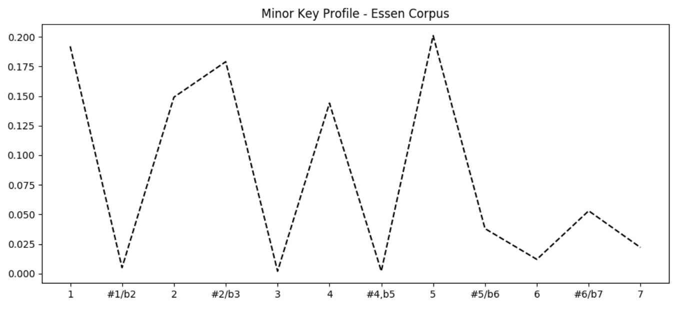


Figure 4.7: Key Profile for Minor Keys



[26] David Temperley. A probabilistic model of melody perception. *Cognitive Science*, 32(2):418–444, 2008

In his paper[26], Temperley suggested using a bias for the probability of a Major Key (0.88) vs a Minor Key (0.12), however that was evaluated as over-fitting to the corpus, and decision was made to give them equal probabilities (0.5). The parameters generated from the corpus are given in the table 4.1.

Table 4.1: Parameters Generated for the Key-finding Mode

Parameter	Value
Central Pitch Profile	*mean* = 69.0, *variance* = 16.17
Range Profile	*variance* = 25.24
Major Key Profile	{ 0.184, 0.001, 0.155, 0.003, 0.191, 0.109, 0.005, 0.214, 0.001, 0.078, 0.004, 0.055 }
Minor Key Profile	{ 0.192, 0.005, 0.149, 0.179, 0.002, 0.144, 0.002, 0.201, 0.038, 0.012, 0.015, 0.060 }
Proximity Profile	*variance* = 8.76
P(Key)	(1/12) * 0.5

Memoisation

Looking at the model, it was noticed that often for evaluation, the same values are going to be computed again and again, decision was made to utilise the principle of *Memoisation*[27].

[27] Memoisation is an optimisation technique of storing results of expensive function calls

MIDI notes exist in a range of 0 (C-1) and 127 (G9), that showed that there are only a finite number of note combinations that need to be calculated. We begun by calculating *Central Pitch Profile table*

(Shape 128) which was the value of the probability of all notes within the MIDI range, given the mean and variance parameters of the Central Pitch Profile Normal Distribution [28]. Similarly for all possible central notes, The probability of a given note occurring was calculated as the *Range Profile Probability Table (Shape (128 x 128))* using the Range Profile Normal Distribution, and for all notes, the probability of a previous note occurring was calculated as the *Proximity Profile Probability Table (Shape (128 x 128))* using the Proximity Profile Normal Distribution.

[28] Scipy - https://docs.scipy.org/doc/, a versatile package for Science and Mathematics. *scipy.stats.norm* was used for calculation of all normal distributions.

These tables were used to generate the KeyValues dictionary, which essentially stored values of probability of a note occurring, given the previous note, the central pitch, the key type (major or minor), and the Key root note. The process is demonstrated using the psuedocode (Figure 4.8). Note about the *shiftKP function* [29]

[29] The Key root note was essentially used to shift the Key Profile across the MIDI number range and generate a 'shifted' Key-Profile. For eg. for the Key of C Major, MIDI numbers 0, 12, 24 would have the values of degree 1 from Major Key Profile, where as for a Key of D Major, MIDI Numbers 2, 14, 26 would have the values of degree 1 from Major Key Profile

```
$params \leftarrow dict()$
$notes \leftarrow [C, C\sharp, D, E\flat, E, F, F\sharp, G, G\sharp, A, B\flat, B]$
for each $root \in notes$ do
    $params[root] \leftarrow dict()$
    for each $type \in ['maj', 'min']$ do
        $params[root][type] \leftarrow dict()$
        if $type = maj$ then
            $kp \leftarrow keyprofileMajor$
        else
            $kp \leftarrow keyprofileMinor$
        end if
        $kp \leftarrow shiftKP(kp, root)$
        for $c \in \{0, \dots, 127\}$ do
            $params[root][type][c] \leftarrow dict()$
            for $note \in \{0, \dots, 127\}$ do
                $params[root][type][c][note] \leftarrow dict()$
                for $prev_{note} \in \{-1, \dots, 127\}$ do
                    $rpk \leftarrow []$
                    for $i, p \leftarrow enumerate(kp)$ do
                        $rpk[i] \leftarrow p * rangeTable[i][c] * proximityTable[i][prev_{note}]$
                    end for
                    $rpk_{normalised} \leftarrow normalise(rpk)$
                    $params[root][type][c][note][prev_{note}] \leftarrow rpk_{normalised}[note]$
                end for
            end for
        end for
    end for
end for
$KeyValues \leftarrow params$
return $KeyValues$
```

Figure 4.8: Algorithm for Memoisation of Key Values

Using this memoised table, to evaluate a given melody's Key, the probabilities are looked up, summed and multiplied according to the Equation 4.2 and the Key with the highest Probability is picked as the most likely Key (Equation 4.3).

This Trained KeyModel Class Object was stored using *Pickle Library*[30], and is loaded up to evaluate a new melody's Key. The Memoisation Process speeded up the process of finding the Key from **8.2 mins** to **300ms**

[30] `https://docs.python.org/3.6/library/pickle.html`

4.5 *Evaluation*

We test the model on the 100 test files of the Essen dataset corpus. The model picked *78 out of 100* keys correctly. After manually observing the results it was noted that a majority of the model's wrong calculations was because the relative key was picked instead (For eg. A minor in place of C major). Since for the task of transcription both a major key and its relative minor are represented as the same (Key Signature) in the music notation, we changed the definition of 'right' key as including the key's relative key as well. After running the model again with this modified metric, it picked *92 out of 100* keys correctly. It should also be noted that this error arises due to lack of accompanying chords in the musical piece, which usually emphasises the difference between a major and its relative minor key.

Out of the rest 8, 5 were *modal melodies*, ie melodies in which the most predominant note is not the tonal center of the picked key, but all notes of the melody are from the key's scale. From this observation, one should note that picking the right key for modal melodies is inherently complicated because these melodies enforce the key by repetition of a certain pitch in structurally important places, and is often ambiguous.

The key model could hence be improved in the future by giving weights to notes in structurally important places in the melody and factoring that in when deciding on the Key, also further features should be incorporated in terms of detecting Key changes between different parts of the melody (ie. modulation).

5 THE BEAT

This chapter explores the field of meter detection, some popular methods in the literature, and the how one of these models has been extended to be used in our system.

5.1 Fundamentals

The Beat or The Tactus is the the basic of unit of time in a musical piece. It is most commonly described as the event at which a listener would tap their foot to.

5.1.1 Note Durations

A note along with its pitch has an important property, the *duration*. In Western music, these durations are named as relative to the Whole Note. So a Whole note consists of 2 Half-notes, a Half-Notes consists of 2 Quarter Notes and so on. The further subdivisions and their equivalent symbols in staff-notation is shown in the Figure 5.1. A

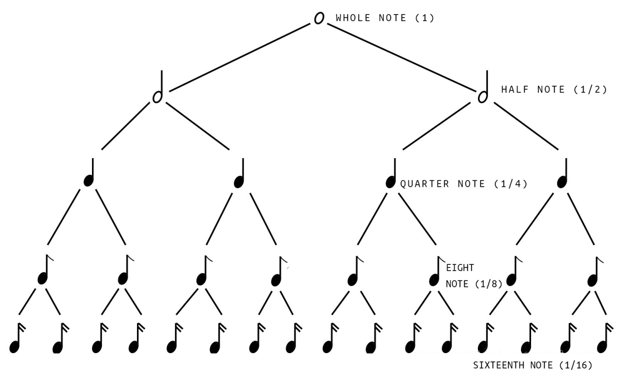


Figure 5.1: Relative Note Durations and their staff symbols (US Naming)

dotted note is a note with its duration increased by half of its original duration[1]

[1] So a Dotted-Quarter note's duration would be $\frac{1}{4} + \frac{1}{8}$

5.1.2 Time Signature

Beats are divided into smaller segments called *Bars* or *Measures*. The *Time Signature* for a musical piece essentially conveys how many beats comprise a bar. In Western Music, its donated as a fraction of the form 4/4, the lower number represents *which note type* represents the beat - 4 would mean Quarter Note, 8 Would mean an Eight Note and so on - and the upper number represents *how many of those notes* constitute a bar. [2]

[2] As an example, a Time Signature of 3/4 would mean each bar contains 3 Quarter Notes, whereas a Time Signature of 6/8 would mean each bar contains 6 Eight Notes

The Time Signature can be classified into two categories:

- *Simple* - Where the tactus or the beat is divided into two sub beats. In this case the beat is a Quarter note eg. 3/4, 4/4, etc.
- *Compound* - Where the tactus or the beat is divided into three sub beats. In this case the beat is a Dotted-Quarter note eg. 6/8, 9/8 etc.

5.1.3 Tempo

Tempo is defined as the number of quarter notes per minute.

5.2 Prior Work

As was discussed in the Introduction chapter, meter plays a fundamental role in the process of music transcription, the problem is multi-faceted in terms of solving the following sub-problems:

- Inferring the right metrical grid (The organisation of beats on different levels)
- Quantising the note pattern to fit the metrical structure ie, which note lasts for how long based on the metrical structure.

Both are required for a complete model of meter detection. There has been an abundance of research in terms of meter perception, with important contribution over the past 4 decades, however to list all the models is beyond the scope of this report, but if the reader is interested, there's a good comparison paper[3] that compares and (tries) to give a brief overview of 25 of the most popular computational models.

