Songwriter 3000

Faizan Hassan, Gabriel Ronan, Randall Schliebe

University of Michigan - Ann Arbor, College of Engineering

EECS 452: Digital Signal Processing Design Laboratory

Dr. Hun Seok Kim

ABSTRACT

The Songwriter 3000 is a digital signal processing tool used by musicians in order to efficiently and accurately transcribe sheet music. This report will focus on the digital signal processing methods used in order to fully transcribe music from live recorded audio. By focusing on the digital signal processing techniques used, this project aims to further the use of technology within the world of music. The Songwriter 3000 algorithm performs Fourier frequency analysis on input audio signals, finding pitch from frequency information and rhythm from time information. The final algorithm transcribes audio samples, both ideal and nonideal, with 90% accuracy. Future areas of focus include noise management, intelligently resolving unusual transcriptions, and developing a GUI.

PURPOSE

Songwriter 3000 is a tool to be used by music composers to simplify sheet music transcription. Music transcription can be an arduous and challenging process, requiring multiple attempts, especially for amateur musicians. Even for high-level musicians, the process of transcribing musical ideas to paper is slow. Unlike typical transcription software that requires MIDI data, Songwriter 3000 aims to transcribe sheet music from solely audio input. Only one performance is necessary to fully record a melody, an application that is helpful for amateur and professional musicians alike.

PROJECT DESCRIPTION

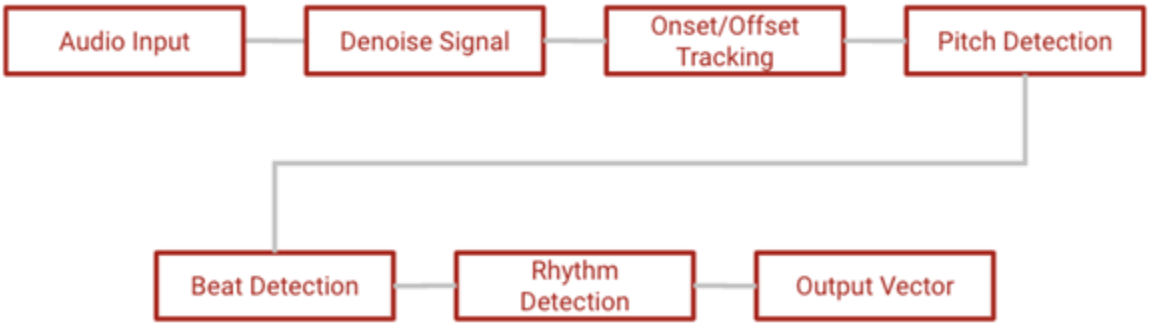

Figure 1. Key Project Algorithms

The Songwriter 3000 system takes audio as input and returns a matrix of digital sheet music. Figure 1 depicts the major process within Songwriter 3000. The tool first asks for the sampling rate, input tempo, time signature, and duration of the recording. While the metronome is running, the program starts recording music. This music then is converted into a vector which is used for the rest of the algorithm. The recorded audio is first denoised in order to remove any unwanted frequencies. The denoised signal is then put through onset/offset tracking. Within onset/offset

tracking, the input tempo and time signature are used in conjunction with the denoised signal in order to find the pitch and maximum volume of each window. The maximum volume of each window is used within beat detection to find the perceived start of each note. The fundamental frequencies and start of each note is then used in rhythm detection in order to produce the matrix of notes. This matrix of notes is a form of rudimentary digital sheet music.

ONSET/OFFSET TRACKING

Writing sheet music involves qualifying sounds with notation that dictate, in a simple sense, time and pitch. As the composer dictates what tempo he/she wishes to play at, this is required from the user as an input. To differentiate between notes, the algorithm observes the audio in both the time domain and frequency domain.

For our program, the bulk of our frequency and time domain analysis is performed within "onset offset tracking." This function requires the input audio as an array, the frequency at which the audio was sampled, and the input tempo in terms of beats per minute and outputs the fundamental frequencies of each note, maximum volume of each window, as well as the number of windows per quarter note.

