Brandon Wright
ECE 6950
Fall 2023

WAV-to-MIDI Audio Processing MATLAB Program

Abstract – MIDI is a universal technical standard that can be used by a digital synthesizer to create music that can sound like virtually any instrument conceivable. This paper proposes a methodology and algorithm that takes a digital recording of a melody created by any physical instrument and reproduces the same melody on MIDI-based instrument by detecting the change in pitch and volume with respect to time and creating a MIDI data stream that can be read in real time by any MIDI-enabled instrument or software. Using an off-the-shelf USB-based audio interface connected to a personal computer, an audio clip is recorded and its frequency, timing, and volume are processed by a MATLAB script to create a MIDI signal that can be recorded or played back on any MIDI-compatible device or program connected to the personal computer.

INTRODUCTION

With the proliferation of affordable microphones and digital audio recording software, recording music at home has become an affordable alternative to professional recording[1]. Technical standards such as Musical Instrument Digital Interface (MIDI) provide a universal framework for musicians to create, store, and play back music with digital synthesizers that can mimic the sound and timbre of virtually any instrument[2]. Social media platforms such as SoundCloud allow musicians to release their music to listeners on any nearly any budget[3]. Additionally, learning an instrument has never been easier and more affordable, with websites such as Justin Guitar and YouTube offering free tutorials to help amateur musicians become proficient in many musical instruments, such as the guitar or keyboard[4]. The ability to read sheet music is no longer a prerequisite to create high-quality music.

This paper proposes a methodology to help amateur musicians create and play back music on any digital instrument by simply recording a clip of music on any physical instrument. Using a MATLAB script, the change in pitch and volume of such a recording is measured and processed to create a MIDI data stream that can be read in real time by any MIDI software or instrument connected to a personal computer. Using this script, the musician can play a melody on one instrument and play back and record the same melody on the digital recreation of any other instrument. By using this script, creating music with MIDI no longer requires proficiency on a keyboard-based instrument or music notation. The barrier of entry to music creation is dramatically lowered so that anyone who can play—or even hum—a melody

can create high quality music that can be played and recorded on any instrument or machine that is MIDI compatible.

Musical Notes and Equal-temperament Tuning

Music theory is based around the pitch or frequency of individual notes. The most fundamental interval in music is the octave[5]. When given a base note, or root, with frequency f_0, a second note an octave higher has a frequency of $2f_0$[5]. A note that is an octave lower than the root has a frequency of ½ f_0[5]. An octave can be divided up into as many notes as desired, but an octave comprised of 12 notes was canonized during the inception of Western music[6]. The piano keyboard, as shown in **Figure 1**, is a common way of depicting the 12 notes in an octave. As music theory became more standardized, alphabetic letters (A through G) were assigned to white keys on the keyboard, as seen in **Figure 1**. An octave can begin on one any one letter, such as A, and by the time all the letters have repeated on the keyboard, a new octave has been reached. Notes that are an octave apart can be intuitively heard because the frequency of the higher note is perfectly divisible by the frequency of the lower note, as seen in **Figure 2**[6]. Put another way, one period of a musical note contains exactly two periods of a note one octave higher than it.

