Monophonic MIDI Transcriber

Grant Kilmer and Xuefan Hu

University of Rochester

Abstract

The Monophonic MIDI Transcriber is a program that allows a user to import a monophonic, mono sound file, and receive a MIDI representation of that file as the output. This functionality would be useful to experienced and beginning musicians alike, as this could be applied to well known songs or even just music heard from a street performer on any day. The MIDI representation can be imported into any digital audio workstation which would allow the user to see the score of the melody that was played, and thus be able to sample it or even learn it. This algorithm would seamlessly improve or enhance the creative musical environment that is unique to each user.

Index Terms— Tempo, Pitch, Beat, MIDI

1. Introduction

Applying audio analysis of an audio file and extracting pitch and beat information in the form of MIDI file can be widely used in many different ways. Many of the music we listen today does not come with score or midi information. Wanting to convert something you hear into adjustable and visible midi file can be a time consuming and frustrating process. This paper intends to develop a method through pitch and beat detection using MATLAB to automatically transcribe a monophonic piece of music.

The paper will explain pitch and beat detections separately. Then in the fourth section, it will explain how to use the gathered information through the previous detected methods to output MIDI files in MATLAB.

2. Pitch Detection - YIIN Method

Single pitch detection of frames of an input signal is not a new feature. It has been done before and there are many different methods. For implementation into this algorithm, the YIN method has been used [1]. This method will be described in full with accompanying figures for this algorithm.

2. 1. Breaking input into frames

Before the inputted audio file can be processed and the pitch can be detected, it must be broken into frames so that each frame can be processed individually. A 50ms frame length has been used for this algorithm, arbitrarily chosen from trial and error of the pitch detector, and the frame hop size is half of the frame length. With this information, the input data can be split up based on these parameters, and the data at each frame is multiplied by a hamming window to avoid errors and discrepancies where each frame is cut.

2. 2. YIN Implementation

The YIN method initially resembles an auto-correlation function where each frame of a signal is manipulated based on shifted versions of the data contained within that frame. However, the YIN method utilizes a “difference function” on each frame to manipulate the information in each frame as shown:

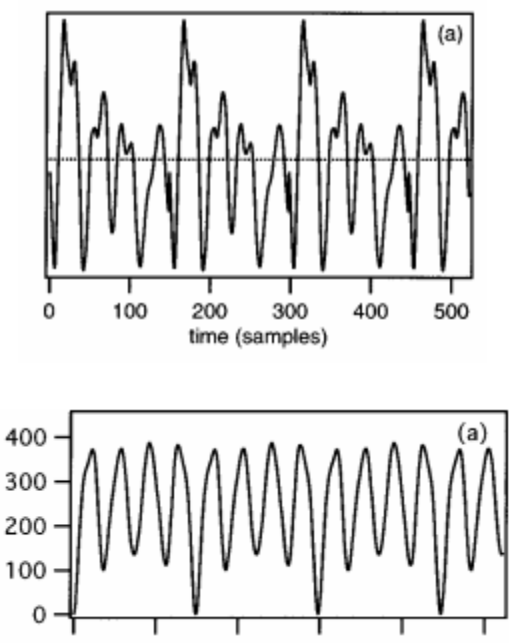
$$d_t(\tau) = \sum_{j=1}^{W} (x_j - x_{j+\tau})^2, \quad [1]$$

Here, each frame is subtracted with a shifted version of itself which is then squared and summed together through all values of the shift which gives you the difference function value for that frame. After going through this, the next step is to calculate the "mean difference function" which is done as follows:

$$d'_t(\tau) = \begin{cases} 1, & \text{if } \tau = 0, \\ d_t(\tau) \Big/ \left[(1/\tau) \sum_{j=1}^{\tau} d_t(j) \right] & \text{otherwise.} \end{cases}$$

[1]

Doing this avoids the problem of when tau becomes so small that the difference function is essentially subtracting the values within a frame by themselves. A sample of a signal being passed through the functionality at this point is shown in figure 1.



