ENGIN 100: Music Signal Processing

PROJECT #3

Music Synthesizer and Transcriber

Professor Andrew E. Yagle

Dept. of EECS, The University of Michigan, Ann Arbor, MI 48109-2122

Revised July 2016

I. ABSTRACT

This is the main project for the course. It has two parts: (1) Programming a simple music synthesizer entirely in Matlab using a Matlab musical GUI similar to the one you developed in Project #1, using snippets of real instruments; (2) Programming and evaluating a simple music transcriber entirely in Matlab using the Matlab stem-based staff-like notation in Project #1, but now including note duration information.

II. BACKGROUND

Music synthesizers can be as simple as recording the sounds of actual musical instruments and playing them back at variable speeds to obtain the desired pitches (wavetable synthesis), filtering of simple signals like square waves or sawtooth waves to achieve a desired effect (subtractive synthesis), or as complicated as actually computing each harmonic of a note separately and adding them together (additive synthesis). See "Music Synthesis" on Canvas.

Musical transcription, which generates sheet music (or the equivalent) directly from a musical recording, is much more difficult. It is by no means a completely solved problem, and present techniques involve signal processing concepts such as hidden Markov models that are graduate-level material, since real music is more complicated than the periodic signals whose periods change abruptly that we have considered in this course.

Nonetheless, transcription of simple music can be accomplished using the techniques presented in this course. You saw in Lab #4 that the spectrogram of music could function as a type of musical notation. Other approaches should be tried as well, such as the fft analysis at specific frequencies used in Project #2.

The results of this project will be two .m files. One implements a music synthesizer which accepts either on-screen keys clicked with a mouse or a sequence of letters representing musical notes and outputs a melody using any of several musical instruments (guitars, clarinets, and others) selected from an on-screen menu. Of course, these can be added together ("mixed") to achieve the effect of several instruments played together.

The other .m file implements a music transcriber which accepts a .wav or .mat file from your synthesizer

and generates a musical-staff-like transcription using Matlab's stem command. Details are provided below. Some suggested approaches for the transcriber will be discussed in lecture (see `proj3lecture.pdf`).

III. PROJECT #3: WHAT YOU HAVE TO DO

Download the file `proj3.wav` from the course web site. This file contains snippets of length 32768 samples each from an electric guitar, a clarinet, a trumpet, and a single tone, all sampled at $44100\frac{\text{SAMPLE}}{\text{SECOND}}$. Each instrument is playing the 13 notes of an octave (including low and high notes G) in succession.

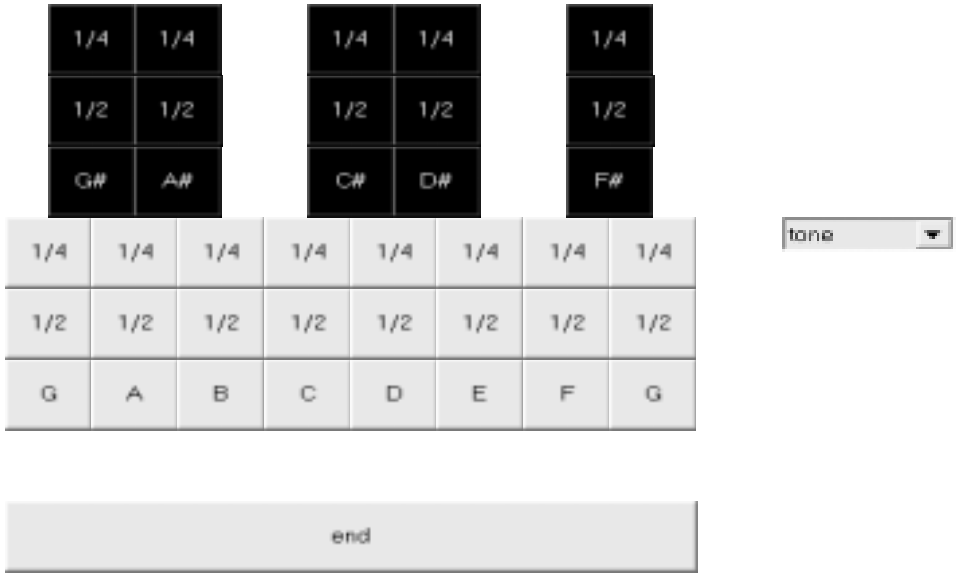
Design your own instrument using additive synthesis (adding together harmonics with amplitudes selected by you) or multirate processing and the Circle of Fifths (see "Music Synthesis") and label it with your team name. Also create a marching band sound by reverbing each trumpet note. Do this by adding copies of trumpet to itself, with each copy delayed slightly from the previous copy.

Do a noise analysis performance like the one you did in Project #2: error rate vs. SNR. Discuss results.

A. Synthesizer Specifications

Your synthesizer should have a keyboard, like the one shown below, and note durations (in #samples)

NOTE	1 second	WHOLE	HALF NOTES	QUARTER
LENGTH	44100	32668+100	16284+100	8092+100



- The final 100 samples of each note should be zeros, to indicate when a note stops.
- The pull-down menu offers a choice of instruments, prior to use of the keyboard. Use:
- `H=uicontrol('Style','Popup','Position',[500 250 100 50],'String','guitar|clarinet|trumpet|own');`
- `pause;I=get(H,'Value');` This generates I=1,2,3,4 depending on the selection made.