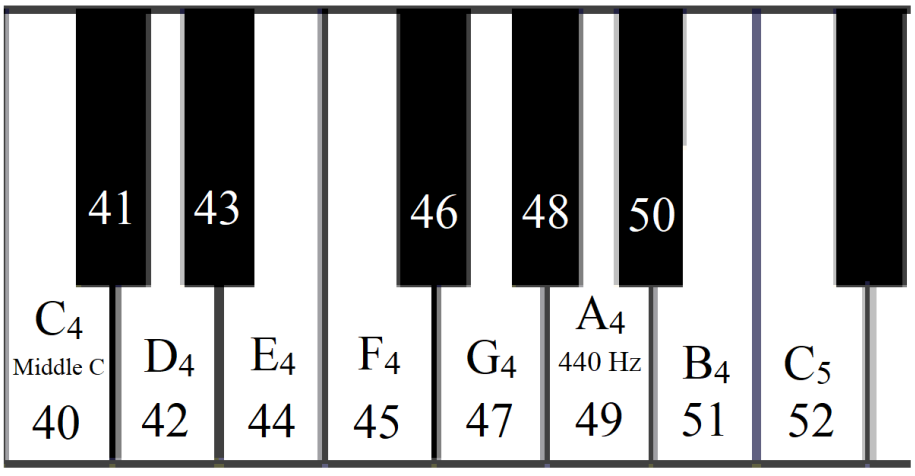

Figure 1. Musical notes for one octave on keyboard

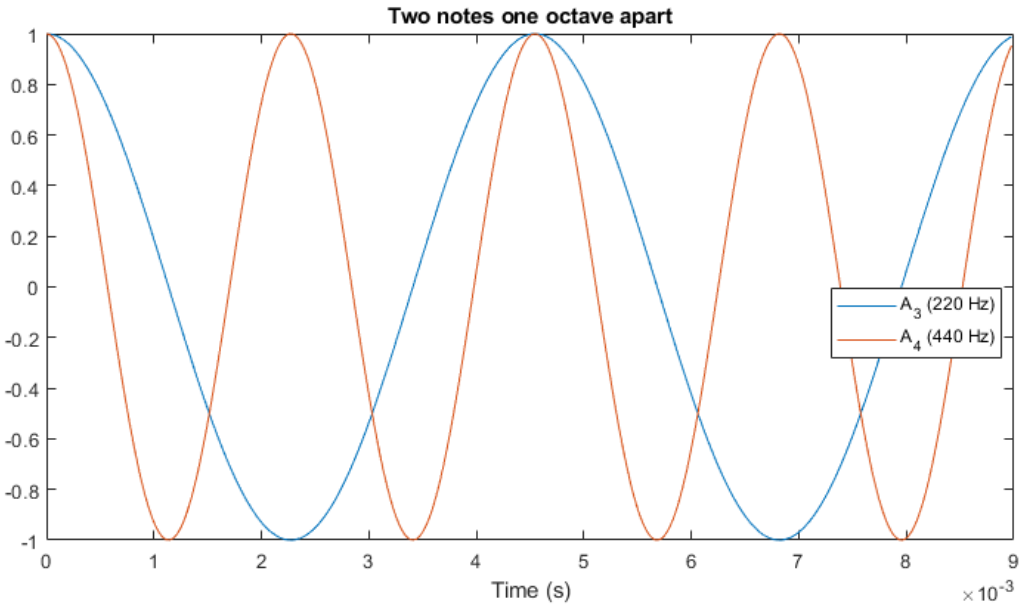

Figure 2. Frequencies of two notes an octave apart

Because most instruments are capable of playing notes from multiple octaves, a subscript is often placed below the letter to denote which octave that note belongs to. For example, the lowest note on the standard 88-note piano is A_0, and the highest note is C_8. The starting point for an octave is arbitrary, but most common tunings set the note A_4 to 440 Hz[6]. The frequencies of the other notes are determined relative to A_4 using 12-tone equal-temperament (12-TET) tuning, as the octave is broken up into 12 notes that are equally spaced around the log-2 scale and use 440 Hz and A_4 as the starting point[5].

The equation for 12-TET tuning can be used to derive the number of the key (N) on the standard keyboard, from 1 to 88, as seen in **Equation 1**[6]. A_4 is the 49th key on the keyboard, and the pitch and thus the key number go up when moving right across the keyboard[5]. The difference in pitch between two adjacent keys is called a semitone. Notes an octave apart are separated by 12 semitones. The black notes on the keyboard are referred to as sharps (♯) and flats (♭) and complete the 12 notes that comprise a 12-TET octave. **Table 1** depicts the frequencies and key number of the octave C_4 to C_5.

Note	**C_4**	**$C♯_4/Db_4$**	**D_4**	**$D♯_4/Eb_4$**	**E_4**	**F_4**	**$F♯_4/Gb_4$**	**G_4**	**$G♯_4/Ab_4$**	**A_4**	**$A♯_4/Bb_4$**	**B_4**	**C_5**
Number	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**
f_0 (Hz)	~262	~277	~294	~311	~330	~349	~370	~392	~415	440	~466	~494	~523

Table 1. Frequencies of octave beginning with Middle C (C_4)

$$f_0 = 440 * 2^{\frac{N-49}{12}}$$

$$N = 12 \log_2 \frac{f_0}{440} + 49$$

Equation 1. Converting frequency to musical key number

Other instruments have different ways of manipulating the pitch produced by the instrument to match the notes in 12-TET tuning. The standard guitar has six strings which are tuned to different pitches, and the fingers are placed upon certain points on the neck of the guitar, which are divisible by sections called frets. **Figure 3** depicts the frets on the guitar which correspond to white keys on a piano keyboard. Going up the frets on a single string on the guitar is equivalent to moving up the keyboard one semitone.

