Brandon Wright

ECE 6950

Fall 2023

**WAV-to-MIDI Audio Processing MATLAB Program**

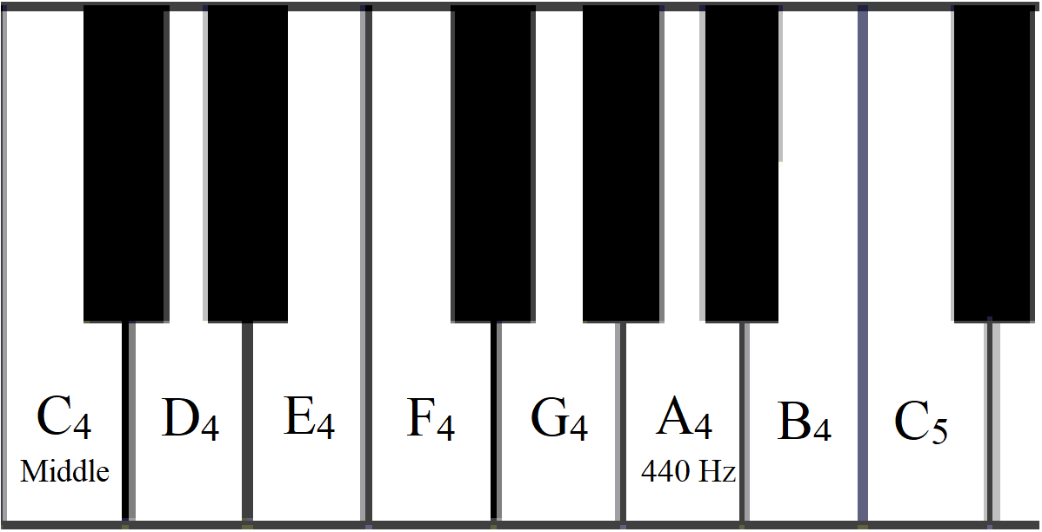
**Abstract –**

**I. INTRODUCTION**

*A. Music Notes and 12-TET Tuning*

Source DSP First chapter 4

* Insert picture of keyboard with keys

****

**Figure 1.** Music notes for one octave on keyboard

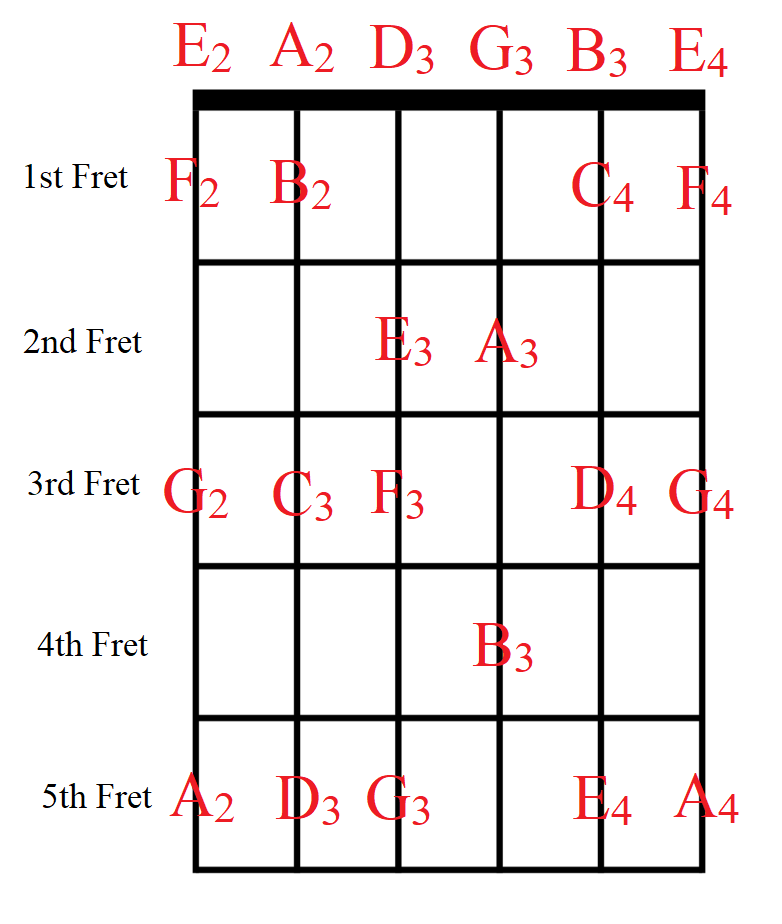
* Insert equation for keyboard notes

**Equation 1.** Converting frequency to music note

(DSP first? Or is there a better source?

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Note** | **C4** | **C♯4/D♭4** | **D4** | **D♯4/E♭4** | **E4** | **F4** | **F♯4/G♭4** | **G4** | **G♯4/A♭4** | **A4** | **A♯4/B♭4** | **B4** |
| **f0 (Hz)** | ~262 | ~277 | ~294 | ~311 | ~330 | ~349 | ~370 | ~392 | ~415 | 440 | ~466 | ~494 |

**Figure 2.** Frequencies of octave beginning with Middle C (C4)

* 

**Figure 3**. Notes on guitar fret board

* Explain how electric guitar works

*B. Musical Instrument Digital Interface (MIDI)*

Source <https://www.instructables.com/What-is-MIDI/>

* Explain what it is
* History
* Hardware and software compatibility
* Use in the music industry
* Sequence of bytes/file format
* How MATLAB handles midi
* MIDI inputs and outputs on computer
* DAWs and audio recording
* VST Instruments

*C. Digital Audio Sampling*

- ADC and sampling frequency

- Info on how windows handles sound

- Wave file format

- Technical specifications on M-Track Solo

*D. Pitch Detection with CREPE Neural Network*

- What is a neural network

- How does it learn/how is it trained

- More about CREPE in matlab

*E. Physics of sound*

- Sound pressure and volume

- How brain detects pitch

- Fundamental frequency and harmonics