[3] David Temperley. An evaluation system for metrical models. *Computer Music Journal*, 28(3):28–44, 2004b

The computational models can be categorised in multiple ways - either Symbolic (notes, onset - offset times identified) vs Audio (working directly from raw audio signals), with further subdivision of symbolic into quantised vs non-quantised, or based on the approach taken -

- A rule based approach, building beats by analysing notes in a left to right manner, as in the model proposed by Longuet-Higgins and Steedman[4] or Lee[5].
- A connectionist approach, where nodes in a neural network represent beat values as in the model[6]
- An Oscillator based approach where an oscillator tries to synchronise with an input pattern[7]
- A constraint satisfaction approach, where the whole input is analysed multiple times with different outputs and the one satisfying certain constraints is picked[8,9]

[4] H Christopher Longuet-Higgins and Mark J Steedman. On interpreting bach. *Machine intelligence*, 6:221–241, 1971

[5] Christopher S Lee. The perception of metrical structure: Experimental evidence and a model. *Representing musical structure*, pages 59–127, 1991

[6] Peter Desain and Henkjan Honing. Computational models of beat induction: The rule-based approach. *Journal of new music research*, 28(1):29–42, 1999

[7] Edward W Large and John F Kolen. Resonance and the perception of musical meter. *Connection science*, 6(2-3):177–208, 1994

[8] David Temperley and Daniel Sleator. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23(1):10–27, 1999. DOI: 10.1162/014892699559616. URL `https://doi.org/10.1162/014892699559616`

[9] Dirk-Jan Povel and Peter Essens. Perception of temporal patterns. *Music Perception: An Interdisciplinary Journal*, 2(4): 411–440, 1985

Some models in the realm of Probabilistic modelling however require a bit more of a mention -

In the model[10], the method of vector quantisation is used where, multiple scores (various combinations of time signatures) are created, within each score each note is given a fractional label, quantifying its position in a measure (eg. 2/3 for a quarter note in 3/4 time signature occurring at the second beat of a bar), and the goal is to determine *P(score | note onsets)* given *P(note onsets | score)P(score)*, the *P(note onsets | score)* is calculated using experimental data. The model has quite a few drawbacks, but the biggest one is that it can only detect rhythm patterns for very short short number of note onsets (4).

[10] Ali Taylan Cemgil, Bert Kappen, and Peter Desain. Rhythm quantization for transcription. *Comput. Music J.*, 24(2):60–76, July 2000a. ISSN 0148-9267. DOI: 10.1162/014892600559218. URL `http://dx.doi.org/10.1162/014892600559218`

In another model[11] presented by Cemgil and his team, a beat-tracking system is used to derive a single level of tactus beats, in which the the 'tactus track' maximising the Bayesian model of *P(tactus track | note onsets)* is picked as the most probable one. The prior probability of the tactus track ie *P(tactus track)* is estimated using the technique of Kalman filtering, which essentially favours evenly spaced beats, and *P(note onsets | tactus track)* is estimated as the product of a smoothed note onset track and the pulses of the tactus track. However this model doesn't present any information about the metrical grid except the tactus level, which would be required in the task of inferring Key signature.

[11] Ali Taylan Cemgil, Bert Kappen, Peter Desain, and Henkjan Honing. On tempo tracking: Tempogram representation and kalman filtering. *Journal of New Music Research*, 29(4):259–273, 2000b

5.3 *Melisma 2 Model*

The model that was decided on was Temperley's Melisma v2[12], which is a probabilistic interpretation of the model (Temperley and Sleator)[13]. The model combines three aspects of music analysis - metrical structure, harmonic structure[14] and stream segregation [15].The reason this model was picked, even though we do not require stream segregation or harmonic analysis is because of prior evidence in literature which strongly suggests the influence of harmonic properties of a melody on the metrical structure for eg. tendency for harmonic changes to happen on strong beats.

[12] David Temperley. A unified probabilistic model for polyphonic music analysis. 38, 03 2009

[13] David Temperley and Daniel Sleator. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23(1):10–27, 1999. DOI: 10.1162/014892699559616. URL `https://doi.org/10.1162/014892699559616`

[14] A harmonic structure is a segmentation of a piece into time-spans labelled with root note of chords

[15] a stream structure is a grouping of the notes of a polyphonic texture into melodic lines or voices

5.3.1 *Input*

The Input to the model is the Note (in MIDI Number) with onset and offset time (In milliseconds).

5.3.2 *The Output*

The Output of the model is the notes with aligned beat onsets, offsets, the stream it belongs to, and the harmonic chords.

5.3.3 *The Generative Process*

The generative aspect of the model is to create joint structures (see Figure 5.2) with probabilities (Equation 5.1) consisting of - metrical structure, a harmonic structure, a stream structure in combination with pattern of notes aligned to these structure,

$$P(M, H, S, N) = P(N|M, H, S) * P(S|M) * P(H|M) * P(M) \qquad (5.1)$$

The metrical grid consists of 4 levels (This means the model at a min-

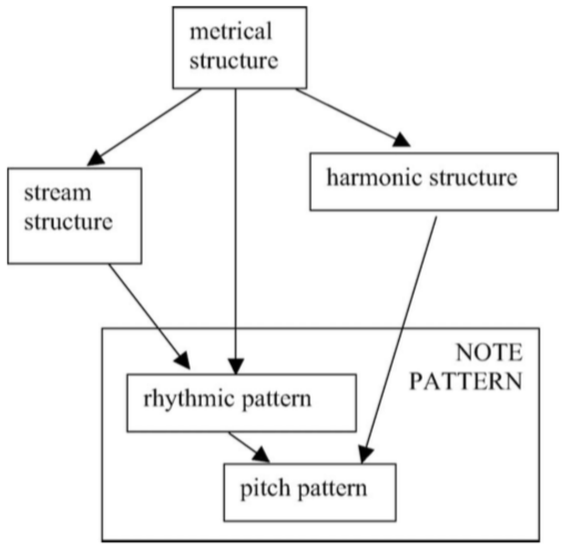


Figure 5.2: The Structure of the generative process (Image taken from the original paper)

imum detects 1/16th beat), Level 2 (L2) is taken as the *Tactus level*, the generation of the metrical grid goes as follows (also demonstrated in Figure 5.3):

- First tactus interval is set using a distribution which is biased towards intervals between 600ms - 800ms.
- Subsequent tactus intervals are generated using a distribution based on the previous tactus interval, favouring regular intervals.

- Decisions are made whether Level 3(L3) should be duple[16] or triple[17] as well as whether L2 is duple or triple.[18]
- Exact placement of beats at Level 1 and Level 0 are found using distributions favouring roughly equal higher level beat intervals.
- The phase of L3 is decided as whether the first L2 beat is the first, second or third (only in case of triple L3) of the L3 beat interval.

[16] Every 2nd Lower level beat is a beat in the current level

[17] Every 3rd Lower level beat is a beat in the current level

[18] L1 is always duple since triple L1 is extremely rare in Western Music

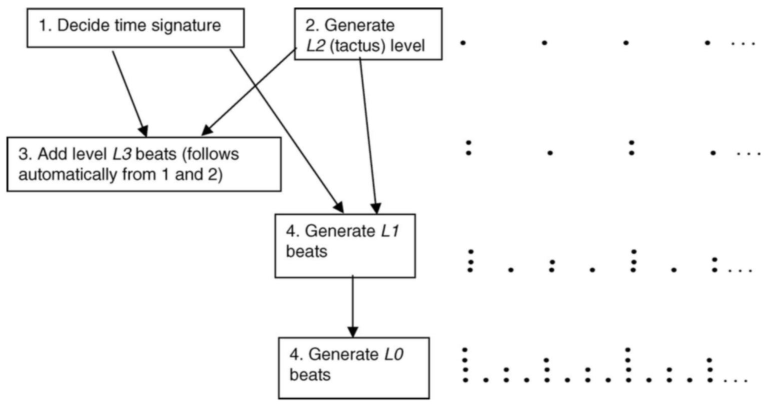

1. Decide time signature
2. Generate L2 (tactus) level
3. Add level L3 beats (follows automatically from 1 and 2)
4. Generate L1 beats
4. Generate L0 beats

Figure 5.3: The Metrical structure generation process (Image taken from the original paper)

After the generation of the Metrical Structure, the Harmonic structure is generated which segregates the piece into spans with the common root note chord, this is done by essentially picking a 'root pitch' for each tactus interval:

- For the first tactus interval the root is selected from a normal distribution.
- For subsequent intervals, a decision is made as to whether continue the same root or change to a new root, the probability of change on an L3 beat is higher than that of an L2 beat. The probability of moving to a new root is biased towards favouring roots that are a perfect-fifth(7 semitones) from the current root.

The task for stream structure is to generate a set of streams spanning a particular duration, it takes place as follows:

- At each tactus beat a Poisson distribution[19], with an expected value $<<<1$, is used to generate n possible new streams.
- After a stream is generated, at each subsequent tactus beat, decision is made whether to continue the stream or not from a distribution.

[19] $P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$

After the above structures have been generated, the note-onset pattern is generated as follows:

- For each stream, at each pip[20], decision is made to generate a note onset or not, with a very low probability given to non-pip note onsets (For the odd 1/32nd note), the probability is derived from a method called *Metrical Anchoring*[21,22] The process goes in a top down manner, from L3 to L0 beat, at each level considering the neighbouring notes at the higher level to decide whether the beat gets a note onset or not.

[20] level 0 beat

[21] David Temperley. *Music and probability*. Mit Press, 2007

[22] Metrical Anchoring is based around the idea that the probability of a note-onset at a beat depends on the presence of notes (of the same stream) at the surrounding higher-level beats. This indirectly incorporates the preference for longer notes at stronger beats.

- After all note onsets have been generated, for each note onset at each subsequent tactus beat, a stochastic decision is made whether to end the note at that beat or not, but if the next note onset occurs at that tactus beat or before, the note is ended where the next note onset begins.

Utilising this note-onset pattern, for each note onset, a pitch is selected from a joint normal distribution based on the current and previous pitch (Range Profile) and Chord Profile.[23]

[23] Heavily biased towards notes that form chord of the current harmony, as well as slightly biased towards notes that are part of the major and minor scale of the current harmonic root.

5.3.4 *The Analytical Process*

Using the generative process above the model evaluates the joint structure of meter,harmony, stream with the maximum probability a given note pattern (the input) (See Equation 5.2). Going through all possible combinations of the structures generated from the above process is barely computational, and Temperley has proposed certain ways of handling that in his paper [24], However the details of how this is accomplished in the model is beyond the scope of this report.