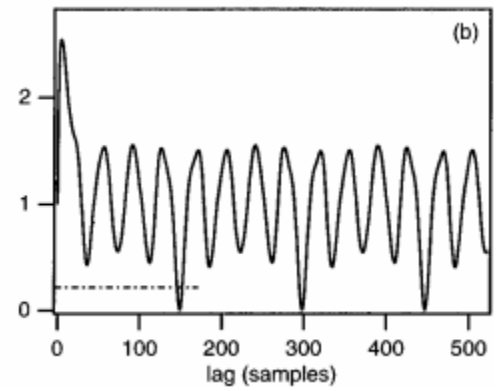


Figure 1. A frame of a signal being passed through the difference function and then the cumulative mean function [1].

At this point the pitch detection algorithm calculates the probability that this given frame is pitched by subtracting the minimum value of the cumulative mean difference from 1. This will mean that strongly periodic or pitched signals will have pitch probabilities very close to 1 since the periodicity results in strong dips in the difference functions. This also allows a threshold to be set on this probability which will eliminate inharmonic and noisy frames from being calculated for pitch. The pitch probability function displayed for a whole signal is shown in figure 2.

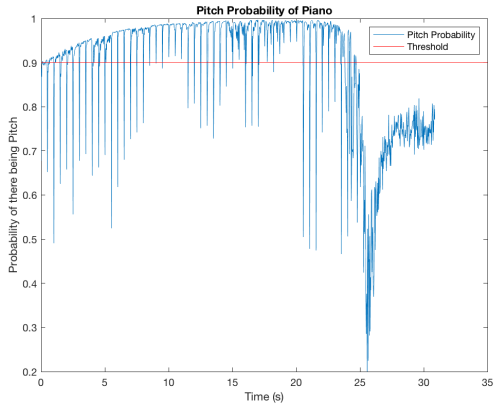


Figure 2. Pitch Probability with threshold of an inputted piano sound.

Now, if a frame's pitch probability is above the threshold, the fundamental frequency can be calculated from the first large dip in the

cumulative mean difference function. This value corresponds to a frame's period and the frequency is found by dividing the sampling frequency by this number. If the frame did not pass the threshold for its pitch probability, then its frequency value is set to "not a number" so that it is not graphed and ignored. A sample output of the pitch detection functionality is shown in figure 3.

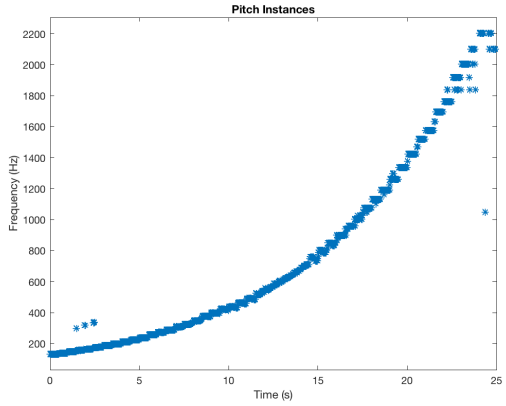


Figure 3. Sample output of pitch detection where each point corresponds to the found fundamental frequency of each frame of an inputted piano signal that steps through keys from low to high.

3. Beat Detection

After tracking the pitches, another important factor in MIDI, actually all music, is the instance of where the note occurs. This section intends to achieve two goals: Finding the instances of where the musical notes occur and then using a Tempogram to estimate the tempo of the musical piece. This information will be passed into the MIDI generating toolkit along with the pitch information.

A note usually occurs at *onsets,* which can be defined as "a single instant chosen to mark the temporally extended transient." [3]. There are several methods to achieve onset detection. Because we are analyzing audio of monophonic nature, the initial hit of each note being played will be very distinct. Therefore, the choice of method is through *energy* based onset detection.

Before applying the energy onset detection method, an optional pre-processing stage can be applied to filter the result. Pre-processing is basically a function that applies Fourier transform to the audio signal and chooses a band to attenuate or boost in the frequency domain. Then, using the inverse Fourier transform, it recreates an audio signal with altered frequency components. This step is recommended if the monophonic instrument is definitely playing within a certain frequency band. The processing will enhance prominence of this instrument and fade unwanted frequencies.