**II. METHODS**

The audio from the guitar was recorded and processed using a custom script in MATLAB. **Code Snippet 1** shows the program that runs while the script is live. The script records ten seconds of audio from the Windows audio input, and then processes the recording and plays back the MIDI messages. First, the MIDI devices are initialized using functions from MATLAB’s Audio Toolbox plugin. There are multiple MIDI outputs in Windows, and they can change based on software or hardware configurations. MIDI mapper plays the MIDI message back in the default piano sound over the current Windows audio output. Loop MIDI takes the outputted MIDI message and loops it back into Windows as a MIDI input. Loop MIDI is chosen as the output when you wish to record to a MIDI message with a DAW.

After the MIDI outputs are initialized, the MATLAB audio recorder is configured. A sampling frequency Fs of 44,100 bits/s was chosen to obtain a high-quality recording. The bit depth of the recording was set to 16 bits, and the recording was performed on one channel, i.e. in mono. The recording duration was set to 10 seconds. In the main while loop, audio is recorded for ten seconds and audio recording is processed by the create\_midi script to create the MIDI signal. The script then uses the Audio Toolbox MATLAB function midisend to send the MIDI message to the selected MIDI output device. Finally, the while loop in the main script pauses for 11 seconds while the MIDI message plays, and the while loop runs again until the script is terminated by the user.



*A. Create MIDI Algorithm*

The following algorithm was used to convert the WAV file into a MIDI signal. First, the fundamental frequency of the notes were found using the CREPE neural network MATLAB plugin. Next, the notes of the frequencies were found using **Equation 1**. Next, the index of the note array where the notes changed were found and a for loop was run for every one of those change indeces.

In the for loop, the note at the change index of the current iteration of the for loop was found, along with the max sound pressure during which the note was played (note peak). Next, the length of the note was determined by finding the number of samples between the current change index and the next index.