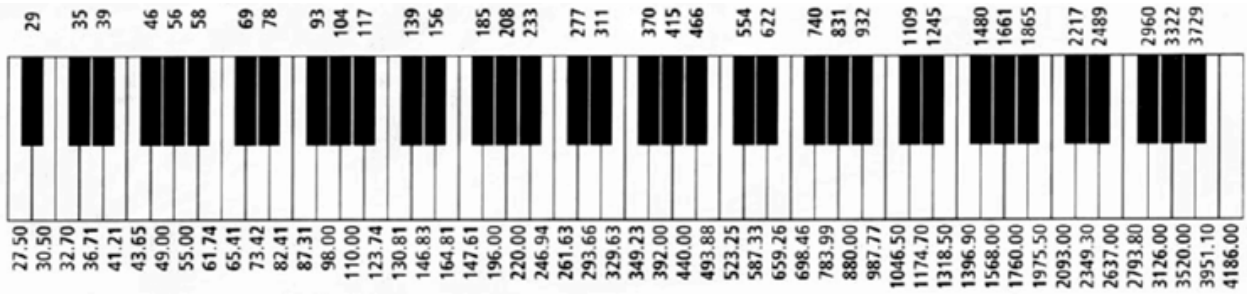

Figure 2. Piano Note Fundamental Frequencies

As evident in figure 2, each note on a piano has a unique fundamental frequency. These fundamental frequencies serve as distinguishing characteristics from one note to another. In order to observe these fundamental frequencies, we must convert our time domain signal to the frequency domain.

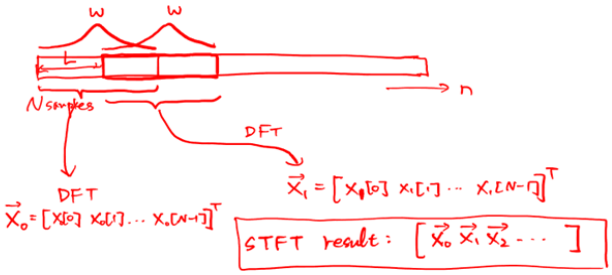

Figure 3. Short Time Fourier Transform Diagram

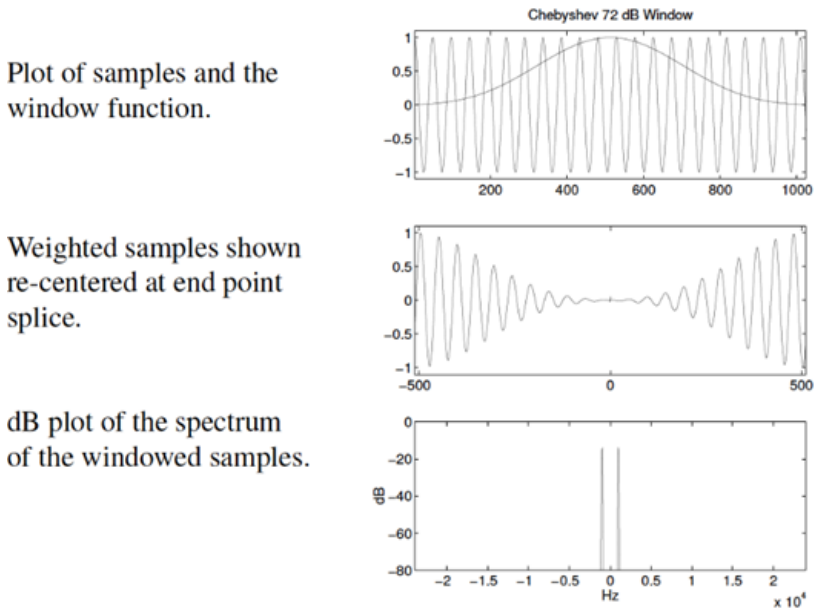

Figure 4. Windowing an Input Signal

Figures 3 and 4 represent the methods used for frequency domain analysis. Figure 3 is a visual used to explain the Short Time Fourier Transform, STFT. As shown on the diagram, the input signal is segmented into signals of length N where each smaller vector, x_n, has an overlap of length N-L with the previous vector. Before applying the Discrete Fourier Transform, DFT, to each vector, a Chebyshev window of length N is applied, and the result is illustrated by the second graph of figure 4. The third graph of figure 4 represents the decibel plot of the DFT of one x_n. The peaks observed are crucial when determining what note is being played as they represent the fundamental frequency of the note.

