MIDI MATRIX TRANSCRIPTION USING CEPSTRAL AND AUTOCORRELATION BASED PITCH DETECTION WITH TIMBRE MODELING USING MATLAB

Nicholas Bruno, Daniel Kannen

Audio Music Engineering, Department of Electrical & Computer Engineering, University of Rochester

ABSTRACT

The purpose of our project is to create a MATLAB program that will output a MIDI matrix when given a monophonic sound file. This can be achieved through pitch detection using either the autocorrelation or the cepstral based method, both of which we will consider in this research, as well as onset detection using either a spectral based or an energy based approach. We will then use cepstral coefficients to determine the general timbre of the sound file, which we will use to give its track number within the MIDI matrix. By doing this we will not only be able to sample a monophonic audio file directly into a MIDI matrix, but we will also be able to use this information to improve our understanding of polyphonic audio file analysis for future research. This project shows that while possible, accurate MIDI matrix transcription derived from even a monophonic sound file is not a trivial task, and can always be improved upon.

Index Terms— Signal Processing, Matlab, MIDI, Audio, Transcription, Timbre

1. INTRODUCTION

MIDI transmission is one of the most useful and powerful forms of audio transmission used to date. Coming to fruition in late 1982, MIDI (musical instrumental digital interface) was created as a universal language so that synthesizers and other electronic instruments could communicate effectively, regardless of make or manufacturer, with Dave Smith and Ikutaru Kakehashi being officially credited with creating the idea for and helping implement this protocol [1].

While the main information that MIDI transmits has remained constant throughout its existence (i.e. note on, note off, velocity), it has advanced to contain more data such as standardized MIDI song files, as well as supporting more transmission methods, like FireWire or USB [1]. Shown in figure 1 is this basic MIDI file that we previously described. It is a monophonic MIDI piano roll, which contains the MIDI note value, on and off times, and velocity as displayed through a common digital audio workstation, Fruity Loops Studio. This encompasses the basis of our research, as our final goal is to output a file that can be read into a DAW and sent into a piano roll as such.

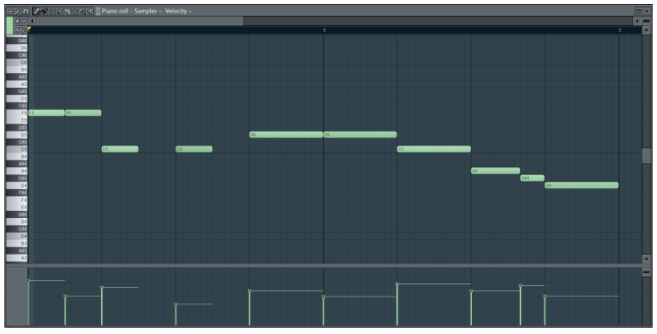

Figure 1: MIDI piano roll as displayed in most DAWs

2. METHOD OVERVIEW

To achieve our goal, we have developed a three-step plan which we will use to retrieve all of the information we need from our audio file to be transcribed into our MIDI matrix. The methods are, in the order of their use; pitch detection, onset detection, and cepstral analysis. We then will take this data and use it to create our MIDI matrix, which we will discuss later in this paper.

2.1. Pitch Detection

Before outputting a MIDI note number, we must first detect the frequency of the note being played in hertz (Hz) within our audio file. We implemented two different algorithms to retrieve this data; a cepstral based method and an autocorrelation based method.

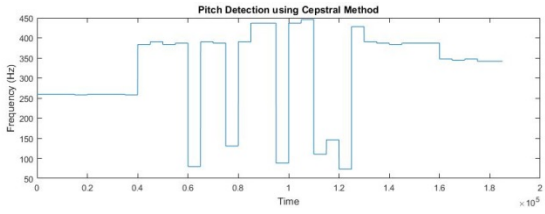

Figure 2: Detected pitch of the first few notes of "Twinkle Twinkle" using the cepstral based method

The cepstral based approach was implemented through two different MATLAB functions. The first function was used to calculate the cepstral coefficients of a windowed section of our signal, while the second function then found the location of the maximum cepstral value of each window, respectively. These locations on the quefrency axis correspond to the fundamental frequency of the windowed signal [2]. We then convert this value back to frequency to be referenced later. This frequency in Hz as a function of time is shown in figure 2. Here we can see the error that is encountered using this method, as transients are being mistaken for frequency, which will cause problems in later analysis.