Next, another for loop put together a midi message using the midi notes, used the note peak to find the note volume, and determined the duration and timestamp of the note using the sampling frequency Fs. An array of these midi messages was then returned by the function to be played by the main script.

**Algorithm:** WAV-to-MIDI Conversion

**Input:** N samples of soundwave amplitude for n = 1, 2, 3, … N

Find f0[n]

Find notes corresponding to f0[n]: note = 12\*log2(f0/440) + 49

Find sound pressure levels Lp[n]

Find values of n where notes change (change\_indeces)

Find note\_peaks, note\_lengths, and midi\_notes

for (i = 1; i++; i < length(change\_indeces))

if i == 1

midi\_notes(i) = notes(change\_indeces(i));

note\_peaks(i) = max(Lp(i:change\_indeces(i)));

note\_lengths(i) = change\_indeces(i);

else if i == length(change\_indeces)

note\_peaks(i) = Lp(i);

midi\_notes(i) = notes(change\_indeces(i));

note\_lengths(i) = change\_indeces(i) - change\_indeces(i-1);

else

midi\_notes(i) = notes(change\_indeces(i));

note\_lengths(i) = change\_indeces(i) - change\_indeces(i-1);

peak = max(findpeaks(pressure\_levels(change\_indeces(i-1):change\_indeces(i))));

if(isempty(peak)); peak = 0; end

note\_peaks(i) = peak;

end

end

Find time t = 0:1/Fs:length(notes)/Fs

Generate MIDI signal

for(i = 1; i++; i < length(change\_indeces)

if(midi\_notes(i) == 0)

volume(i) = 0

else

volume(i) = floor(127\*note\_peak(i)/100);

end

midi\_message =

{

command = “Note”;

channel = 1;

note = midi\_notes(i);

velocity = volume(i);

duration = note\_lengths(i)/Fs

timestamp = t(change\_indeces(i - 1) + 1);

}

end

**Algorithm 1.** WAV-to-MIDI conversion

*B. Determining Pitch and Note*

The fundamental frequencies of the recording were estimated using the pretrained CREPE convolution neural network. The CREPE network is available as part of the Audio Toolbox plugin for MATLAB. CREPE makes estimating pitch very easy, as can be seen in **Code Snippet 2**. The CREPE training data was downloaded from the MathWorks website and unzipped in the same path as the main script. However, the length of the array of frequencies created by CREPE is not the same length as the input array. Accordingly, the array of frequencies needs to be stretched out to reach the right number of samples, as can be seen in **Code Snippet 2**.



**Code Snippet 2**. Determining the pitches of the recording

Next, the musical notes that correspond to the pitches are calculated using **Equation 1**, as can be seen in **Code Snippet 3**. Because a guitar must be manually tuned, every string might not be exactly have the right pitch as the desired note. Therefore, the floor of the note calculated by **Equation 1** must be found because a MIDI message must have a note that is an integer. Additionally, because the CREPE network assigns a value of NaN to a pitch it cannot detect, all instances of NaN in are set to 0, which is a valid MIDI note value.



**Code Snippet 3.** Determining musical note from frequency

*C. Determining note length and timing*

The note length and timing is determined by determining what indeces in the recording array correspond to a change in the note (change\_indeces), and using the difference between concurrent values of change\_indeces along with Fs to determine the length of the note in seconds. **Code Snippet 4** shows the code that implements this procedure. Next, an array of time values the same length as the recording was created using Fs to come up with a timeline in seconds. In the last for loop, the value of the change index for a particular note was used as the index for the value in the time array (t) to get the moment in time the note was first played. Both the timestamp and the note duration were saved in the MIDI message for each particular note.



**Code Snippet 4**. Determining note length and timestamp

*D. Determining note volume*

The volume of a note was determined by calculating the sound pressure generated by a sound wave. Because a sound wave has peaks a troughs, volume is correlated to the total pressure created in the air. As seen in **Code Snippet 5**, MATLAB’s Audio Toolbox has a built-in function to calculate sound pressure from a recording. Because the sound pressure might not reach a maximum until after the note is initially strummed, the peak value of the pressure was found during the duration of the note, and then normalized to the scale required by the velocity parameter of a MIDI message, i.e. an integer between 0 and 127.