Before obtaining fundamental frequencies, N and L must be determined such that the windowed signals are synchronized with downbeats and rhythmic subdivisions given time signature. For example, in 4/4 time, N and L are calculated such that each window typically contains one 16^{th} note. The value of N is determined such that the window moves at logical rhythmic intervals. The window's movement is "interpolated." It moves by non-integer values before its location is rounded to the nearest segment of discrete signal.

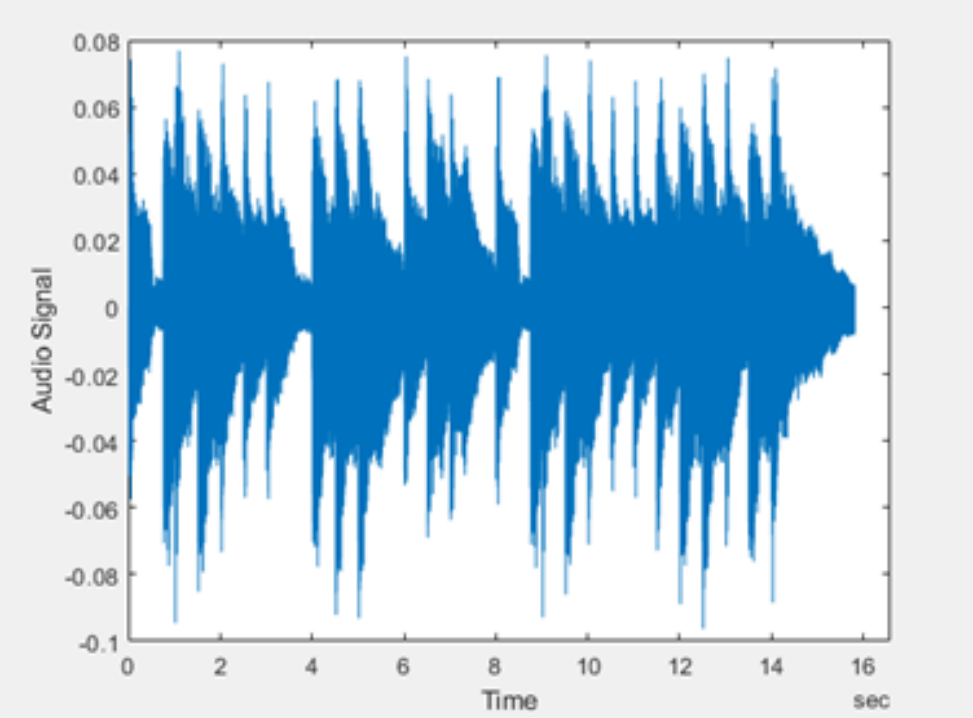

Figure 5. Audio Signal in Time domain

As tempo is used to measure the number of downbeats per second, N can be used with the sampling frequency in terms of samples per second to find the number of samples per downbeat.

$$\frac{Minutes}{Beat} * \frac{60\ Seconds}{1\ Minute} * \frac{Samples}{Second} = \frac{Samples}{Beat}$$

$$\frac{Samples}{Beat} * \frac{1}{4} = N \quad if \quad \frac{log_2(numerator) \subseteq W}{log_2(denominator) \subseteq W}$$

$$\frac{Samples}{Beat} * \frac{1}{6} = N \quad if \quad \frac{numerator\ \%\ 3 = 0}{denominator = 8}$$

Figure 6. Discretizing N

Once N is determined, the input audio signal can be segmented, and findPitch and FFT can be used to determine the fundamental frequencies of each note. The max function is used to determine the maximum volume of each segment.

PITCH DETECTION

One of the two primary musical elements of a melody is pitch. Pitch is the perceived highness or lowness of a note. The word “perceived” is key; pitch is not truly a physical property of an audio

signal but rather a psychoacoustic property. While an audio signal's pitch can be generally described by its frequency, pitch is ultimately a subjective quality. Although this is the case, there exist conventions for describing the pitch of an audio signal via its frequency content.

The pitch of a musical note is typically quantified by its fundamental frequency. If a piano tone has a pitch of 440Hz, this is understood to mean that the tone's fundamental frequency is 440Hz. Fundamental frequency is the lowest-frequency sinusoid present in a signal—consider the superposition principle of periodic signals. If an audio signal is represented in the frequency domain (Figure 7), the fundamental frequency is simply the lowest-frequency peak.