[24] David Temperley. A unified probabilistic model for polyphonic music analysis. 38, 03 2009

$$\{M, H, S\} = \arg\max_{M,H,S} P(M, H, S, N) \tag{5.2}$$

5.4 *Extending the Model*

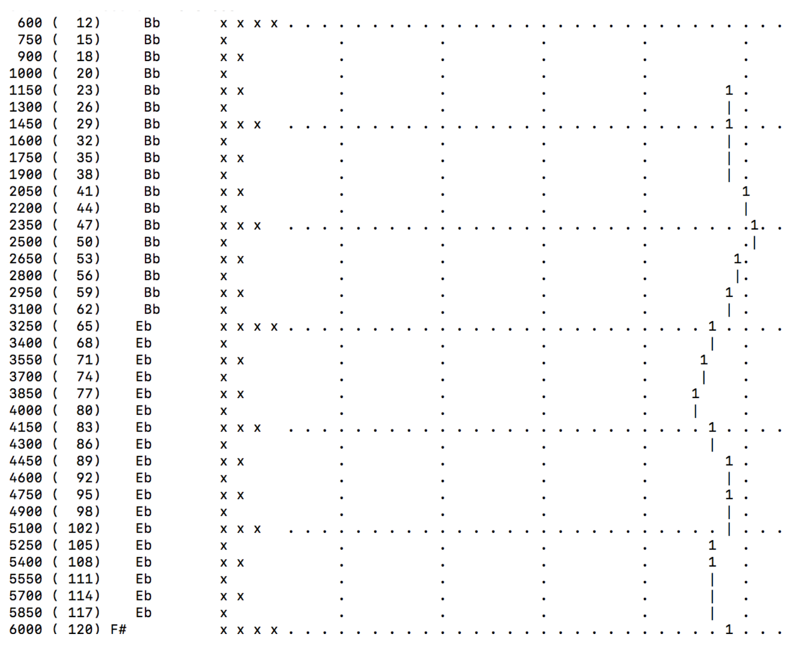
After getting in touch with Temperley, he recommended using the C Implementation of the Melisma Model[25], that he had written for the paper, which he was very kind to send. The implementation follows the analytical process which is highlighted in the paper, and essentially uses parameters trained on the Essen Corpus itself, which saved resources from trying to recalculate the parameters (about 50! for the model).

[25] David Temperley. A unified probabilistic model for polyphonic music analysis. 38, 03 2009

Modifying the Melisma Output

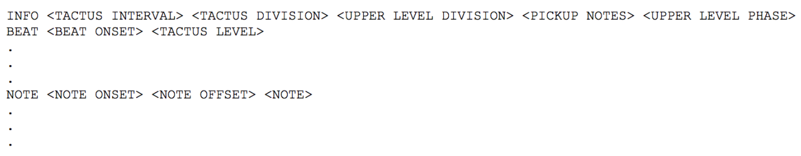
Firstly the requirements for our Meter Program were much more limited than that of the actual intent of the Melisma model. Since the audio is known to be Monophonic we were not interested in the Harmonic Analysis output (No chords) or the Stream differentiation

(Monophonic by its virtue is single stream).The original model displayed the output graphically (see Figure 5.4), so the C code was modified to output the in a stream manner that could be on boarded to the application, the output format is shown in Figure 5.5.



```
 600 (  12)     Bb       x x x x . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 750 (  15)     Bb       x             .           .           .           .           .
 900 (  18)     Bb       x x           .           .           .           .           .
1000 (  20)     Bb       x             .           .           .           .           .
1150 (  23)     Bb       x x           .           .           .           .         1 .
1300 (  26)     Bb       x             .           .           .           .         | .
1450 (  29)     Bb       x x x   . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . .
1600 (  32)     Bb       x             .           .           .           .         | .
1750 (  35)     Bb       x x           .           .           .           .         | .
1900 (  38)     Bb       x             .           .           .           .         | .
2050 (  41)     Bb       x x           .           .           .           .           1
2200 (  44)     Bb       x             .           .           .           .           |
2350 (  47)     Bb       x x x   . . . . . . . . . . . . . . . . . . . . . . . . . . . .1. .
2500 (  50)     Bb       x             .           .           .           .           .|
2650 (  53)     Bb       x x           .           .           .           .          1.
2800 (  56)     Bb       x             .           .           .           .          |.
2950 (  59)     Bb       x x           .           .           .           .         1 .
3100 (  62)     Bb       x             .           .           .           .         | .
3250 (  65)    Eb        x x x x . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . .
3400 (  68)    Eb        x             .           .           .           .       |   .
3550 (  71)    Eb        x x           .           .           .           .      1    .
3700 (  74)    Eb        x             .           .           .           .      |    .
3850 (  77)    Eb        x x           .           .           .           .     1     .
4000 (  80)    Eb        x             .           .           .           .     |     .
4150 (  83)    Eb        x x x   . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . .
4300 (  86)    Eb        x             .           .           .           .       |   .
4450 (  89)    Eb        x x           .           .           .           .         1 .
4600 (  92)    Eb        x             .           .           .           .         | .
4750 (  95)    Eb        x x           .           .           .           .         1 .
4900 (  98)    Eb        x             .           .           .           .         | .
5100 ( 102)    Eb        x x x   . . . . . . . . . . . . . . . . . . . . . . . . . . | . . .
5250 ( 105)    Eb        x             .           .           .           .       1   .
5400 ( 108)    Eb        x x           .           .           .           .       1   .
5550 ( 111)    Eb        x             .           .           .           .       |   .
5700 ( 114)    Eb        x x           .           .           .           .       |   .
5850 ( 117)    Eb        x             .           .           .           .       |   .
6000 ( 120) F#           x x x x . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . .
```

Figure 5.4: Original Output format for the Melisma Model



```
INFO <TACTUS INTERVAL> <TACTUS DIVISION> <UPPER LEVEL DIVISION> <PICKUP NOTES> <UPPER LEVEL PHASE>
BEAT <BEAT ONSET> <TACTUS LEVEL>
.
.
.
NOTE <NOTE ONSET> <NOTE OFFSET> <NOTE>
.
.
.
```

Figure 5.5: Modified Output format for the Melisma Model

Running Melisma from Python

The *meter* package was created to have all the utilities required to run the Melisma program. The *subprocess*[26] module in Python standard library allowed the creation of new processes and connect to the input and output pipes. This was used to execute the Melisma executable,and store the output into python.

[26] `https://docs.python.org/3/library/subprocess.html`

Calculating Note Durations

Next a simple parser was written which essentially processed the output line by line, Using the tag word of the line (INFO, BEAT, NOTE) and creating the appropriate Python data structures with it, the first line of the output was stored as the *Info dictionary*, then the subsequent Beat onsets and tactus levels in the *Beat List* and the Note onsets, offsets and Note MIDI number in the *Note List*.

Given the modified notes with the beat onsets and offsets, and the tactus at each beat the pseudocode in Figure 5.6 was used to calculate the number of Level 0 beats ($1/16^{th}$ notes) each of the notes lasted for, it also found gaps where no notes existed and marked them as rests.

Figure 5.6: Note duration pseudocode

```
$index_{note} \leftarrow 0$
$index_{beat} \leftarrow 0$
$padding \leftarrow 0$
$notesAndRests \leftarrow []$
while $notelist[index_{note}][onset] > beatlist[index_{beat}]$ do
    $index_{beat} \leftarrow index_{beat} + 1$
    $padding \leftarrow padding + 1$
end while
while $index_{note} < len(notelist)$ and $index_{beat} < len(beatlist)$ do
    $note_{current} \leftarrow notelist[index_{note}]$
    $ticks \leftarrow 0$
    $rests \leftarrow 0$
    while $notelist[index_{note}][onset] > beatlist[index_{beat}]$ do
        $rests \leftarrow rests + 1$
        $index_{beat} \leftarrow index_{beat} + 1$
    end while
    if $rests > 0$ then
        $notesAndRests.add(rests)$
    end if
    while $beatlist[index_{beat}] < note_{current}[offset]$ do
        $index_{beat} \leftarrow index_{beat} + 1$
        $ticks \leftarrow ticks + 1$
    end while
    if $current_{note}! = lastnote$ then
        $nextOnset \leftarrow notelist[index_{note} + 1][onset]$
    else
        $nextOnset \leftarrow -1$
    end if
    if $beatlist[index_{beat}] = note_{current}[offset]$ and $nextOnset! = note_{current}[offset]$ then
        $index_{beat} \leftarrow index_{beat} + 1$
    end if
    $note_{current}[ticks] \leftarrow ticks$
    $notesAndRests.add(note_{current})$
    $index_{note} \leftarrow index_{note} + 1$
end while
```

Estimating Time Signature and Tempo

The problem of estimating the 'right' Time Signature can be seen as the problem of assigning the right location of the downbeat[27], however as can be seen from the Figure 5.7, for each combination of Upper Level Division and Tactus Level Division, quite a few possible Key Signatures exist and all of them are technically right, however for eg. in 2/4 the stress would be on 1st Beat, then 3rd Beat, whereas in 4/4 the stress of 1st beat, then 5th beat.

[27] The downbeat is the first beat of the bar

The Tactus level (Level 2) division also dictates an important property- when the division is 2 (Tuple), there are 2 Level 1 beats, which means 2 $1/8^{th}$ notes which implies the tactus beat is a $1/4^{th}$ (Quarter) Note, whereas when the division is 3 (Triple), there are 3 level 1 beats, which means 3 $1/8^{th}$ notes which implies the tactus beat is a Dotted-Quarter ($1/4^{th} + 1/8^{th}$) note. This is useful in establishing the lower fractional in the Time Signature, for division of 2 it's 1/4 ie. *Simple Meter*, and for division of 3 its 1/8 ie. *Compound Meter*.