**Code Snippet 5**. Determining note volume.

CREPE is very sensitive, and it is possible to change the pitch of the guitar without strumming a note. For example, when lifting a finger off a guitar fret, it is possible that a harmonic could ring out before another note is strummed on that string. CREPE picks up these harmonics very well, so the volume must be set to zero for notes that do not create a peak in the sound pressure. If only notes that trigger sound pressure have volumes, the harmonics will not make it into the MIDI message and the guitar riff will sound more like what the human ear hears.

*E. Creating MIDI Messages*

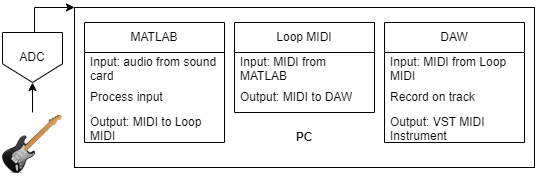
MIDI messages can be created using the MATLAB Audio Toolbox plugin. Different MIDI instructions can be encoded, including note, note on, note off, pitch bend, channel pressure aftertouch, or all sound off. Each MIDI command has a value for channel, note (pitch), velocity (volume), and timestamp. “Note” MIDI messages also have a value for duration. **Code Snippet 6** demonstrates how a MIDI message was created in MATLAB. The comment in the code shows the format of a MIDI message

**Code Snippet 6**. Creating the MIDI messages

*F. Hardware and Software Setup*

Audio was recorded by plugging a Fender Telecaster electric guitar into the ¼ inch line input for an M-Audio M-Track Solo USB recording interface. The M-Track solo was connected through a USB to a personal computer. Audio processing was performed in MATLAB using plugins from MathWork’s Audio Toolbox library. A new MIDI track was created and set to record in the DAW, and a new audio recording was started.

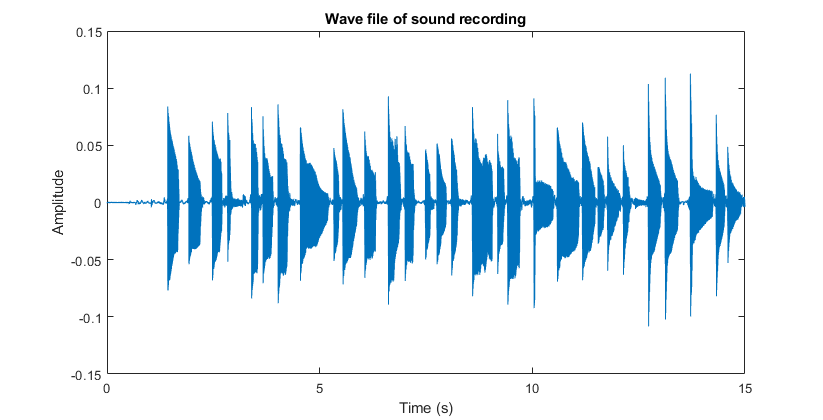
A Guitar short guitar riff was recorded in MATLAB and run through a custom script to determine the melody’s change in pitch, their corresponding musical note values along with the timing and volume information of all the notes. The script then generated a matrix of bytes from a MIDI signal, which encoded the note, its volume, its duration, and its timestamp. The MIDI signal was then streamed into Loop MIDI, which took the signal from the selected MIDI output port and looped the MIDI back into a Windows MIDI input port. Next, the DAW took the input from Loop MIDI and recorded the signal into the track as it was played in MATLAB. The recording session on the DAW was ended, and a track was successfully recorded. Next, by changing the VST MIDI instrument plugin, and desired digital instrument could be used to play back the notes of the guitar riff. Additionally, when opening up the MIDI track in the DAW, individual MIDI notes could be dragged and dropped to change pitch or timing.