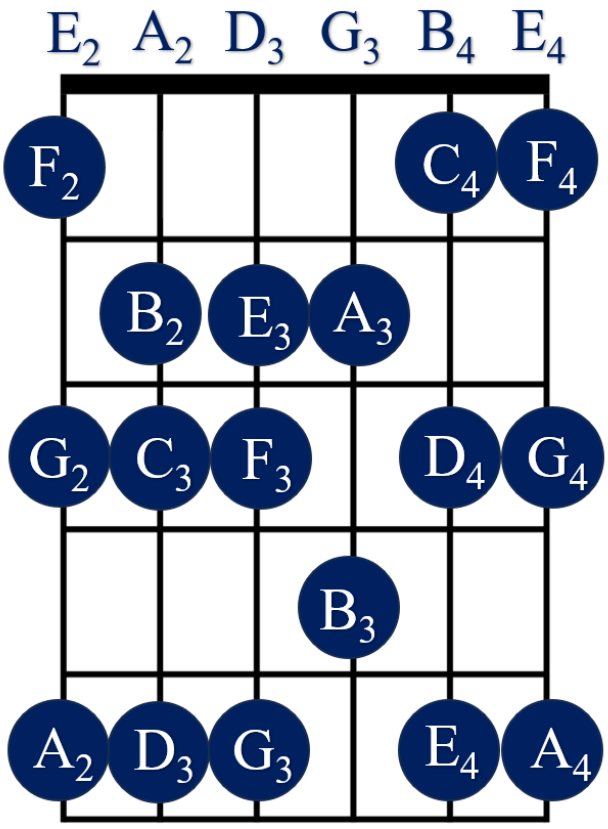

Figure 3. Notes on guitar fret board

The guitar, like other stringed instruments, exhibits a series of harmonics beyond the pitch of the vibrating string, known as the fundamental frequency f_0[6]. The combination of the fundamental

frequency with the series of harmonics contributes to the timbre, i.e., character of the note to distinguish it from a pure sinusoidal sound wave. The first harmonic is $2f_0$[6]. Other harmonics exist at $\frac{3}{2}f_0, \frac{5}{4}f_0, \frac{7}{4}f_0$ and all integer multiples of f_0[6]. The higher the harmonic, the less it contributes to the overall sound of the plucked string[6]. The human brain is adept at picking up f_0 out of the harmonic series[6]. The fundamental frequency of a note can be detected by a computer and computed in a multitude of ways, each with its own tradeoffs[7].

Electric Guitar

The electric guitar uses electromagnetic induction to produce a pitch, which is sent to an amplifier and speaker to produce a tone[8]. Fundamentally, the tone is created by the interaction between the metallic strings and a series of pickup coils, which are comprised of a cylindrical magnetic core surrounded by copper wire[8]. When a voltage is supplied to the pickup, an electric field is created[8]. The pickups are placed beneath the strings on the body of the guitar[8]. When a string vibrates, it disturbs the pickup's electric field at the frequency of the vibration of the string, which is detected by magnetic sensors also on the guitar body[8]. The frequency of this disturbance is then fed into an amplifier and speaker to create the sound of the note[8].

Musical Instrument Digital Interface (MIDI)

Musical Instrument Digital Interface (MIDI) is a technology used to transmit and store musical notes in a digital format that can be used by a variety of synthesizers and other digital instruments[2]. A MIDI message is not an audio file, but rather a sequence of data bytes that encode information about the desired musical notes[2]. MIDI is a universal technology standard that can be used by a variety of hardware and software from virtually every brand or manufacturer[2]. MIDI is flexible and is not beholden to the sound of a single instrument[2]. Using digital synthesizers, the timbre of any instrument can be mimicked and modified and even changed on the fly[2].

The MIDI standard is comprised of three parts[2]. Firstly, MIDI messages are the software protocol that is used to encode music in a digital format[2]. MIDI messages are comprised of a sequence

of bytes that contain information about which note is played, when and how long the note is played, and the velocity at which the note is struck[2]. The velocity of the note can be interpreted by the synthesizers as the volume, brightness, or length of the note, depending on the instrument the synthesizer is programmed to replicate[2]. The units and values of the parameters of a MIDI message are found in **Table 2**.