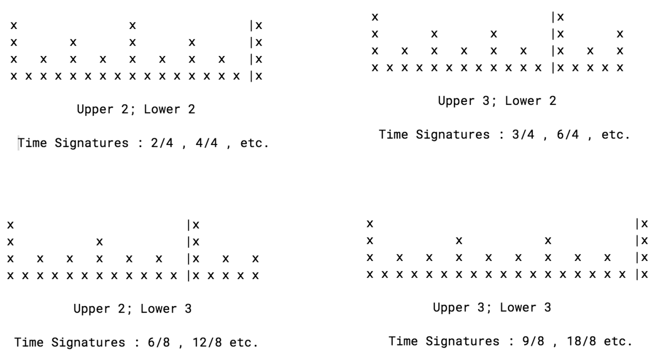

Upper 2; Lower 2
Time Signatures : 2/4 , 4/4 , etc.
Upper 3; Lower 2
Time Signatures : 3/4 , 6/4 , etc.
Upper 2; Lower 3
Time Signatures : 6/8 , 12/8 etc.
Upper 3; Lower 3
Time Signatures : 9/8 , 18/8 etc.

Figure 5.7: Division of Tactus levels, with Downbeat Locations and Possible Time Signatures

Decision was made to use the default for every combination as highlighted in the table 5.1. It is an obvious fault in the Model, and figuring out a way to estimate just the right downbeat location would be part of the future work.

Table 5.1: Default Key Signatures for Level Division Combinations

Upper Division	Tactus Division	Time Signature
2	2	4/4
2	3	6/8
3	2	3/4
3	3	9/8

The tempo calculation was straightforward:

$$Tempo = \frac{interval_{tactus} * 60}{1000 * TactusNoteLength}$$

where the $interval_{tactus}$ is the interval between two tactus beats in *milliseconds* and the *TactusNoteLength* is 1 for Quarter note tactus, and 1.5 for Dotted-Quarter note tactus.

Table 5.2: Level 0 paddings added before the first note to incorporate Upper level phase.

Time Signature	Phase 1	Phase 2	Phase 3
4/4	0	12	0
6/8	0	6	0
3/4	0	8	4
9/8	0	12	6

The last thing to incorporate is the Upper level phase (as mentioned in the generative process), calculations were done manually to incorporate this phase as a quantised number of level 0 beats that should be at added before the first note of the melody as invisible rests, this helps in typesetting the score in the next stage with the right location of the bar. The duration of this phase is directly linked to the time signature, and the table 5.2 shows the values that are picked (Phase 3 is 0 for cases when the upper level division is duple). This invisible rest was added to the beginning of the output *notesAndRest* list.

5.5 Evaluation

The testing of the 'rightness' of metrical grid is inherently still an unsolved problem, one could just check the time signature, however that doesn't convey information about the exact placement of each note with the beat. The testing method used here is the 'note-address' system described by Temperley in his paper[28].

[28] David Temperley. An evaluation system for metrical models. *Computer Music Journal*, 28(3):28–44, 2004b

In this method, each note is assigned an address representing its position in the metrical grid, the address is a (N+1) digit number where N is the maximum number of levels in the metrical grid, in our case 3. The left most number represents the number of L3 beats that the note is on, the second left number represents the L2 beat and so on - for example, the first 1/8th note in the third measure of a piece in 3/4 time signature would be represented as 3000, the second 1/8th note would be 3010, the third quarter note in the 6th measure played would be 6200.

We use this note address system to evaluate the model on the test set of the Essen Corpus dataset, converting the test file's kern score into the note address format and the model's output as well, the accuracy on each metrical level is calculated and averaged over as the overall accuracy for that piece. The results are shown in the table **??**.

Table 5.3: Accuracy of the meter model on Essen Corpus test set.

Level 3	Level 2	Level 1	Level 0	Overall Score
78.6%	89.5%	94.1%	98.3%	90.1%

The tempo was directly evaluated as the tempos for each test file were annotated, the tempo is evaluated on how many numbers its off by, with a penalty of 0.10 per step (ie. the exact tempo gets the score of 1, if it's less by 1 or more by 1 from the original, the score given is 0.9, it's less or more by 3 steps, the score given is 0.7). The accuracy was *95%*

While the model was considerably successful on the test sets, it does have room for improvement - perhaps assigning a higher metrical level, to assign bar location more appropriately, factoring in parallelism (repeated patterns, the tendency to align meter with the patterns so that the strong beat always occurs at the same part of a pattern[29]). Also it should be noted that the meter model has been tuned to a corpus of essentially folk songs, so it does have a tendency to understand the duple and triple meters that are more predominant in Western Music[30], perhaps future improvements would look at training the model on more diverse styles of music, and relaxing some of the assumptions that the model takes as principles.

[29] David Temperley and Christopher Bartlette. Parallelism as a factor in metrical analysis. *Music Perception: An Interdisciplinary Journal*, 20(2):117–149, 2002

[30] Although the melodies from Asian origin do add a little more robustness

6 THE SCORE

In this chapter we walk through the basics of staff notation and show how all the models from the previous chapters are connected to create an automatic music transcription system.

6.1 *Basics of Staff Notation*

Staff Notation is the most common way of writing music, it incorporates both the metrical and harmonic information for a given music piece, and also incorporates the playing style.

The Staff

The staff is the primary drawing board, it consists of the 5 lines, where notes are written either on the line and between the lines. Sometimes staffs lines are extended by drawing lines above or below, to essentially extend the range of notes that can be played.

The Clef

Which line and space of the staff correlates to which pitch is given by the *Clef*. The two most common ones used in music are the Treble clef and The Base clef, the figures 6.1 6.2 shows where pitches are on the staff for the given clef.

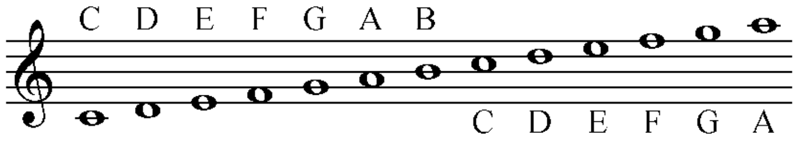


Figure 6.1: Treble Clef with Notes

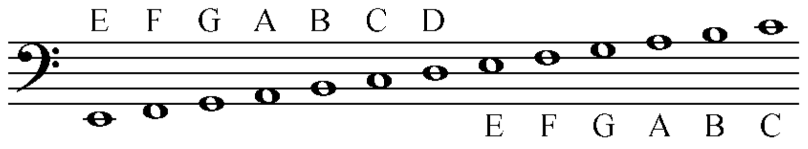


Figure 6.2: Bass Clef with Notes

The Note

The note is divided into two parts, the *head* and the *stem*, the position of the head of the note dictates what pitch to play, the stem (or lack of) dictates the duration of the note (The symbols are given in the figure 5.1). Notes above the middle staff line have a downward stem, whereas the notes below an upward stem - this plays no importance whatsoever and is merely a presentational thing.

Sometimes notes are joined by bars, its just a way of writing simultaneous 1/8th notes (1 bar), 1/16th notes (2 bars) and so on.

A dot after the note head, adds another half of that noteś duration to it. So, a quarter note with a dot would equal a quarter note and an eight note; a quarter note with a dot equals a quarter plus an eighth note. A tie may also be used to extend a note. Two notes tied together extends the duration of the note to the sum of the two notes, they are commonly used to signify held notes that cross bars.

A note can have have an accidental ($\sharp, \flat, \natural$), sometimes double as well, however a single accidental is usually preferred, as a double can be written as another note (C$\sharp\sharp$ is D).

Key Signature

The Key Signature is represented in the form of sharps $\sharp$ or flats $\flat$ on the staff at the beginning with the Clef, the lines or the space they're put on represents which pitches are altered with the accidental.

6.2 The Transcription Model

Up to this point the individual models for melody extraction, key detection and meter have been trained and stored. We proceed to explain how they all are tied in together (Figure 6.3).

- Audio file is given to the system and loaded using librosa library [1]

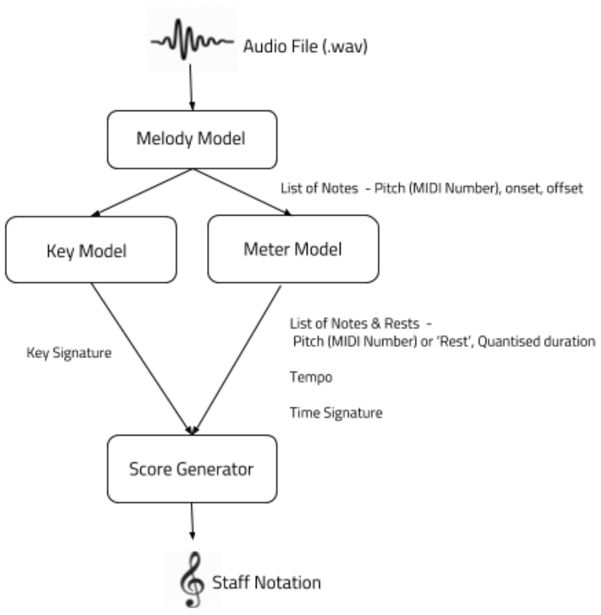
- Audio file is fed into the trained pYIN model (using VAMP[2]), which gives the output of the extracted *note, onsets and offset times.*

[1] https://librosa.github.io/librosa/

[2] https://www.vamp-plugins.org/vampy.html

- These note (converted to MIDI), onset and offset times are fed into the Meter Model, which returns the *Time Signature, Tempo, Notes and Rests with their quantised durations*
- The notes are fed into the Key Model which returns the *Key*
- All the above are fed into the Score Generator which typesets and displays the score.

Figure 6.3: The Transcription Model


Audio File (.wav)
Melody Model
List of Notes - Pitch (MIDI Number), onset, offset
Key Model
Meter Model
Key Signature
List of Notes & Rests - Pitch (MIDI Number) or 'Rest', Quantised duration
Tempo
Time Signature
Score Generator
Staff Notation

6.2.1 *Generating the Score*

We had mentioned music21[3] before as a robust music analysis toolkit based on python, we utilise this for converting our outputs from the previous model into a score. The primary element is the *music21.stream.Stream* class which allows us to create the wrapper structure for the score. We begin by initialising a stream object.