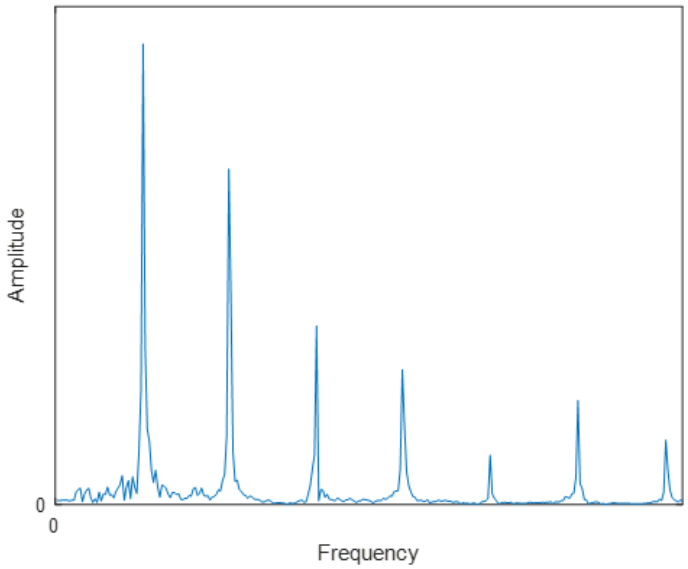

Figure 7. Typical Musical Tone Visualized

Given this convention for describing pitch, a basic pitch detection algorithm would look for the fundamental frequency of a signal. In the frequency domain, two intuitive approaches would be to find the first peak or the absolute maximum. These approaches may work for simple cases. However, low frequency noise could present challenges for the peak-finding approach, and locating the absolute maximum does not work if there are larger peaks at other frequencies. A more robust pitch detection algorithm would consider properties that are more specific to musical tones.

Pitched instruments and human vocal cords are resonators, and as such, they produce harmonics. Harmonics are sinusoids of frequencies higher than the fundamental frequency. They follow the harmonic series, with the frequency of each harmonic being some positive integer multiple of the fundamental frequency. An effective pitch detection algorithm can use this information to better locate the fundamental frequency. The Songwriter 3000 algorithm expects to see harmonics, and by knowing their relative spacing, it finds the harmonic frequency series with the highest

amplitude. It makes copies of the original signal in the frequency domain and scales the copies along the frequency axis by positive integer factors. Overlaying the copies with the original signal results in the peaks of the harmonic series coinciding at the fundamental frequency (Figure 8). By combining the signals elementwise, the algorithm can search for the absolute maximum to find the fundamental frequency. The Songwriter 3000 performs elementwise multiplication. Elementwise addition is also effective, but multiplication will reduce peak height if other signals don't also have peaks. This is desirable assuming that there consistently exist peaks at the first few harmonics.