Parameter	Value range	Description
Note	1-127 (C_{-1}-G_9) [1]	Musical note
Velocity	1-127	How hard note was played
Duration	Any value in seconds	Length of note
Timestamp	Any value in seconds	Time when note begins

Table 2. Midi message parameters

Secondly, the MIDI standard provides a universal physical transport interface that allows a variety of MIDI-equipped devices to communicate with one another[2]. MIDI messages can be streamed between two devices in real time without any perceptible delay[2]. MIDI devices include digital keyboards, electric drum pads, or digital audio workstations (DAWs) that run on most standard operating systems, include PC, Mac, and Linux. DAWs can be used to record MIDI messages and the resultant synthesized audio can be added to the mix of a multi-track digital audio recording.

Thirdly, the MIDI standard provides a file format for the storage of MIDI messages that can be saved[2]. These files can include multiple tracks for different instruments and contain information for the tempo or speed of the song[2]. These files can be opened in any DAW and some physical instruments such as keyboards and drum pads and the notes and tempo can be further manipulated as desired by the musician. Some music software can be used to create sheet music that can be converted to or from a standard MIDI file.

Digital Audio Sampling

Analog audio is recorded into a digital format through digital sampling of the analog signal at a sampling rate f_s[8]. Nyquist theorem states that f_s of a periodic signal must be twice as high as the highest

[1] MIDI represents A_4 as note 69 instead of 49

frequency present in the analog source without introducing aliasing into the digital sample[8]. Because the human ear can detect frequencies up to around 20 kHz, digital audio is usually sampled at around 44.1-48 kHz[8]. An analog-to-digital convertor (ADC) for digital audio encodes the amplitude of a sound wave as a binary number, usually with a 16- or 24-bit depth[8]. Because a higher bit depth is capable of encoding more quantized levels in one sample, recording at a higher bit depth produces a more accurate digital recording that is capable of higher signal-to-noise ratio[8].

Digital audio can be stored in different file formats with different values for f_s and bit depth. The WAV file format is the most commonly used uncompressed audio file format[8]. During digital audio recording and playback, the binary values of the audio source are written and read into the buffer memory of the soundcard of the computer at a constant rate[8]. Direct boxes (DI) are external set top boxes that can be connected to an analog sound source (such as a microphone or electric guitar) and contain an ADC that streams the sampled digital audio to the soundcard's memory buffer through USB or other I/O interfaces[8].

Pitch Detection with CREPE Neural Network

There are a multitude of techniques to determine the fundamental frequency of a note, each with its own tradeoffs[7]. Many algorithms and models exist to process a digital audio file and determine its f_0[7]. Kim, et. al propose a deep convolution neural network operating in the time domain to more accurately determine f_0[7]. Kim, et. al refer to their neural network as a Convolutional Representation for Pitch Estimation (CREPE)[7]. CREPE takes a 1024-sample slice of a time-domain audio signal with a 16 kHz sampling rate[7]. The data is run through six convolutional layers to produce a 360-dimensional output vector comprised of f_0 with respect to time[7].

CREPE was trained with two data sets[7]. The first data set was a simple synth-based set of audio recordings with very little variation in timbre[7]. Next, CREPE was trained with a set of real-world recording stems from a multitude of different sources[7]. The end result of training led to a pitch detection accuracy of over 90%[7]. CREPE was able to operate effectively for samples with a signal-to-noise ration below 10 dB[7].

METHODS

The audio from an electric 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 one wishes to record a MIDI message with a DAW.

After the MIDI outputs are initialized, the MATLAB audio recorder is configured. A sampling frequency F_s of 44.1 kbits/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.

Create MIDI Algorithm

The following algorithm was used to convert the WAV file into a MIDI signal. First, the fundamental frequencies 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 indices.

```
while 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Record sound from Windows default audio input                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp("start recoding")
    recordblocking(recObj, recDuration);
    disp("end stop recording")
    sigsound = getaudiodata(recObj);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find MIDI message from audio recordin                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    midi_messages = create_midi(sigsound, Fs)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Send MIDI messages to selected MIDI output                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp("playing midi")
    midisend(device, midi_messages);
    pause(recDuration+1)
end
```