[3] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010

6.2.2 *Key to Key Signature*

Given the output of the KeyModel of form "Root - Major/Minor" it was needed to convert this into a list of accidentals, that is which of the 12 natural notes are modified by an accidental. The *music21.key.Key* class was used to get the number of sharps/flats in the Key, and selection of the correct Key Signature was made from that (*music21.meter.KeySignature* object) . Also, certain Keys were modified for better typesetting ie. any Key with more than 6 sharps was represented in flats (For example, C♯ major Key has 7 ♯s, and has the notes C♯, D♯, E♯, F♯, G♯, A♯, B♯, writing which is not technically right and at the same time laborious to read, however when transposed to D♭, the key has 5♭s and the notes become D♭, E♭, F, G♭, A♭, B♭, C, which are much more readable). This key signature object (*music21.key.KeySignature* object) is then appended to the stream object.

6.2.3 *Modifying the notes to fit the Key Signature*

Before we can add the notes from the meter model to the stream class object, one final step remained, which is to adjust the notes to fit the key signature, as was mentioned in the fundamentals, If the accidental is mentioned in the Key Signature, the note doesn't have to have the accidental next to it in the score. This was accomplished using the *music*21*.key.Key* class again, which had the function to return all the notes of the given Key's corresponding scale (a list of *music21.note.Note* objects). The process then proceeded as follows:

- All input MIDI notes were converted to *music21.note.Note* objects, and the rests to *music21.note.Rest* objects.
- Iterating over each of these note objects, if the corresponding note existed in the scale list, the note from the scale replaced the given note, also if it had an accidental, that was removed. (So, For example, if the given Key Signature is D♭, if a note A♯ was found in our input, its converted first to B♭ and then the accidental is dropped - making the final note *B*).
- Finally for all the notes, the appropriate octave and duration was set, and for all the rests the duration was set. These were then added to the stream object.

Time Signature and Measures

music21 has a powerful under the hood feature of automatically generating the bar locations given the time signature. The time signature output from the meter model is used to initialise the *music21.meter.TimeSignature* class object. Similarly the tempo from the meter model is used to initalise the *music21.tempo.MetronomeMark* object. All these are appended to the stream object.

Picking the Clef

The stream object is smart in the sense it picks the best clef given the range of octaves of the notes entered in it, so no additional computations were required for that.

LilyPond and the final output

music21 allows a range of output formats for a given stream, which include musicxml[4], MIDI, etc however the requirement was for the score to be either in PDF or an Image format. music21 did offer a way to convert the stream object to LilyPond[5,6] format, which can generate images, PDFs (and also MIDI, in case one needs to hear the final result to compare against the input!). So the stream object was converted to LilyPond's native format, and then using LilyPond, to a PDF / Image (As required, default is PDF).

[4] The musicxml format. `https://www.musicxml.com/`

[5] The lilypond notation. `http://lilypond.org/`

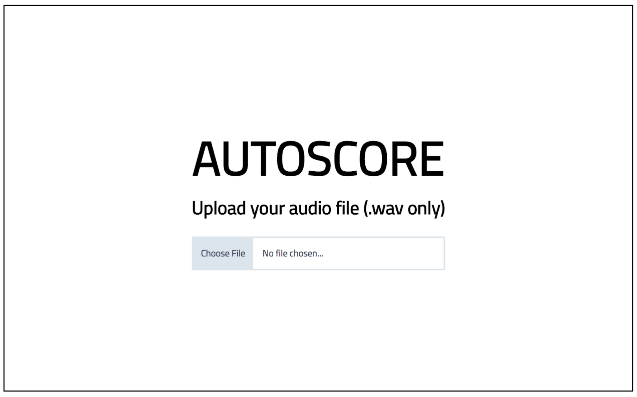
[6] LilyPond is a markdown style music engraving software, widely accepted in the community to produce accurate and quite aesthetically pleasing outputs

6.2.4 *Web Application*

A simple web application was then created for the model to be run, Flask[7] a microframework to create web application in Python, was used. A jinja2[8] template was created with an upload form (limiting the upload format to only wav), after a song is uploaded, its processed by the model, and the generated image is sent to the server, which loads it on another jinja2 template and displays to the user.

[7] http://flask.pocoo.org/

[8] http://jinja.pocoo.org/ A simple templating engine written in python to output formats like HTML



AUTOSCORE

Upload your audio file (.wav only)

Figure 6.4: The Web App upload screen

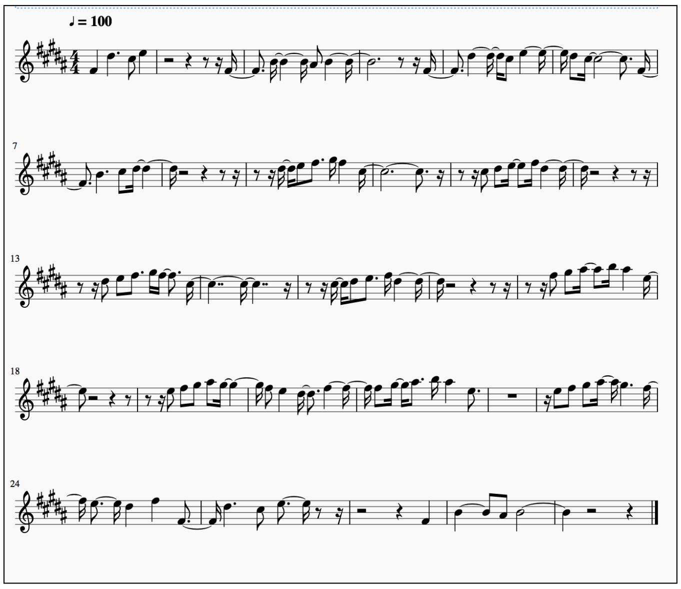


Figure 6.5: The Web App result screen - Abba's I have a dream

7 FINALE

7.1 Comparisson with AnthemScore

AnthemScore[1] is one of the most popular audio transcription packages available, It treats the problem as that of an image recognition problem and runs on a convolutional neural network to generate the transcription [2]. It supports polyphonic audio (however instruments are not separated).

[1] `https://www.lunaverus.com/`

[2] Read here for how the algorithm works `https://www.lunaverus.com/cnn`

For the testing, monophonic live audio recordings are hard to come by, so I recorded [3] 10 different[4] melodies on a piano and guitar, and ran all of them through our system and AnthemScore. We utilise the metrics defined in Equation3.6 for pitch estimations, section 4.5 for key evaluation and section 5.5 for metrical estimations. The table 7.1 shows the accuracies for each part of the pipeline.

[3] trained musician myself

[4] Varying Genres - classical, pop, nursery rhyme, folk

Table 7.1: Comparison of AnthemScore and our system

Part	AnthemScore	System
Pitch Detection	78.32%	84.74%
Key Detection	100%	100%
Meter Detection (Note address overall)	93.21%	83.50%
Tempo	97.00 %	89.00%

As is evident from the results, our system performed much better than AnthemScore in Pitch detection - AnthemScore had a tendency to pick a note along with the same note an octave or 2 octaves below at the same time, even though the input was monophonic. In terms of key finding, both models were the same. The metrical grid was where our system performed poorly, at times not associating the right rests, other times making a mistake with the metrical grid itself - this is could be partially attributed to the error being compounded from the pitch extraction phase, where certain notes were not picked up, also this is a pointer to improvement needed in the model to differentiate between time signatures (eg. between 2/4 and 4/4). Patterns like triplets were clearly missed by our system, this should be factored in for future improvements to the meter model. The file by file comparison of generated scores are given in Appendix A.

7.2 *Discussion & Future Work*

The report has demonstrated that a system for music transcription built purely on a set of individual probabilistic models, optimised using a training corpus of folk melodies, utilising fundamental principles and experimental patterns in Western music, is *quite successful* and has a respectable performance. However there are considerable improvements that need to be taken to make it competitive with the state of the art systems out there. The system also reinforces that approaching the problem from a probabilistic view does have its merits in modelling higher level principles governing music, often presenting useful insights that were previously unknown, and is a viable route to be taken for future work.

Below I have presented a non-exhaustive list of future improvements and direction the work can take:

- Extending to Polyphonic Music - An obvious extension, this would be multi-faceted in terms of improving the pitch model to detect and recognise multiple notes and instruments, the key model to factor in tonal harmonies and supporting chords, the meter model to take into account stream separations. This would also open the doorway of using more corpora that we couldn't use in our system.
- Preprocessing Audio - Audio files could be preprocessed to remove background noise, reverb and echo.
- Training on live recordings - As suggested in chapters *The Pitch* and *The Beat*, since the current pitch model optimises the parameters using synthesised audio files in which the note durations are always precise, whereas in an actual performance, the musician can never be so precise, the system can be made more robust by training the models on live recordings.
- Training on different genres of music - The current system's parameters and principles have been based on folk music, which has lead to the model performing better on Western styles of music (predominantly classical music, church hymns, nursery rhymes etc), however the system does falter a bit for genres like jazz and rock. This could be improved by incorporating the nuisances of different genres into the individual models themselves, and training them with more a diverse corpus.
- Training on different instruments - Each instrument has different tonal qualities, a human voice sounds different from a violin, a violin sounds different from a flute, also different makes of violins sound different! The system was trained on synthesised sounds of

just piano, electric guitar and flute which does make the system work for other unseen instruments and voices as well, however still leaves a big room for improvement.

- Unifying the models - The different models of the system (pitch extraction, key finding and meter extraction) are trained and operated individually, which in some cases does lead to errors getting missed and compounded through the pipeline. Unifying these models so that they communicate more with each other both in training and evaluation phase would be a viable and rewarding route to pursue.

BIBLIOGRAPHY

The humdrum **kern representaton. `https://musiccog.ohio-state.edu/Humdrum/representations/kern.html`.

The lilypond notation. `http://lilypond.org/`.

What is midi? `http://www.instructables.com/id/What-is-MIDI/`.

The musicxml format. `https://www.musicxml.com/`.

Onur Babacan, Thomas Drugman, Nicolas d'Alessandro, Nathalie Henrich, and Thierry Dutoit. A comparative study of pitch extraction algorithms on a large variety of singing sounds. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7815–7819. IEEE, 2013.

Sebastian Böck and Markus Schedl. Polyphonic piano note transcription with recurrent neural networks. In *Acoustics, speech and signal processing (ICASSP), 2012 ieee international conference on*, pages 121–124. IEEE, 2012.