B. Transcriber Specifications

- The transcriber should output a musical scale and notes like that of Project #1, except:
- The separation between transcribed notes should indicate the length of each note:

- A whole note followed by 3 spaces; a half note by 1 space; a quarter note by none.
- The transcriber need not be able to handle the electric guitar. This requires a bass scale.
- Perform a **noise analysis** (error rate vs. SNR plot) as you did in Project #2.

C. Presentation

Your results will be presented in both written form (a technical report) and orally to the class and instructors, who will function here as your co-workers and bosses. You will be graded on the presentation as well as on the results. This is very realistic–the greatest invention since sliced bread is worthless unless you can explain to bosses and customers what it does and why it is valuable.

Each team should email its presentation to its TC instructor and its .m files to its lab instructor. Each team should bring 2 copies of your report to the final exam.

IV. Synthesizer Hints

A. Getting Started

Download *proj3.wav* from Canvas. Then do this:

- `>>[X Fs]=audioread('proj3.wav');XX=reshape(X,32768,13,4);`
- Stores each of 13 notes of duration 32768, played by each of 4 instruments in separate columns of `XX`.
- For example, `>>soundsc(XX(:,1,3),Fs)` is trumpet (instrument #3) playing low G (first of 13 notes).
- Your synthesizer can access any instrument playing any note easily from `XX`. Don't forget to add 100 zeros.

B. Own Instrument

To create your team's own instrument (labelled by your team name, e.g., the "Yaglephone"):

- Use additive synthesis (see p. 37 of course notes),
- Your report should specify the harmonic amplitudes $\{c_k\}$.
- Use multirate signal processing (see "Music Synthesis")
- The criterion here is that your team has to agree that its instrument sounds "cool."
- You might find a snippet online, and analyze its spectrum, e.g., oboe, piano, 1980s video games.
- To get the "marching band sound," reverb the trumpet (see p. 37 of course notes).
- Your report should specify the #copies and time delay D between them.
- Your synthesizer does not need to be able to switch instruments in the middle of a run.

V. Transcriber Hints

- You need NOT transcribe the bass guitar, which would require a separate bass scale.
- The bass guitar is just for fun. Try playing "Ironman" or "Smoke on the Water." (I'm showing my age).
- Remember you know you are looking for one of 13 possible musical notes.
- The pure tones are in proj3.wav so you know exactly what frequencies you are looking for.
- There is no right answer for the best transcription method. Different teams use different methods.

- This is for calibration of your transcriber. The frequencies are not the same as those in Lab #2.
- You do not need to be able to transcribe the pure tones (harmonic product spectrum won't work on them).
- Transcriber approaches are in Chapter 5 of the course notes.
- Discuss how you handled the octave problem (distinguishing high G from low G).

– Fundamental frequency identification: Use $f = (K-1)S/N$, but remember that N=length in #samples of the note could be 32768,16384 or 8192. You can use the harmonic amplitudes to determine what instrument is being played. But you also have to handle the octave problem for the trumpet.

– Harmonic product spectrum (HPS): You need to multiply 4 or 5 sets of harmonics, not 3. You need NOT be able to transcribe pure tones, for which HPS won't work. NOT a good excuse not to use HPS.

– Autocorrelation: When noise is added, the peak at index $n = 0$ (MATLAB index=1) will be much larger than the peaks at n=multiples kN of the period N. See the figure on p. 36.

A. Note Durations

- Let your synthesizer output be stored in *output.wav*. Then do this:

– `>>[X Fs]=audioread('output.wav');XX=reshape(X,8192,length(X)/8192);`

– `F` is the vector of length `length(X)/8192` of stem heights to be plotted.

– `>>N=sum(abs(XX(8093:8192,:)));NN=find(N<0.001);stem(NN,F)` plots the stems

– With gaps of appropriate durations *before* each stem.

– Example: `X` consists of a whole note, then a half note, then a quarter note.

– with durations 32768, 16384, 8192.

– Then: XX=reshape(X,8192,7);N=sum(abs(XX(8093:8192,:))); will look like this: `[* * * 0 * 0 0]`;

– and `NN=find(N<0.001);` will look like this: `[4 6 7]`.

VI. Grading of Technical Content

Fulfilling the above specs is worth 50/100.

Some suggestions of other things to do (think of these as extra credit):

- Noise analysis: Peform a noise analysis for:

– Several transcriber approaches, not just the one you used;

– Several different instruments.

- Synthesizer:

– Additional keys: "Delete" (the just-played note); "echo" (list the notes played).

– use the keyboard to play notes, using `input` so you can play chords.

– Slide switches for reverb: #trumpets and delay between them.

– Apply an ASDR (Attack, Sustain, Decay, Release) envelope to each note played.

– Plot the spectrum of your own instrument playing a specific note.

– Button symbols for various operations.

– Teams in the past have included help screens and pop-op manuals (!).

- Transcriber:
  - Instead of using spacing to depict note lengths (whole, half, quarter), use an unfilled stem with no staff for whole notes, a filled stem with no staff for half notes, and a filled stem with staff for quarter notes. This looks much more like actual music notation. How? Try >>`help stem` You can be very ingenious here!
  - Beyond just identifying each note played, have it identify the instrument playing it. Do this by matching the relative amplitudes of harmonics to the relative amplitudes of harmonics for each possible instrument. Maybe just use the first two harmonics.
  - Identify sharps and flats on transcriber output: >>`text(X,Y,'#')` places a # at coordinates (X,Y).
  - Instead of using `stem`, place actual musical staff symbols to identify notes. Consider `Maestro`.