**Figure \*\*\*.** Audio hardware and software setup

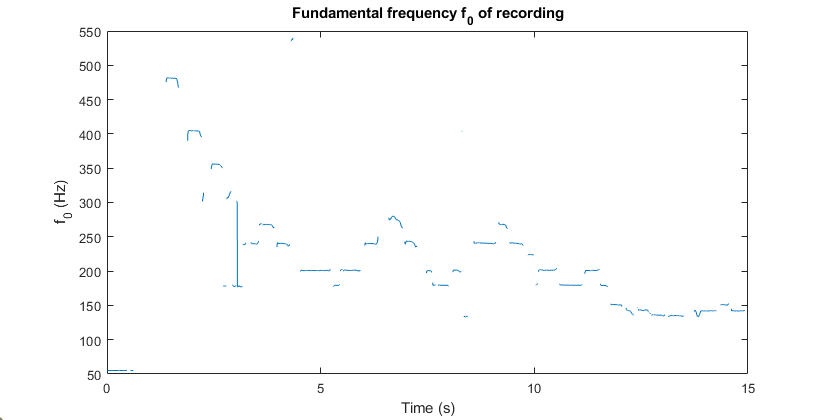
**III. RESULTS**

*A. Setting up MIDI Ports and Recording Audio*

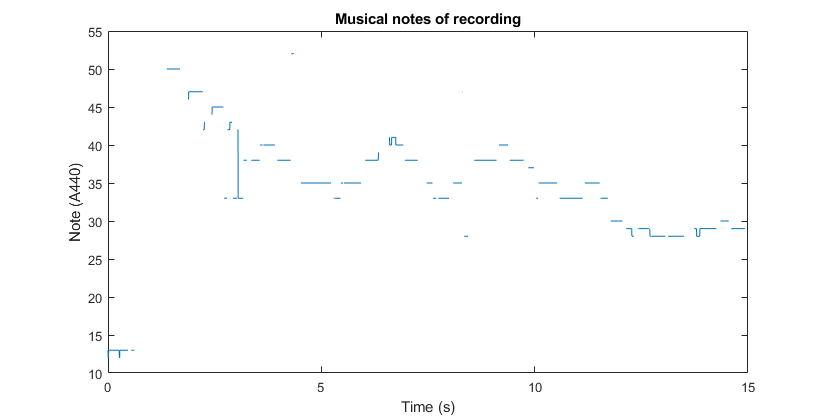


**Figure \*\*\*.** Sound wave of recording

*B. Detecting Pitch and Note*

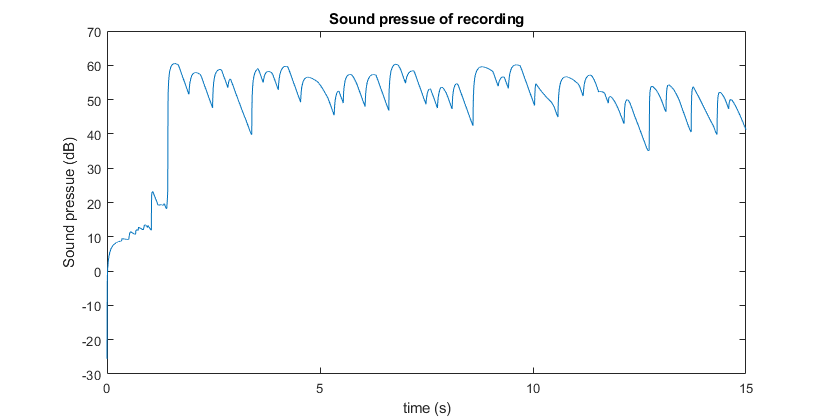


**Figure \*\*\***. Fundamental frequency of open strings on guitar



**Figure \*\*\*.** Keys of open strings on guitar

*C. Detecting sound pressure*



**Figure \*\*\*.** Sound pressure of recording

*D. Description of how well it worked*

**IV. CONCLUSION**

* Summary
* Possible improvements
* Lessons learned

**References**