Helen Brown, David Butler, and Mari Riess Jones. Musical and temporal influences on key discovery. *Music Perception: An Interdisciplinary Journal*, 11(4):371–407, 1994.

Ali Taylan Cemgil. *Bayesian music transcription*. [Sl: sn], 2004.

Ali Taylan Cemgil, Bert Kappen, and Peter Desain. Rhythm quantization for transcription. *Comput. Music J.*, 24(2):60–76, July 2000a. ISSN 0148-9267. DOI: 10.1162/014892600559218. URL `http://dx.doi.org/10.1162/014892600559218`.

Ali Taylan Cemgil, Bert Kappen, Peter Desain, and Henkjan Honing. On tempo tracking: Tempogram representation and kalman filtering. *Journal of New Music Research*, 29(4):259–273, 2000b.

Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010.

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. URL `http://recherche.ircam.fr/equipes/pcm/cheveign/pss/2002_JASA_YIN.pdf`.

Peter Desain and Henkjan Honing. Computational models of beat induction: The rule-based approach. *Journal of new music research*, 28(1):29–42, 1999.

Diana Deutsch. Grouping mechanisms in music. In *The Psychology of Music (Third Edition)*, pages 183–248. Elsevier, 2013.

David Gerhard. *Pitch extraction and fundamental frequency: History and current techniques*. Department of Computer Science, University of Regina Regina, 2003.

Anssi Klapuri and Manuel Davy. *Signal processing methods for music transcription*. Springer Science & Business Media, 2007.

Carol L Krumhansl. Cognitive foundations of musical pitch. 1990.

Carol L Krumhansl and Edward J Kessler. Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological review*, 89(4):334, 1982.

Edward W Large and John F Kolen. Resonance and the perception of musical meter. *Connection science*, 6(2-3):177–208, 1994.

Christopher S Lee. The perception of metrical structure: Experimental evidence and a model. *Representing musical structure*, pages 59–127, 1991.

H Christopher Longuet-Higgins and Mark J Steedman. On interpreting bach. *Machine intelligence*, 6:221–241, 1971.

Matthias Mauch and Simon Dixon. pyin: A fundamental frequency estimator using probabilistic threshold distributions. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014)*, 2014. in press.

George A Miller and George A Heise. The trill threshold. *The Journal of the Acoustical Society of America*, 22(5):637–638, 1950.

Dirk-Jan Povel and Peter Essens. Perception of temporal patterns. *Music Perception: An Interdisciplinary Journal*, 2(4):411–440, 1985.

Christopher Raphael. Automatic transcription of piano music. In *ISMIR*, 2002.

Helmut Schaffrath and David Huron. The essen folksong collection in the humdrum kern format. *Menlo Park, CA: Center for Computer Assisted Research in the Humanities*, 1995.

Ilya Shmulevich and Olli Yli-Harja. Localized key finding: Algorithms and applications. *Music Perception: An Interdisciplinary Journal*, 17(4):531–544, 2000.

Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 24(5):927–939, 2016.

David Temperley. *The cognition of basic musical structures*. MIT press, 2004a.

David Temperley. An evaluation system for metrical models. *Computer Music Journal*, 28(3):28–44, 2004b.

David Temperley. *Music and probability*. Mit Press, 2007.

David Temperley. A probabilistic model of melody perception. *Cognitive Science*, 32(2):418–444, 2008.

David Temperley. A unified probabilistic model for polyphonic music analysis. 38, 03 2009.

David Temperley and Christopher Bartlette. Parallelism as a factor in metrical analysis. *Music Perception: An Interdisciplinary Journal*, 20(2):117–149, 2002.

David Temperley and Daniel Sleator. Modeling meter and harmony: A preference-rule approach. *Computer Music Journal*, 23(1):10–27, 1999. DOI: 10.1162/014892699559616. URL `https://doi.org/10.1162/014892699559616`.

Guido Van Rossum and Fred L Drake. *The python language reference manual*. Network Theory Ltd., 2011.

Paul Von Hippel and David Huron. Why do skips precede reversals? the effect of tessitura on melodic structure. *Music Perception: An Interdisciplinary Journal*, 18(1):59–85, 2000.

APPENDIX A - GENERATED SCORES OF LIVE RECORDINGS

Each page of the following section contains the comparative outputs of our AMT system and AnthemScore on the recorded test files. The AnthemScore outputs have both Treble and Bass Clef due to its default configuration.