As we came to realize, the autocorrelation approach to pitch detection was much more accurate for use within the scope of our project, and its implementation is fairly similar to that of its cepstral counterpart. For this method we first cross-correlate our function with itself, and then find the maximum value of this correlation within a windowed section of the signal, which corresponds to the fundamental frequency of the audio being analyzed [2]. In figure 3 we can see the effectiveness of this method, and how much more accurate it is than the cepstral method when applied to the same audio file.

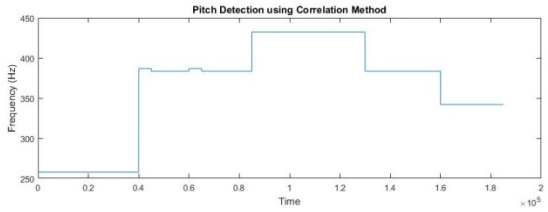

Figure 3: Detected pitch of the first few notes of "Twinkle Twinkle" using the auto-correlation based method

2.2. Onset Detection

While developing our program, we experimented with two different forms of onset detection; spectral based and energy based. Both of these techniques offer unique advantages, as we will discuss. However, we decided to implement energy based onset detection within our final product, and we believe that it accomplishes the task nearly perfectly.

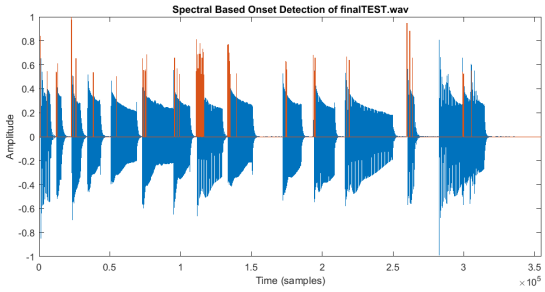

Figure 4: Spectral based onset detection results

While experimenting with the implementation of spectral based onset detection, we noticed that we were not achieving the accuracy that was necessary for our project. As evident in figure 4, onsets were being detected, however they were not always lined up correctly, and we also noticed the detection of "false" onsets. This led to the creation of MIDI notes in the final output that were not present in the original score.

When exploring the capabilities of energy based onset detection, we noticed much improved accuracy over spectral based onset detection. We believe that this was mainly due to the fact that the audio file being analyzed is monophonic, so all onsets are fairly obvious and abrupt. For this method we compare the energy of a point in time against the energy of a point one sample further in time. If the change in energy is large and positive, we know we have encountered an onset. In figure 5 we see how accurate this method proved to be with our test file.

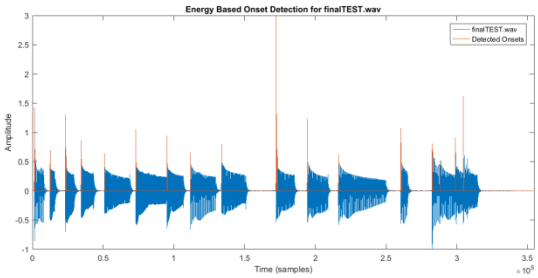

Figure 5: Energy based onset detection results

2.3. Timbre Modeling

With the timbre modeling portion of our project, we wanted to provide the user with an instrument recommendation congruent with the timbre of the input audio file. This will give the user an easy way to pick a sound to reproduce the original audio file within a DAW, if so desired.

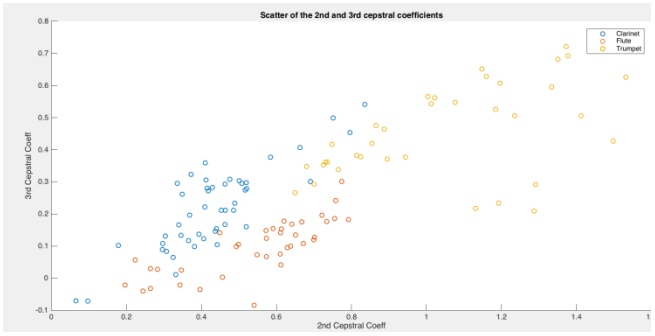

Figure 6: Example of 2nd and 3rd cepstral coefficients of trumpet, clarinet, and flute

To do this, we first built an extensive library of cepstral coefficients from a variety of instruments. As evident in

figure 6, for the most part, each instrument occupies its own space in the quefrency domain.