To apply the energy onset detection method, framing is applied through defining hop size and window length. Then by taking the sum of the square of the windowed we defined an *Energy Envelope* function.

$$\mathrm{E}_w^x(n) := \sum_{m=-M}^{M} |x(n+m)w(m)|^2$$

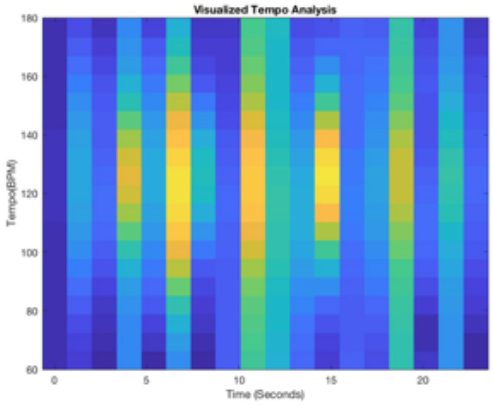
Source: AME 477 Topic 5 Lecture Slides

This function detects the energy of each frame. Because energy rise occurs at a small instance, the derivative of the energy change will give a more precise depiction of where the onsets occur. The derivative is applied as a comparison between the energies of different frames and determine the direction of change. The conditional loop will keep the frames that have positive derivative, hence increase in energy, and st the negative derivatives to zero. This stage is sometimes referred to as rectifying.

As a result, the program will generate an array called *"onset strengths"* which are the respective frame locations and their rectified energy envelope, hence "strength". If a threshold is set,

onset locations can be found using the “find” function in MATLAB.

Now that onset locations are determined, there is one more factor that would be helpful for accurate MIDI transcription: tempo. The concept of finding the tempo is rather simple. Another segmentation is applied to the *onset strength* array to obtain a series of frames containing the onsets. Then, it creates a spectrogram with the Fourier transform of each onset frames. This spectrogram is used for determining the frequency contents of each onset frames. Visually analyzing the spectrogram, a visible horizontal frequency line will be predominant frequency of the onset appearances. Converting this frequency to its period and use 60 seconds to divide by this value will result an estimated result for BPM. A sample tempogram looks like this:



4. Outputting to MIDI

Once the audio file has reached this point and gone through the pitch and beat detection, it is ready to be outputted to a MIDI file. First, the values within the frequency array are averaged between each onset and then compared to a MIDI matrix containing the MIDI note values with their corresponding frequency values, and the frequency that is closest to the frequency of the note has its MIDI number assigned to that note. Then, the miditoolbox is used and the arguments for a matrix to be converted to MIDI information are onset instance in beats, duration in beats, channel, note number, velocity, onset instance in time, and duration in time [2]. Since quantization can be done within any DAW, the onset instance in beats was calculated by dividing the BPM by 60 and then multiplying this number by the onset instance in time to roughly get the beat at which each note occurs. A similar method is used to calculate the duration in beats and the velocity is arbitrarily set to 80. The matrix is then put into a piano roll function to visually display it, as shown in figure 4, and the matrix is then outputted as a “.mid” file using a built in function within the miditoolbox.

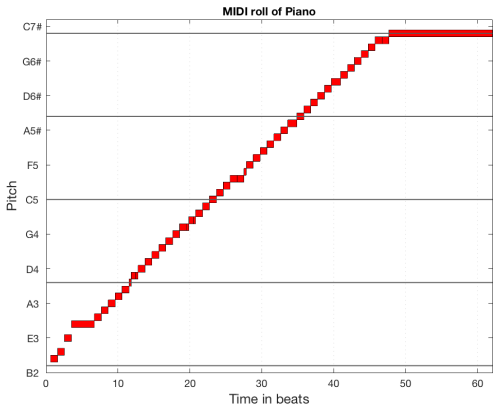


Figure 4. Sample piano roll output after a rising piano file has run through the algorithm.

11. Conclusion

The method displays an accurate MIDI result. Some of the onsets, however, may be false detections caused by the variation of strength within a note. This can be improved by isolating the outliers and evaluate their contents with narrow frames.

To improve the results, the functionality of quantizing MIDI notes to the nearest beat can be developed for a more neat MIDI output. This would require an implementation of beat tracking use the beat locations as references for MIDI output resulting in a much more accurate representation of the input signal and would not require quantization within a DAW.

12. References

[1] de Cheveigne, A., & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. JASA.

[2] Toiviainen, P., & Eerola, T. (2016). MIDI Toolbox 1.1. URL: https://github.com/miditoolbox/1.1

[3] J. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, “A tutorial on onset detection in music signals,” IEEE Trans. on Speech and Audio Processing, vol. 13, no. 5, 2005.