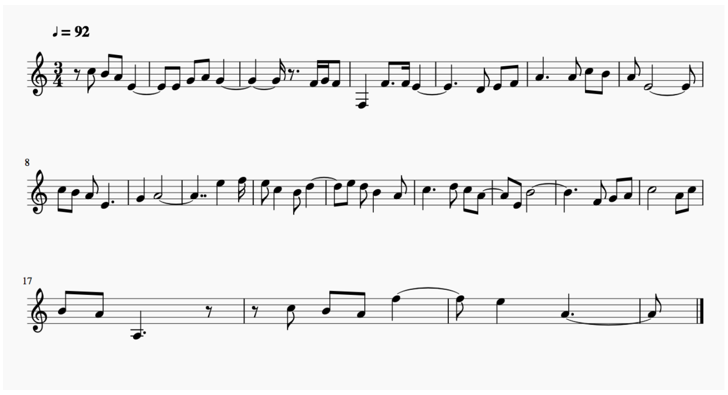


Figure 7.1: System output for piece in A minor

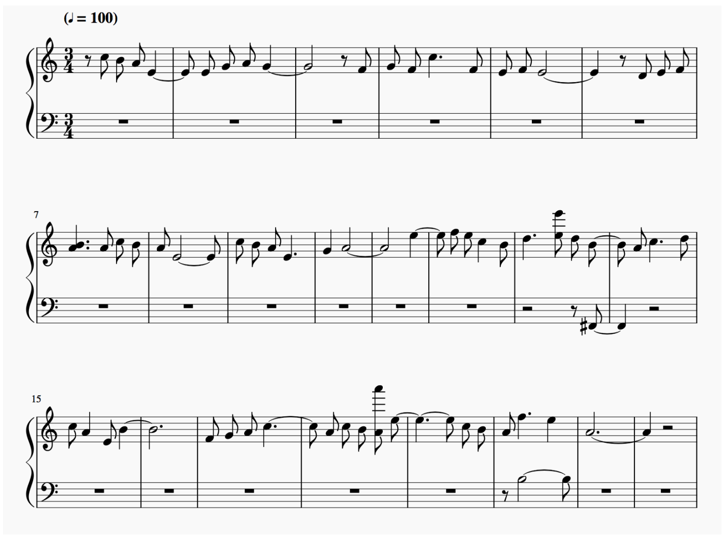


Figure 7.2: AnthemScore output for piece in A minor

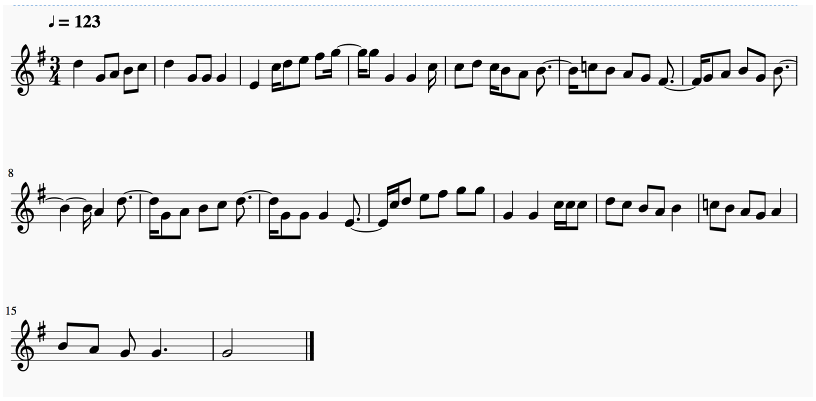


Figure 7.3: System output for Bach's Minuet

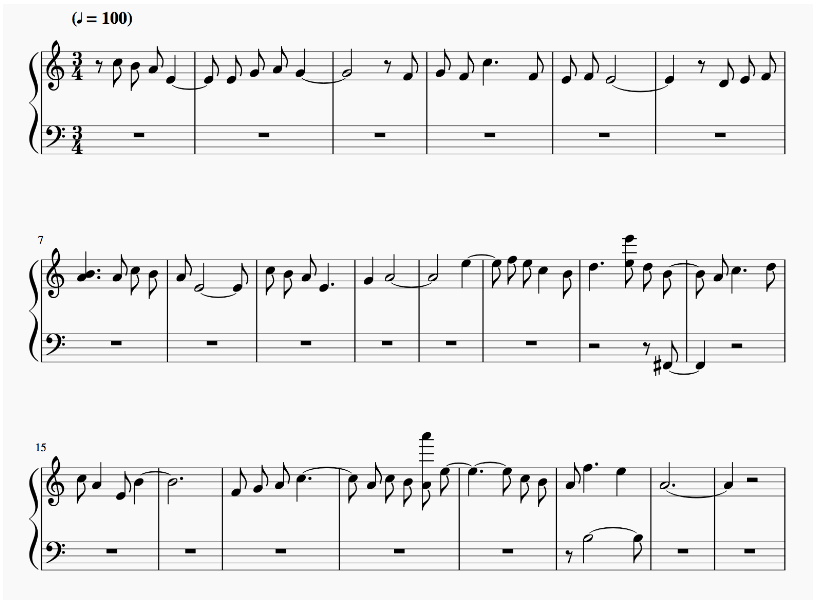


Figure 7.4: AnthemScore output for Bach's Minuet

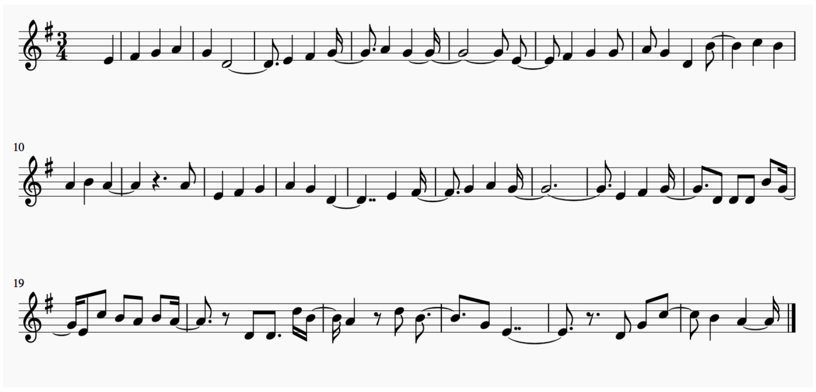


Figure 7.5: System output for Can you feel the love

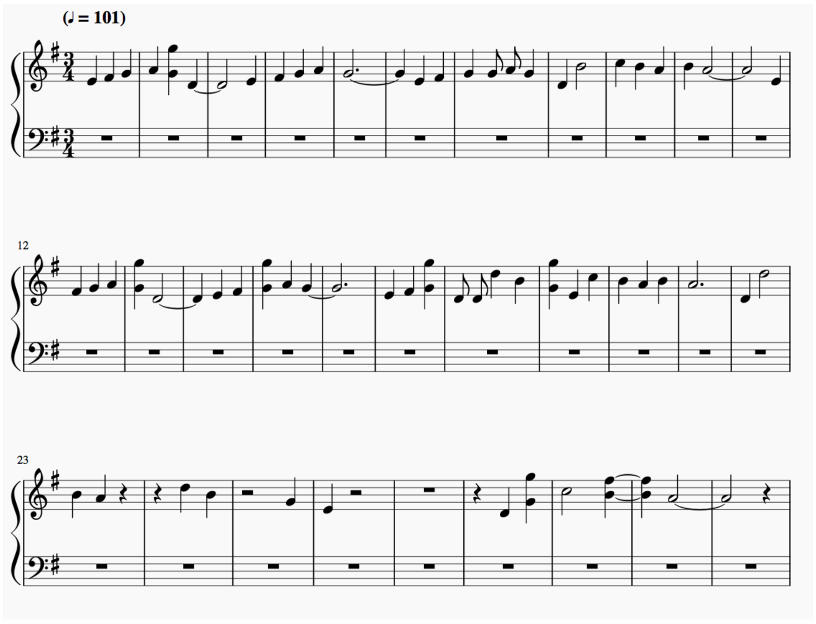


Figure 7.6: AnthemScore output for Can you feel the love

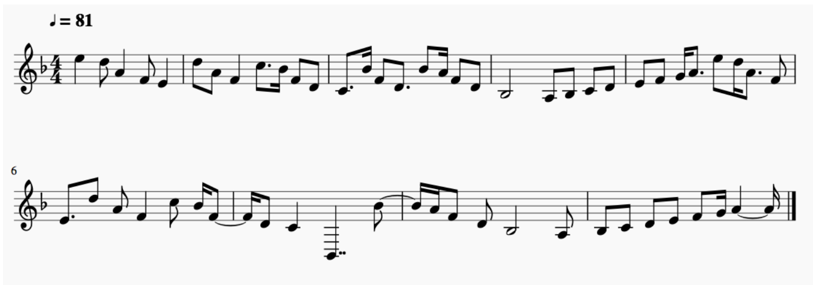


Figure 7.7: System output for Careless Whisper

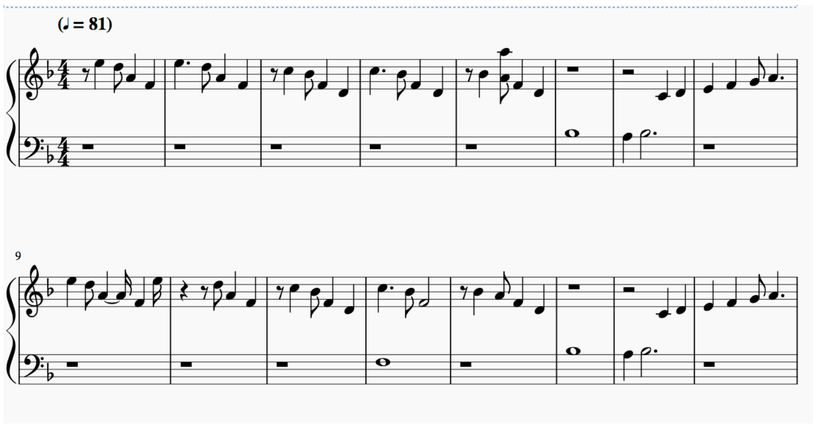


Figure 7.8: AnthemScore output for Careless Whisper

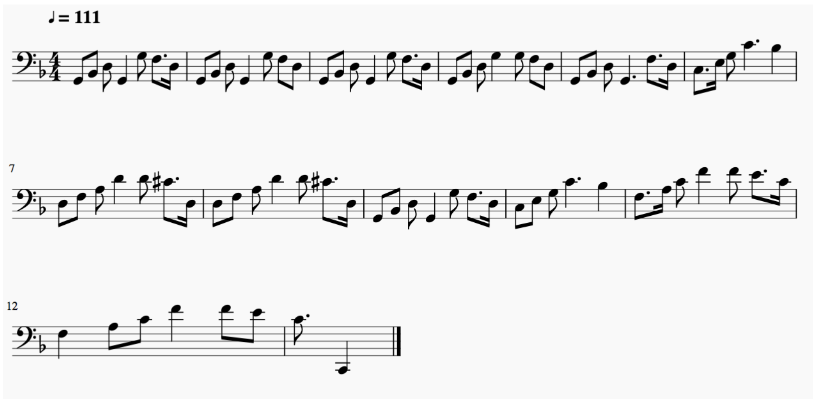


Figure 7.9: System output for City of Stars (La la land)

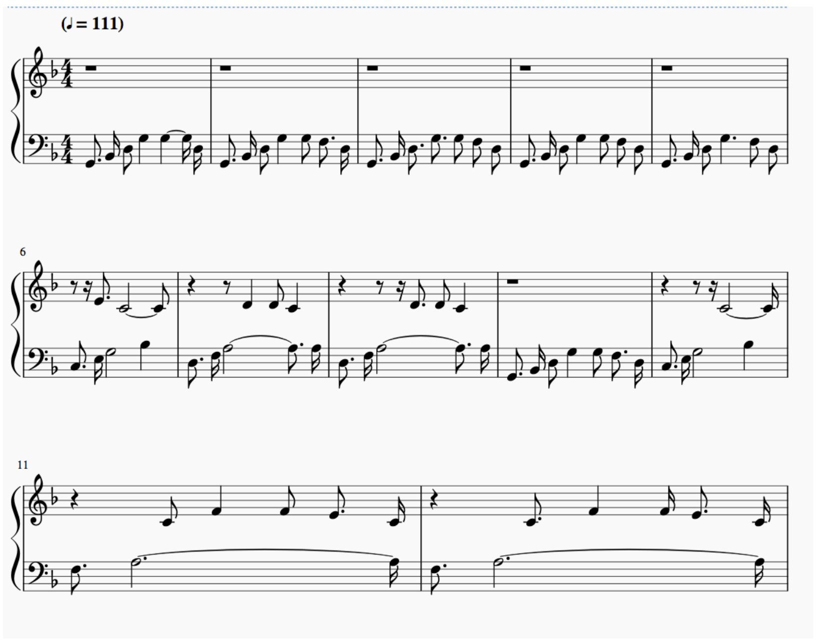


Figure 7.10: AnthemScore output for City of Stars (La la land)