The first step in building this library was downloading note samples from these instruments. The Philharmonia Orchestra of London has thousands of samples available for download from their website [3]. Each instrument pack includes about 700 separate note samples, each one with varying pitch and velocity. We fed these samples through a MATLAB program and took the first 21 cepstral coefficients of each note. We then averaged the coefficients to provide an easier way to search our database. In our main program, we call a method compare_Ceps.m, which gets the 2nd cepstral coefficient of the input file and compares it with the database of cepstral coefficient averages that we have compiled. The program is able to provide the timbre recommendations based on this comparison.

2.4. MIDI Matrix Conversion

The final step in exporting our MIDI file is converting the information that we have retrieved into a MIDI matrix. This MIDI matrix can then be fed into a matrix to MIDI converter for use or further analysis in any digital audio workstation. To do this, we used a MATLAB function written by Ken Schutte that performs this matrix to MIDI conversion to a very accurate extent.

3. EXPERIMENTATION

To test the output and overall accuracy of our program, we created two monophonic audio test files within Logic. We then compared our programs output to the original wav file as well as against another commonly used and freely available MIDI transcription program called TONY. In this way, we could not only check the overall accuracy of our program, but also see how it stacks up against similar programs that already exist.

Overall, our program performed considerably better than TONY when processing the same audio file. In figure 7 we can see the original audio file (finalTEST.wav) matched against our program's output as well as TONY's output. Here we can see that overall our program outperformed TONY, most noticeably in onset detection. In the middle and end of the file it is clear that TONY detected notes that weren't present in the original audio file, and created unnecessary MIDI notes in their place. This is most likely due to the transients in the audio, which get registered as onsets. We compensated for this in the sensitivity of our energy based onset detection, which we dialed in to be very accurate through trial and error.

The area in which our program fell short in comparison to TONY is in note duration detection. As you can also see in figure 6, each of our MIDI notes take on an arbitrary value of 0.5 seconds, whereas TONY was able to detect on and off times of the notes which were actually present. We would like to implement a so-called note "offset" detection for this purpose in the future.

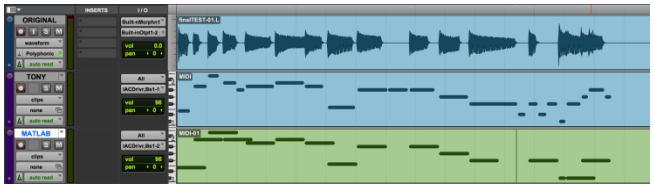

Figure 7: Comparison between the original audio file, TONY's results, and our program's results, respectively.

4. RESULTS AND CONCLUSION

We were very satisfied with the results we achieved as a whole, as they were very much in line with the goals we set for this project before beginning our research. We are confident in saying that we successively wrote and implemented a MATLAB program that could perform monophonic audio analysis and MIDI transcription better than the currently available TONY application. Timbre modeling, which is not available in TONY, was also successfully implemented.

5. FUTURE WORK

We hope to eventually delve deeper into the concept of polyphonic audio analysis and MIDI transcription. We hope that our findings assist in ours as well as others future research in this area. Also, as referenced earlier, we would like to implement a function that can detect note off times. In addition, we would like to eventually write a standalone application separate from MATLAB, which can be provided with audio files and export a MIDI file, and even create/export already synthesized versions.

6. REFERENCES

[1] Anderton, Craig. "Craig Anderton's Brief History Of MIDI." MIDI Association. N.p., n.d. Web. 25 Apr. 2016.

[2] Seo, Naotoshi. "Project: Pitch Detectionon." Naotoshi Seo. N.p., n.d. Web. 02 Apr. 2016.

[3] "Make Music :: Philharmonia Orchestra." Philharmonia Orchestra. N.p., n.d. Web. 20 April 2016.

[4] Schutte, Ken. "MATLAB and MIDI." Ken Schutte. N.p., n.d. Web. 02 Apr. 2016.