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 F_s. 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 $f_0[n]$
Find notes corresponding to $f_0[n]$: note = $12*\log_2(f_0/440) + 49$
Find sound pressure levels $L_p[n]$
Find values of n where notes change (change_indices)
Find note_peaks, note_lengths, and midi_notes

```
for (i = 1; i++; i < length(change_indices))
    if i == 1
        midi_notes(i) = notes(change_indices(i));
        note_peaks(i) = maximum peak of Lp(i:change_indices(i));
        note_lengths(i) = change_indices(i);
    else if i == length(change_indices)
       note_peaks(i) = = maximum peak of Lp(change_indices(i):change_indices(i-1));
       midi_notes(i) = notes(change_indices(i));
       note_lengths(i) = change_indices(i) - change_indices(i-1);
    else
       midi_notes(i) = notes(change_indices(i));
       note_lengths(i) = change_indices(i) - change_indices(i-1);
       peak = maximum peak of Lp(change_indices(i):change_indices(i-1));
     if(peak is empty); peak = 0; end
       note_peaks(i) = peak;
    end
end
Time t = 0:1/Fs:length(notes)/Fs
Generate MIDI signal
for(i = 1; i++; i < length(change_indices)
  if(midi_notes(i) == 0)
       volume(i) = 0
  else
       volume(i) = floor of 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_indices(i - 1) + 1);
       }
end
```

Algorithm 1. WAV-to-MIDI conversion

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**.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Determine fundamental frequencies of recording                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f0 = pitchnn(recording, Fs);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Adjust frequency array to be same length as recording                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n1 = numel(f0);
n2 = length(recording);
f02 = interp1(1:n1, f0, linspace(1, n1, n2), 'nearest');
```

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 have exactly the right pitch as the desired note. Therefore, the note calculated by **Equation 1** must be rounded to the nearest integer. A MIDI message must have a note number that is an integer between 0-127. 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.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find musical notes from frequencies                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
notes = freq_to_note(f02);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Detect indeces where musical notes change                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
notes(isnan(notes))=0;
```

Code Snippet 3. Determining musical note from frequency

Determining note length and timing

The note length and timing are determined by finding which indices in the recording array correspond to a change in the note (change_indices), and using the difference between concurrent values

of change_indices along with F_s 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 F_s 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.

```
% detect change in notes
pitch_change = diff(notes) ~= 0;
% find indices where note changes
change_indices = find(pitch_change);
change_indices = [change_indices length(notes)];
…
% determine the note length for each note
note_lengths(i) = change_indices(i) - change_indices(i-1);
…
% create time array
t = 0:1/Fs:length(notes)/Fs;
…
% determine note length and note timestamp
note_length = note_lengths(i);
timestamp = t(change_indices(i-1)+1);
```

Code Snippet 4. Determining note length and timestamp

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 and 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 value until after the note is initially strummed, the peak value of the pressure was found during the whole 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.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find volumes of recording                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SPL = splMeter("SampleRate", Fs);
pressure_levels = SPL(recording);
…
peak = max(findpeaks(pressure_levels(change_indices(i-1):change_indices(i))));
if(isempty(peak)); peak = 0; end
note_peaks(i) = peak;
…
volume(i) = find_volume(note_peaks(i));
…
function volume = find_volume(note_peak)
    volume = floor(127*note_peak/100);
end
```

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.

Creating MIDI Messages

MIDI messages can be created using the MATLAB Audio Toolbox plugin. **Code Snippet 6** demonstrates how a MIDI message was created in MATLAB. The comment in the code shows the format of a MIDI message.

Hardware and Software Setup

Audio was recorded by plugging a Fender Telecaster electric guitar into the ¼ inch line input of an M-Audio M-Track Solo USB recording interface. The M-Track solo was connected through USB to a

personal computer. Audio processing was performed by running the MATLAB script create_midi.m. A new MIDI track was created and set to record in the DAW, and a new audio recording was started.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Create MIDI messages                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
note = midi_notes(i);
if note == 0
      volume(i) = 0;
else
      volume(i) = find_volume(note_peaks(i));
end
note_length = note_lengths(i);
timestamp = t(change_indices(i-1)+1);
midi_m(i, :) = midimsg("Note", 1, note, volume(i), note_length/fs, timestamp);
% NoteOn  Channel: 1  Note: 46  Velocity: 0   Timestamp: 1.89073
% NoteOn  Channel: 1  Note: 47  Velocity: 73  Timestamp: 1.89073
% NoteOn  Channel: 1  Note: 47  Velocity: 0   Timestamp: 2.21955
end
…
midisend(device, midi_messages);
```

Code Snippet 6. Creating the MIDI messages

A Guitar short guitar riff was recorded in MATLAB and run through the 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 the MIDI track in the DAW, individual MIDI notes could be dragged and dropped to change pitch or timing.