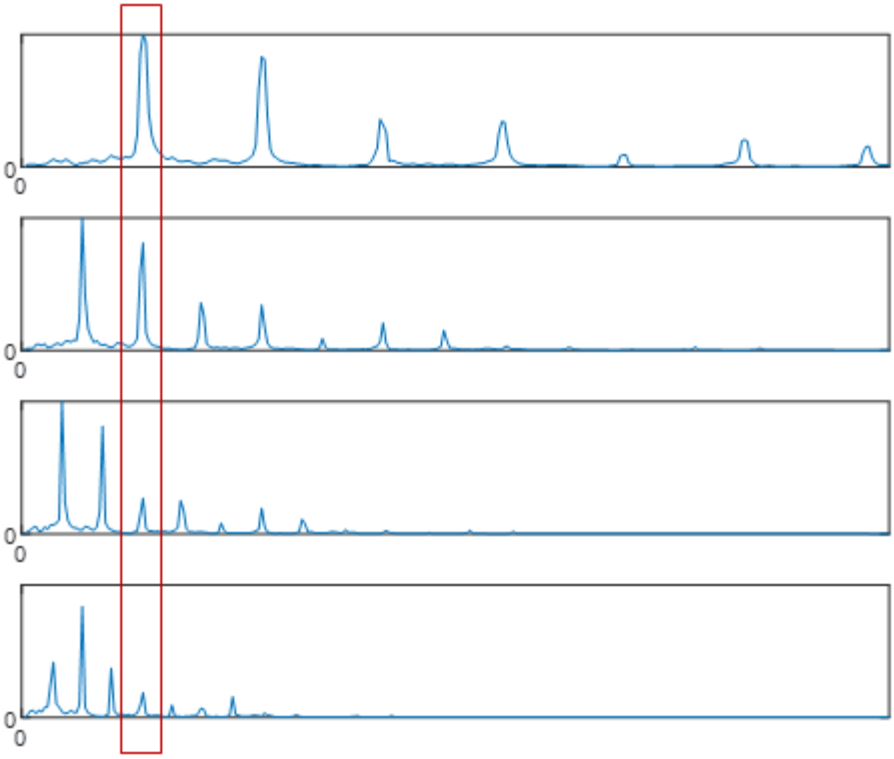

Figure 8. Songwriter 3000 PDA Visualized

With pitch information available in the form of frequency, these frequencies must be put into a form that is chartable on sheet music. The Songwriter 3000 algorithm matches pitches to the 12-tone equal temperament (12-TET) scale. The 12-TET system is the standard tuning system used in Western music, e.g., on the piano. 12-TET divides the octave, i.e., from some pitch to twice that pitch's frequency, into 12 tones with frequency intervals of a single constant factor. The Songwriter 3000 algorithm matches detected pitches to the nearest note in the 12-TET scale. Note that the scale is logarithmic.

BEAT DETECTION

Once the pitches are found for each window, the length of each note is found by identifying the onset of that note. This is later used with the fact that the offset of the note is assumed to be when the next rest or next note is detected. A rest is identified using a lower bound that is a function of the maximum amplitude. If a constant factor was used, 3% to 5% of the maximum amplitude worked best. To find the onset of the note, frequency is analyzed. A relative change in frequency between windows means that a beat is detected there. If the change in frequency was greater than a fraction of the detected frequency range, it counts as a beat. If the frequency did not change between consecutive notes, then amplitude (volume) is analyzed. If the window contains a peak, then the ascent and descent to the adjacent windows are taken into consideration. In general, a point in the vicinity of a steep ascent is almost always a valid beat. A descent is also required, but its size is not as important. Figures 9 and 10 illustrate the frequency and amplitude over time for the song Mary Had a Little Lamb.

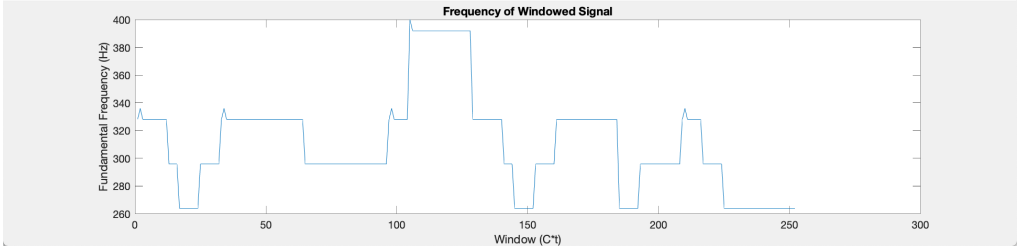

Figure 9. Frequency of the windowed signal Mary Had a Little Lamb

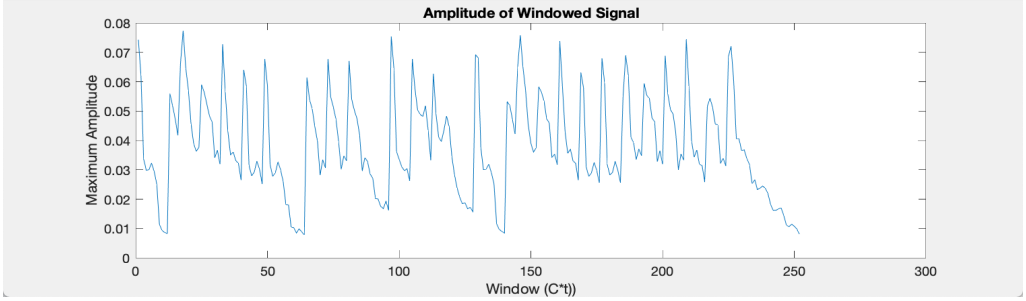

Figure 10. Amplitude of the windowed signal Mary Had a Little Lamb

RHYTHM DETECTION

The second major musical element of a melody is rhythm. While pitch is frequency information, rhythm is time information. Rhythm can be described as the sequential durations of sounds and silences, i.e., rests. In sheet music the duration of each pitch and silence is notated within periodic divisions called measures. Duration is notated as simple fractions of beats or other duration units. A rhythm detection algorithm for generating sheet music must interpret an audio signal's time content in a way that can be charted in sheet music. That is, it must find a realistic

combination of durations, even if the time content of an audio signal doesn't precisely match a chartable rhythm.