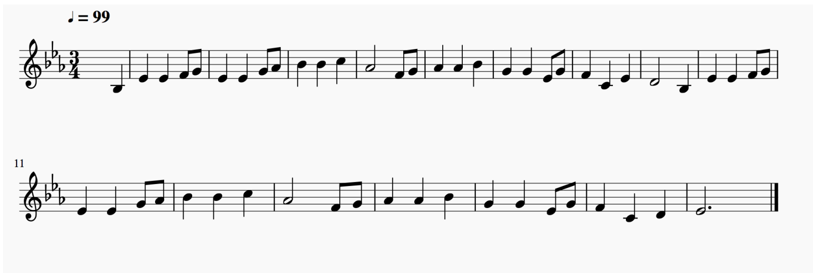


Figure 7.11: System output for piece in E flat

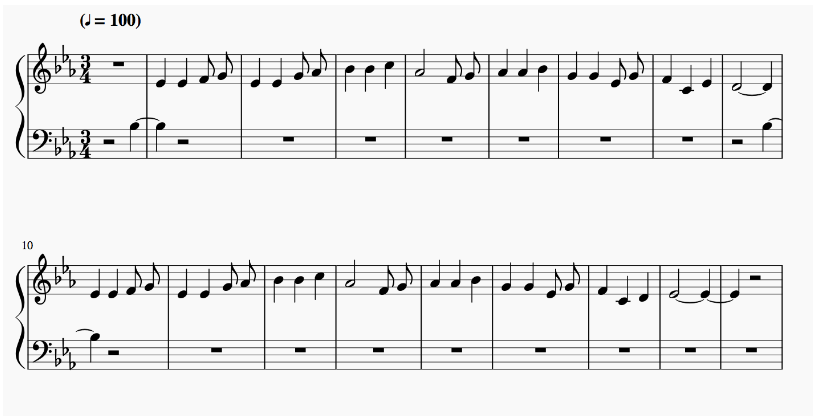


Figure 7.12: AnthemScore output for piece in E flat

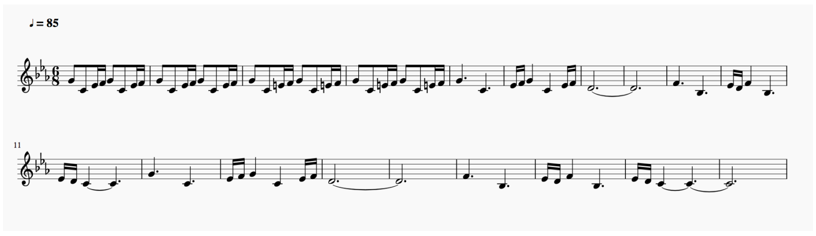


Figure 7.13: System output for Game of Thrones main theme

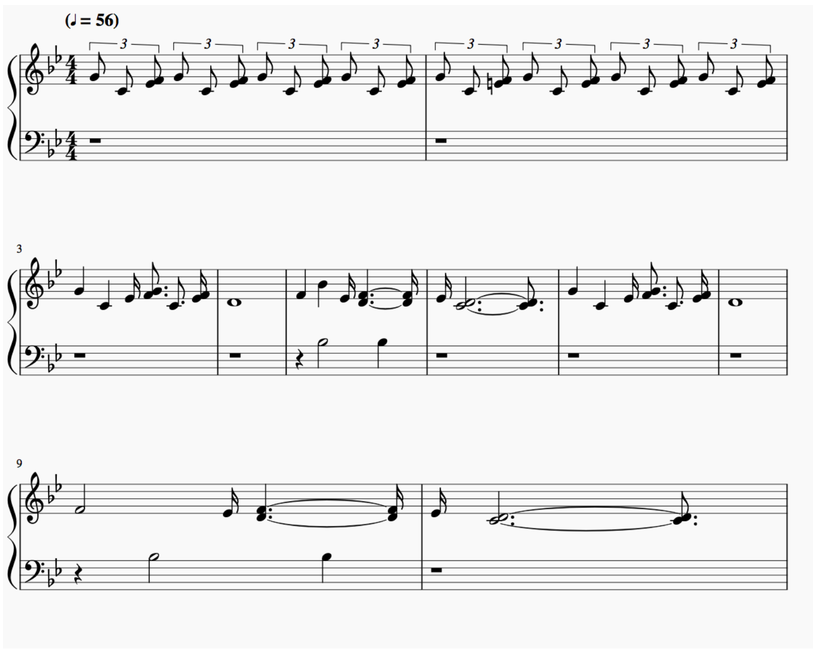


Figure 7.14: AnthemScore output for Game of Thrones main theme

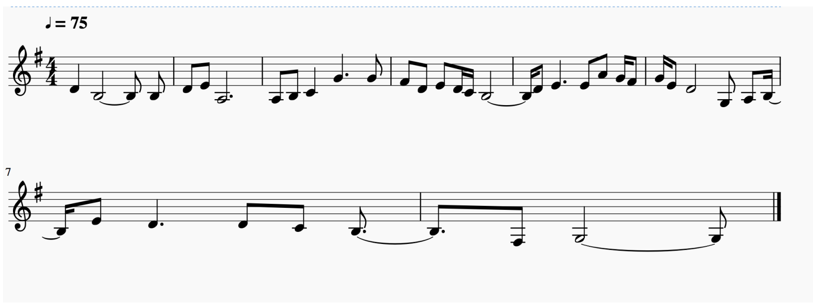


Figure 7.15: System output for Hey Jude by Beatles

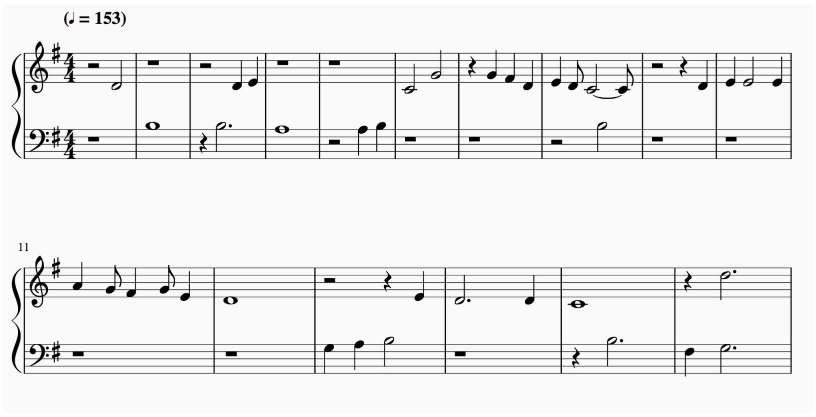


Figure 7.16: AnthemScore output for Hey Jude by Beatles

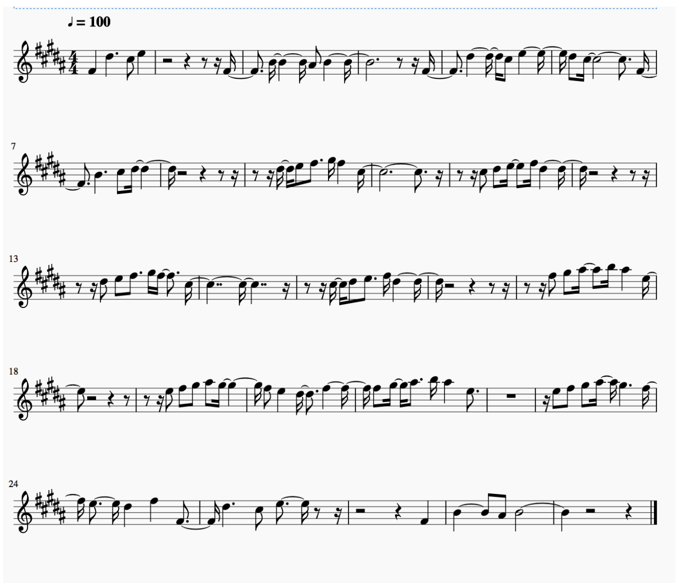


Figure 7.17: System output for I have a dream by Abba

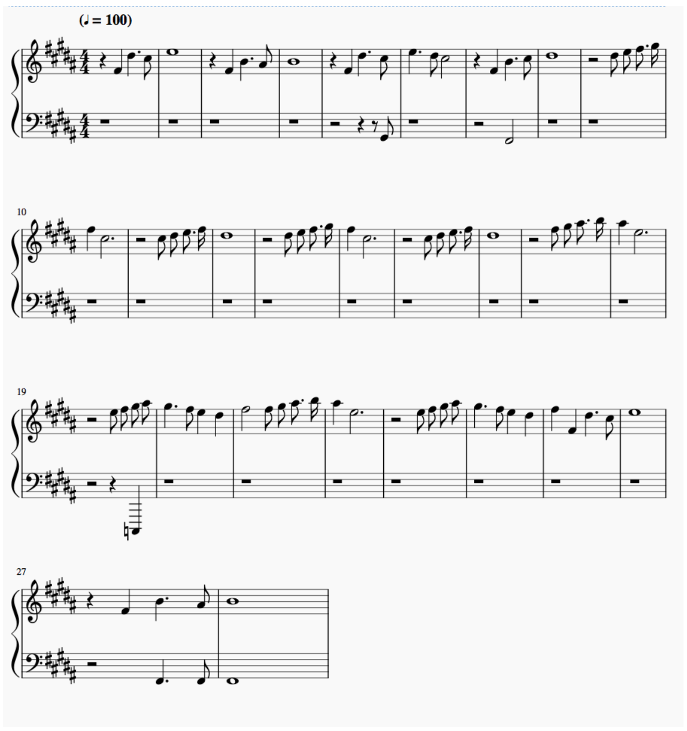


Figure 7.18: AnthemScore output for I have a dream by Abba

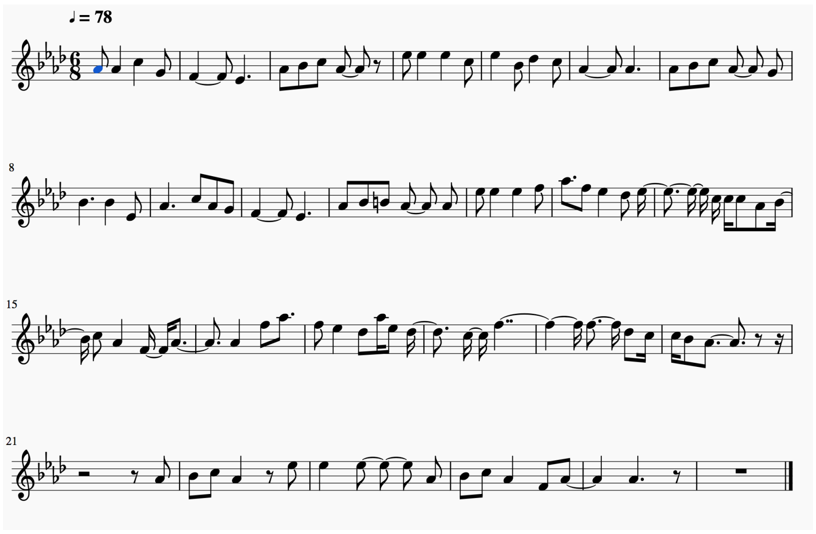


Figure 7.19: System output for O mio babbino

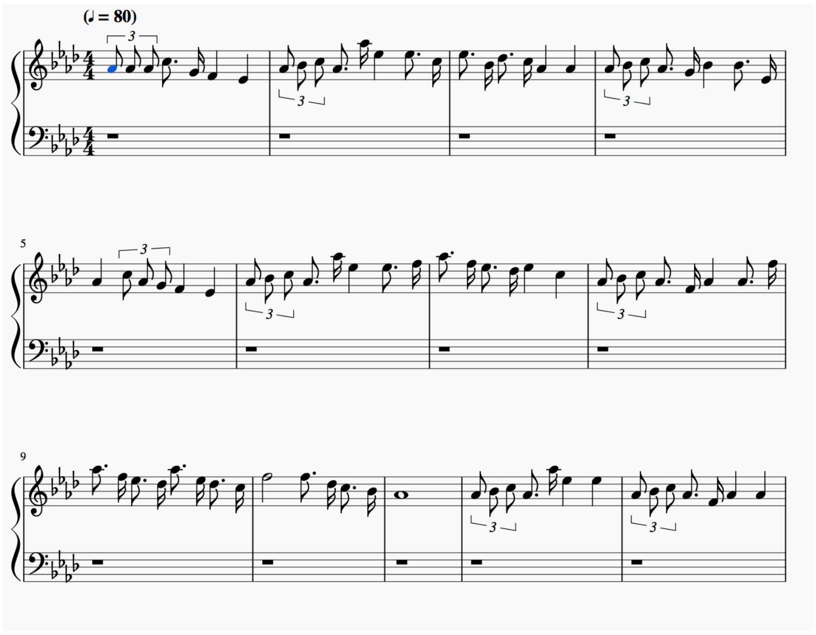


Figure 7.20: AnthemScore output for O mio babbino