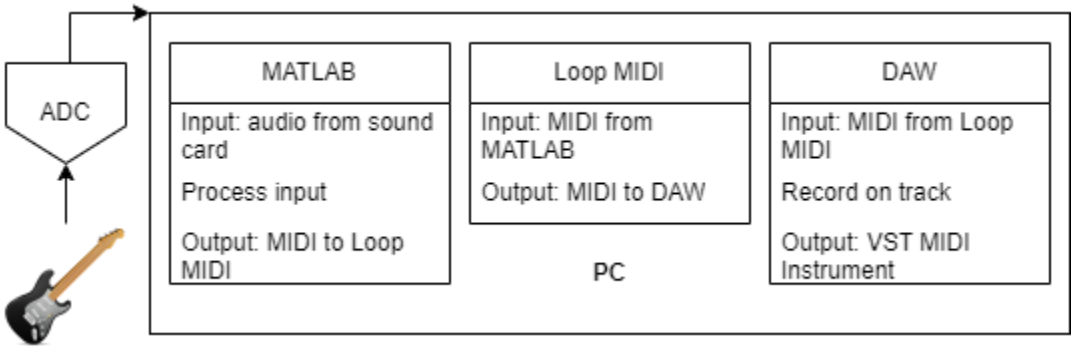

Figure 4. Audio hardware and software setup

RESULTS

The melody of the recorded instrument was successfully converted into a MIDI message, played back on a MIDI instrument, and recorded in real time on an audio track in a DAW. **Figure 5** shows a log-based representation of the recorded WAV file of a 15-second guitar riff. The y-axis of the graph has been normalized to a value between -1 and 1. Using the CREPE neural network, the fundamental frequency of the same guitar riff was measured with respect to time, as shown in **Figure 5**. By using the find_note function, the change in the musical note was determined with respect to time, as shown in **Figure 5**. The MATLAB script also successfully determined the sound pressure of the same guitar riff, as shown in **Figure 5.** The MIDI message generated by MATLAB was successfully read and recorded by the DAW through Loop MIDI, as shown in **Figure 6**.

CONCLUSION

The MATLAB script successfully recorded the audio of a guitar riff and played its melody back with a MIDI instrument. The change in pitch of the recording was successfully detected and converted into a musical note number and, using the sound pressure, a data stream of the MIDI messages was created using the create midi algorithm and recorded in the DAW. Although there was some roughness to the recreation around rapidly changing notes, the resultant melody was very accurate to the original audio. Some of the roughness could come from the fact that two strings on the guitar could have been unintentionally vibrating at the same time, and the CREPE network can only determine a single frequency at once. Additionally, it is difficult to have the guitar be perfectly in tune with the desired 12-TET

frequency, and the create midi algorithm only detected the nearest note on the piano. MIDI only allows the creation of notes that are integers, so any bending of the pitch by moving the string perpendicular to the neck of the guitar could not be recreated.

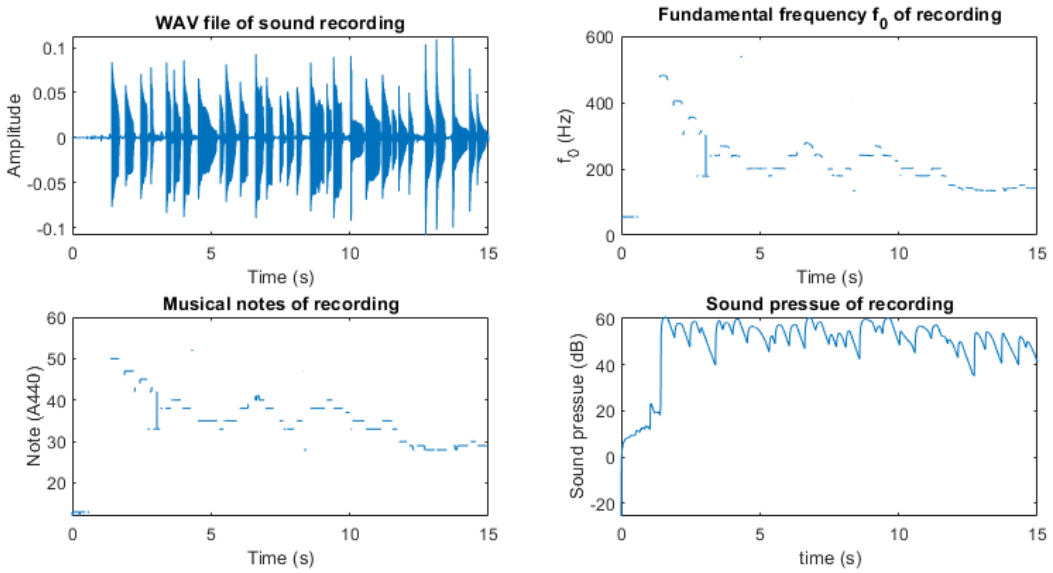

Figure 5. (a) WAV file. (b) fundamental frequencies. (c) musical notes. (d) sound pressure