Following onset-offset tracking, the Songwriter 3000 driver calls a rhythm detection function. Rhythm detection function requires a matrix of beats and musical tones on the 12-TET scale, and knowing tempo and time signature information, it determines the best-fit sequence of notes and rhythms. This output sequence is a basic form of sheet music language—in terms that follow sheet music conventions.

Basic rhythm interpretation

The basic premise behind the Songwriter 3000 rhythm detection algorithm is that any rhythm can be expressed as a combination of shorter rhythms. For example, an eighth note can be expressed as two sixteenth notes tied together. A rhythm detection algorithm can rely on this premise to determine rhythm from a sequential vector of tones in time. The algorithm must first divide the vector of tones into segments of time based on meter and tempo. Next, it must find the note of best fit for each division. Wherever the note doesn't change between sequential divisions, the algorithm should combine notes into one of longer duration (Figure 11). The Songwriter 3000 algorithm uses this approach, finding the best-fit note in each division via a categorical mode.

```
Notes:          A A A A B B B B C C C C C C C C

Divisions:      A A A A|B B B B|C C C C|C C C C

Result:         A (16th note), B (16th note), C (8th note)
```

Figure 11. Basic Rhythm Mapping Visualization

To reliably map the sequence of notes to rhythms, the rhythm detection algorithm must understand both tempo and time signature of the music sample. Tempo dictates the number of notes that occur in the notes vector per downbeat. Time signature dictates the number of divisions per downbeat in order to properly resolve rhythms written in that time signature.

In order to understand when to combine notes, the algorithm must somehow qualify what it means for a note to "change." There are three main conditions that indicate a change, that is, the end of one note and the start of another. Firstly, if the pitch of the note changes, this indicates the end of a note. Secondly, if the silence precedes or follows a note, this indicates a start or end to a note. Thirdly, if a "beat" occurs, as found by a beat detection algorithm, this indicates the start of a new note. The Songwriter 3000 algorithm first checks for a change in pitch or a silence, and if the pitch remains steady and the signal amplitude remains high, it then checks for a beat.

Differentiating between rhythm types

In preparing duration information to be charted in sheet music, it's necessary that the rhythm detection algorithm only fits note timings to "reasonable" rhythms. High fidelity is not helpful or valuable to the user, as it will produce unreasonable sheet music if the input audio isn't rhythmically flawless. Certain complex rhythms will not be performed in any reasonable case, including highly precise rhythms and polyrhythms. Songwriter 3000 considers factors such as time signature to choose a finite set of reasonable rhythms.

In music, rhythms can be divisions of the meter by different factors. Typically, rhythms appear in divisions of twos and fours; however, divisions of three are common as well. The Songwriter 3000 algorithm is designed to consider both duplets and triplets. To prepare for readable sheet music, the algorithm requires that like tuplets are grouped together.

One possible way of detecting both duplets and triplets is to use a "least common denominator" rhythm. For example, this would mean using a 32nd note triplet to detect both 8th note triplets and 16th notes. However, this is a rigid approach with unreasonably high precision. A better approach is to look at longer sections of music to determine whether notes are generally distributed in threes and twos. Songwriter 3000 looks at the distribution of notes between each downbeat to determine whether triplets or duplets are more appropriate. It uses a categorical mode to determine the fit of each rhythmic division.

Creating sheet music

Generating sheet music is a task that requires more than a list of note and rest durations. There exist both conventions and rules for properly notating rhythms. The key notation concern for transcribing sheet music from audio is audio sustains. Certain durations cannot be notated by a single symbol, such as sustains between unlike rhythms. In these cases, notes must be connected via tie, and rests must be separated. The final output is a matrix of notes, durations, and tie locations.