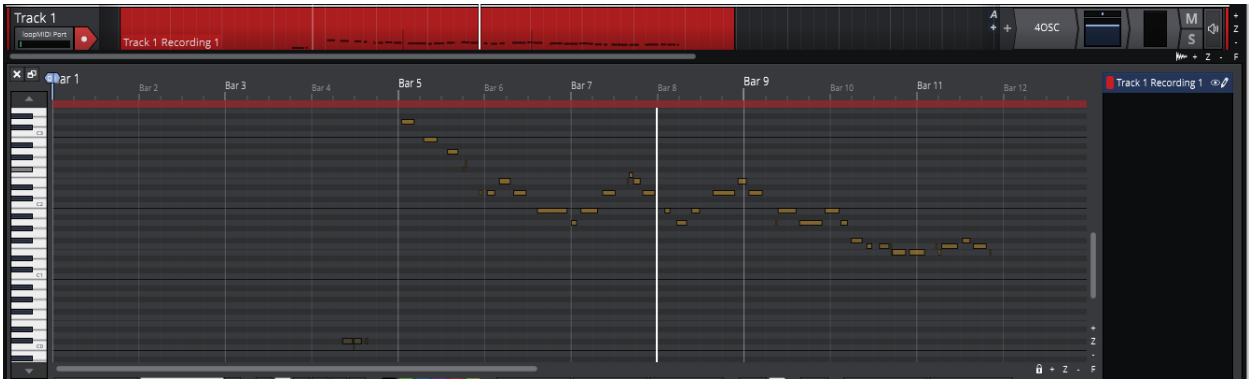

Figure 6. Screenshot of recorded MIDI message in DAW

The algorithm could be improved in a few ways. Firstly, a smoothing filter could be applied to the detected notes to further filter out the frequency jumps on the transition between strings. Secondly, MIDI is capable of producing pitches between notes on the piano through the PitchBend command, so the bending of the pitch on a guitar could be successfully be included as part of the MIDI message. Further investigation could be done into how to detect such bends in pitch. Additionally, another neural network

could be introduced to detect multiple pitches at a time, which would allow for the detection of harmonies and chords produced by the recorded instrument. Additionally, other, faster methods of detecting the fundamental frequency could be investigated to see if the MIDI message could be created closer to real time.

Appendix

main.m

```
%
% This program records ten seconds of audio, creates a MIDI message from
% the recording, and plays back the MIDI message over the Windows MIDI
% Mapper device.
% After the MIDI message is played back, another ten seconds are recorded
% and played back until the script is stopped.
%
clear;
close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Detect MIDI devices and set MIDI output                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
availableDevices = mididevinfo;
device = mididevice(availableDevices.output(2).ID); % MIDI mapper
% device = mididevice(availableDevices.output(4).ID); % loop MIDI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Set up audio recording parameters                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
recDuration = 10;
Fs = 44100;
nBits = 16;
nChannels = 1;
recObj = audiorecorder(Fs, nBits, nChannels);
while 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Record sound from Windows default audio input                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp("start recoding")
    recordblocking(recObj, recDuration);
    disp("end stop recording")
    sigsound = getaudiodata(recObj);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find MIDI message from audio recordin                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    midi_messages = create_midi(sigsound, Fs)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Send MIDI messages to selected MIDI output                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    disp("playing midi")
```

```
    midisend(device, midi_messages);
    pause(recDuration+1)
end
```