RESULTS

The final Songwriter 3000 algorithm outputs a rudimentary form of digital sheet music. The digital sheet music is in the form of a matrix, with one row containing pitch information, one row containing rhythm information, and one row containing information about ties.

```
final =
    44    42    40    44    42    40    40     0     0     0
    12    12    12    12    12    12     4     4     1     4
     0     0     0     0     0     1     0     0     0     0
```

Figure 12. Songwriter 3000 Digital Sheet Music

Figure 12 shows an example of output digital sheet music. Pitches in the top row represented by corresponding piano key numbers. Durations in the second row are described by name, e.g., a “4” for a quarter note. Ties in the third row are indicated by a “1,” and designate a tie to the subsequent note.

When tested against ideal audio samples, i.e., clear, noiseless samples with precise rhythm, the algorithm correctly identified the melody with perfect accuracy (Figure 13). It appropriately notates sheet music given time signature to follow musical conventions.

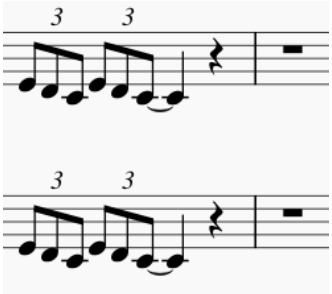

Figure 13. Synthesized Melody (Top) Versus Recreated Melody (Bottom)

When tested against non-ideal samples, Songwriter 3000 produces discrepancies. Below (Figure 14) is a transcription of a tenor saxophone recording, which is in a low register at a high tempo (131 BPM) to challenge the algorithm’s ability to resolve both pitch and rhythm. While one discrepancy is fair, due to error of the instrumentalist, one is a false interpretation.

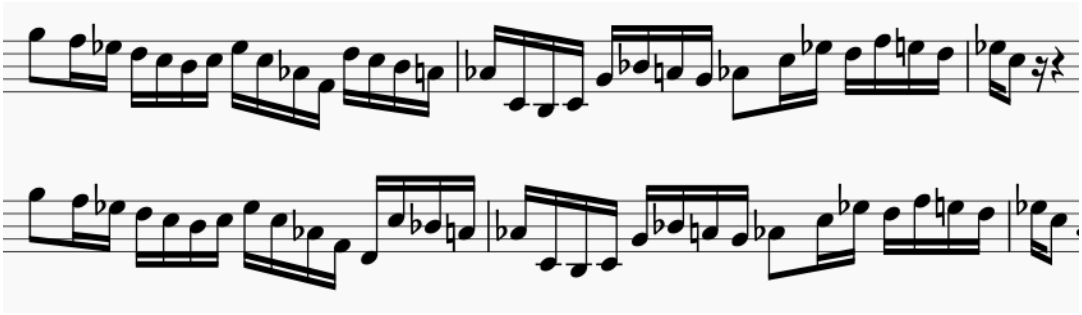

Figure 14. Intended Melody (Top) Versus Transcribed Melody (Bottom)

In some cases, Songwriter 3000 charts low frequency notes when it hears high-amplitude noise (Figure 15). This result is obviously unreasonable

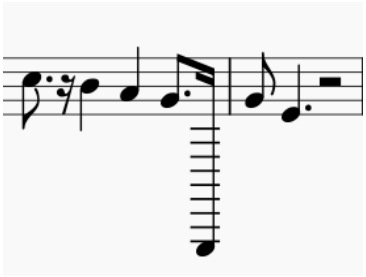

Figure 15. Sheet Music from Noisy Input

Quantifying the algorithm's accuracy is perhaps not revealing. It's important to first qualify its accuracy; the success of the algorithm depends greatly on the quality of the input audio signal. If the input signal is ideal, the algorithm transcribes the melody with around 99% accuracy. In non-ideal cases, it can achieve 90% accuracy, but this could certainly decrease in even poorer conditions.

CONCERNS

The toughest challenge when working with audio is noise. When noise is introduced into an input audio signal, it is difficult to differentiate pitch information from undesired information via frequency analysis. As seen in Figure 15, even though the signal was denoised, some low frequency noise remained as its frequency coincided with that of the fundamental frequency of a note.

```
# Tester file

infile = wave.open("MHALLREMIX.wav")
samples = infile.getnframes()
fs = infile.getframerate()
audio = infile.readframes(samples)

# Convert to float array

audio_int16 = numpy.frombuffer(audio, dtype=numpy.int16)
audio_float32 = audio_int16.astype(numpy.float32)
```

```
signal = audioread('triplettest.wav');
info = audioinfo('triplettest.wav');

fs = info.SampleRate;
player = audioplayer(signal, fs);
```

Figure 16. Python Code vs. Matlab Code

Our second challenge was translating our code from Matlab to Python. Within Matlab there are smaller details that are assumed, such as data type. As seen in figure 16, when converting a .wav file to an array in Matlab the data type is assumed to be float32, whereas in Python, the data type starts as int16 and we must convert it to float32 ourselves. Details such as type casting are handled in Matlab, while in Python, the coder must type cast appropriately such that no important information is lost while conserving memory. As this is the case, Matlab code can not be directly ported into Python and requires more tedious levels of checking. The raspberry pi's

output for both beat and rhythm detection were not as accurate, and did not have time to be completely validated.

FUTURE WORK

As our toughest challenges were handling noisy inputs and translating Matlab to Python, these are the areas that we want to improve the most. Currently, we handle denoising our input audio with a bandpass filter, this filter is not sufficient for handling noise as some noise components can potentially have frequency components within the bandpass. By applying topics such as machine learning we can further differentiate between what is considered a note and noise for more specific applications. Some form of artificial intelligence could also be used to skim the output digital sheet music and identify unusual or nonsensical notes. By utilizing more libraries in Python, we can create a program that is more efficient and less CPU intensive than our current program. Another step we can take to improve our existing system is to include a user interface. Currently, we ask the user for a tempo, time signature, and a duration of the recording through the terminal. By creating a pop-up window for the user, the tool will become more accessible.

CONCLUSION

Our tool, the Songwriter 3000 is a music transcription software which provides an efficient alternative to transcribing by hand. This tool has proved to have exceptional accuracy for ideal samples, high accuracy for non-ideal cases, with a key concern being noise. Though combating noise is the most critical next step in improving our tool, providing a more user friendly interface is critical as well.

REFERENCES

Sengpiel, E. (n.d.). Note Names of Musical Notes. Retrieved December 12, 2020, from http://www.sengpielaudio.com/calculator-notenames.htm

PITCH DETECTION METHODS REVIEW. (n.d.). Retrieved December 12, 2020, from https://ccrma.stanford.edu/~pdelac/154/m154paper.htm

E. Benetos, S. Dixon, Z. Duan and S. Ewert, "Automatic Music Transcription: An Overview," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 20-30, Jan. 2019, doi: 10.1109/MSP.2018.2869928.

Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H., & Klapuri, A. (2013). Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, *41*(3), 407-434.

Vaca, K., Gajjar, A., & Yang, X. (2019). Real-Time Automatic Music Transcription (AMT) with Zync FPGA. 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). doi:10.1109/isvlsi.2019.00075

Michigan, W. (n.d.). The Elements of Music. Kalamazoo, MI: West Michigan University. doi:https://wmich.edu/mus-gened/mus150/Ch1-elements.pdf

Coca, A. E., Tost, G. O., & Zhao, L. (2010). Characterizing chaotic melodies in automatic music composition. *Chaos: An Interdisciplinary Journal of Nonlinear Science, 20*(3), 033125. doi:10.1063/1.3487516

Johnteslade. (2005). *STFT - windows* [Digital Image]. Wikimedia Commons. Retrieved from https://commons.wikimedia.org/wiki/File:STFT_-_windows.png

(2020). *Fourier Transform* [Digital Image]. National Instruments. Retrieved from https://www.ni.com/en-us/support/documentation/supplemental/06/understanding-frequency-performance-specifications.html

Rockikz, A. (2020, February 19). How to Play and Record Audio in Python. Retrieved December 12, 2020, from https://www.thepythoncode.com/article/play-and-record-audio-sound-in-python

CONTRIBUTIONS

Faizan Hassan: Primary writer of Onset/Offset Tracking, Metronome, and Recording, contributed to Processing, Beat Detection, Note Bins. Helped debug issues reading .wav files in Python and Matlab. Debugged issues using STFT in Python. Contributed to report, presentations, input audio

Gabriel Ronan: Primary writer of the beat_detection, note_match, and denoise functions, as well as all project drivers. Contributed to raspberry pi metronome and recording development. Responsible for carryover from MATLAB to the raspberry pi.

Randall Schliebe: designed pitch detection algorithm, designed rhythm detection algorithm, designed sheet music language, established STFT windowing conventions, updated note_match