---

create_midi.m

```
function midi_m = create_midi(recording, Fs)
% CREATE_MIDI  Creates MIDI messages from audio recording.  <---- H1 line
%   M = CREATE_MIDI(RECORDING, FS) creates MIDI messages from RECORDING for
%   sampling frequency FS
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Determine fundamental frequencies of recording                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    f0 = pitchnn(recording, Fs);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Adjust frequency array to be same length as recording                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    n1 = numel(f0);
    n2 = length(recording);
    f02 = interp1(1:n1, f0, linspace(1, n1, n2), 'nearest');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find musical notes from frequencies                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    notes = freq_to_note(f02);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Detect indices where musical notes change                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    notes(isnan(notes))=0;
    pitch_change = diff(notes) ~= 0;
    change_indices = find(pitch_change);
    change_indices = [change_indices length(notes)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find volumes of recording                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    SPL = splMeter("SampleRate", Fs);
    pressure_levels = SPL(recording);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Find peak volume, note length, and musical notes                        %
%   in between note changes                                                %
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:length(change_indices)
        if i == 1
            note_peaks(i) = max(pressure_levels(i:change_indices(i)));
            note_lengths(i) = change_indices(i);
        elseif i == length(change_indices)
            note_peaks(i) = pressure_levels(i);
            midi_notes(i) = notes(change_indices(i));
            note_lengths(i) = change_indices(i) - change_indices(i-1);
        else
            midi_notes(i) = notes(change_indices(i));
            note_lengths(i) = change_indices(i) - change_indices(i-1);
            peak = max(findpeaks(pressure_levels(change_indices(i-
1):change_indices(i))));
            if(isempty(peak)); peak = 0; end
            note_peaks(i) = peak;
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Remove negative sound pressures                                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    note_peaks(note_peaks<0) = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Create time array for timestamps                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    t = 0:1/Fs:length(notes)/Fs;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Create MIDI messages                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    note = midi_notes(1);
    if note == 0
        volume(1) = 0;
    else
        volume(1) = find_volume(note_peaks(1));
    end

    midi_m(i, :) = midimsg("Note", 1, note, volume, note_lengths(1), t(1));

    for i = 2: length(midi_notes)
        note = midi_notes(i);
        if note == 0
            volume(i) = 0;
        else
```

```
            volume(i) = find_volume(note_peaks(i));
        end
        note_length = note_lengths(i);
        timestamp = t(change_indices(i-1)+1);
        midi_m(i, :) = midimsg("Note", 1, note, volume(i), note_length/Fs,
timestamp);
    end
end
```

---

find_volume.m

```
function volume = find_volume(note_peak)
% FIND_VOLUME  Finds MIDI note volume from value of NOTE_PEAK.  <---- H1 line
%   V = FIND_VOLUME(NOTE_PEAK) finds corresponding MIDI note peak from
%   value of NOTE_PEAK
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Determine MIDI note velocity (volume) from peak sound pressure  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    volume = floor(127*note_peak/100);
end
```

---

freq_to_note.m

```
function note = freq_to_note(f)
% FREQ_TO_NOTE  Finds nearest musical note to frequency F.  <---- H1 line
%   N = FREQ_TO_NOTEE(F) finds nearest musical note from frequency F using
%   A440 12-tone equal temperament (12-TET) tuning
%   value of NOTE_PEAK
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Determine nearest 12-TET musical note %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    note = floor(12*log2(f./440) + 69);
end
```

References

[1] Hughes, Russ. (2023, June 28). *Home studio setup costs compared - 1980s and now*. Production Expert. https://www.production-expert.com/production-expert-1/2020/9/8/home-studio-recording-costs-compared-1980s-and-now

[2] *About MIDI - Part 1: Overview*. The MIDI Association. (n.d.). https://www.midi.org/midi-articles/about-midi-part-1-overview

[3] Goodine, Rebecca. (2022, August 25). *How SoundCloud changed music*. Recording Arts Canada. https://recordingarts.com/how-soundcloud-changed-music/

[4] McClellan, James H. (n.d.). *DSP first*. DSP First. Retrieved from: https://dspfirst.gatech.edu/

[5] Justin Guitar (n.d.) *Learn how to play guitar with JustinGuitar.com*. JustinGuitar.com. https://www.justinguitar.com/

[6] Sulzer, David. (2021). *Music, math, and mind: the physics and neuroscience of music*. Columbia University Press.

[7] Kim, Jong & Salamon, Justin & Li, Peter & Bello, Juan. (2018). *CREPE: A convolutional representation for pitch estimation. acoustics, speech, and signal processing*. Retrieved from: https://arxiv.org/abs/1802.06182

[8] Bartlett, Bruce., and Bartlett, Jenny. *Practical recording techniques: the step-by-step approach to professional audio recording*. 7th ed. 2016. Web.

[9] Purdue University Science Express. (2018). *Physics of the electric guitar: connections in electricity and magnetism*. Purdue University Science Express. Retrieved from: https://www.purdue.edu/science/science-express/labs/labs/Physics%20of%20the%20Guitar.pdf