AIST2010 Project Report

Name: Hsu Tung Yu

SID: 1155109831

Title: Chord Identifier and Visualizer on Piano

Abstract

From jazz music to classical music, chords are always one of the building blocks of harmony in different genres of music. in this project, I have tried to implement a chord identifier and a graphical user interface as a piano-style visualizer. Users can either play a chord by using a MIDI keyboard or input an audio file containing the sound of a chord being played, and the results would be shown in real time with the visualizer.

Background

In this project, I have used multiple topics learnt in course lectures. This includes the Python package *librosa* for loading the audio files, musical notes handling, frequency extraction and onset extraction; and the software *SuperCollider* for synthesizing chord sounds. While building the chord identifier, the graphs taught in lectures, such as chromagram, as well as the concept of harmonics are used.

This project is inspired by my experience of me studying for a music theory examination. One part of the examination is to recognize chords from an excerpt of a score. Although this is not particularly difficult, I thought at the time that the process of identifying a chord is very algorithmic: find the unique chord notes, align them in ascending order and finally identify the chord based on the ordered chord notes. Therefore, I chose to implement a chord identifier alongside a visualizer in piano style, since I was taught to use piano to identify the chord.

One existing project that is similar to mine is *PyChord* [1]. *PyChord* is a Python package that has multiple features about handling chords. For example, *PyChord* can also find chords using note names, which is the approach that I have chosen when implementing the chord identifying algorithm on my own;

it could also do chord transposition and create chord progressions. Another similar existing project is *Music21* [2]. *Music21* is also a Python package that can find chords using note names but is a more general Python package dedicated to different aspects of music.

Methodologies

a. Programming Language

Python is used to implement the whole project. This is because Python is taught during the course, and that there are many useful packages that can be used to achieve the intended results. In particular, I have used the Python packages *librosa, numpy* and *matplotlib* which are taught in class. I have also used the Python packages not covered in class, including *PyQt5*, a package for GUI development in Python, *pygame*, a package for MIDI handling and *sounddevice*, a package for playing the audio file.

For generating synthetic chord sound using pure sine waves, the software *SuperCollider* is also used.

b. Note Names Representations

In music, there are twelve notes in an octave, namely C, C♯/D♭, up to B. For simplicity, when note names are written out explicitly in this project, the enharmonic equivalent notes, such as C♯/D♭, are all represented using the sharp accidentals (♯).

Notice that the representation above has no information on the octave of the notes. Therefore, whenever octaves are needed, MIDI numbers of the notes are used. In MIDI number representation, the same note in two consecutive octaves are differed by 12.

c. Chord Identifying Algorithm

As the title of the project, one of the most important features to implement is the chord identifying algorithm. To implement this, I considered how music theory describes chords.

In music theory, a chord has three important characteristics: root note, quality and inversion. It is also known that the voicing of the chord does not matter, i.e., only the unique notes appeared in the chord matter. A chord can be identified by its full name given these three characteristics.

To find the quality and the inversion of a chord, the distances between each consecutive unique notes of the chord can be used. Then, after obtaining the inversion of the chord, the root note can also be found.

A Python dictionary variable *QUALITY* alongside with a Python list containing shortened text representations of the qualities of chords and the position of the root note is created by me manually finding and encoding the distances for each type of chords. For example, to find the distances for major chords in root position, I have considered a C major chord, which has the notes C, E and G. If C is the “middle C”, then the MIDI numbers for the three notes are 60, 64 and 67, which gives distances 4 (64-60 = 4) and 3 (67-64 = 3). Also, the root note in this case is the first note. Therefore, an entry representing major chords in root position in *QUALITY* is *(4, 3): [“a”, 0]*, with a 0 since Python is zero-based.

What is left for the chord identifying algorithm is to find the unique notes for any given voicing of chords played. To do this, first all the notes being played are stored in a Python list *noteMIDI*. Then, the lowest note played is used as a pivot. Next, for each of the other notes, repeatedly subtract 12 from its MIDI number until all notes are between one octave of the pivot note. Distinct notes can then be found and searched in *QUALITY* for the quality and inversion. The root note can then be found by *noteMIDI[i]*, where *i* is the integer given in the value found in the dictionary.

Finally, the MIDI number of the root note is converted back to note name, and the shortened name of the chord played is returned by concatenating the note name with the string given in the value found in the dictionary.

d. <u>Chord Visualizer in Piano Style</u>

Another important feature to implement is the chord visualizer. To implement this, I used the Python package *PyQt5*. The visualizer is inspired by a software named *Synthesia* [3], which can show the notes in a MIDI file in a dropdown manner.

The GUI of the visualizer is shown in figure 1.

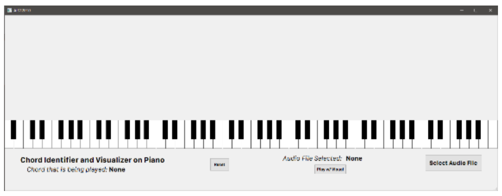
Chord Identifier and Visualizer on Piano
Chord that is being played: None
Reset
Audio File Selected: None
Play w/ Visual
Select Audio File

Figure 1: GUI of the visualizer

In my implementation, I originally aimed to mimic the style of *Synthesia*, i.e., to show any notes being played either on a MIDI keyboard or an audio file in real-time, and the visualizations would disappear once they reach the piano. However, this complete implementation turns out to be too aggressive, as I have just started to learn *PyQt5* for this project and hence was not proficient enough to use this package to implement it.

One of the difficulties is that the visualizer relies on the MIDI module of *pygame* and the trigger mechanisms of *PyQt5*, which makes real-time visualization of any notes, including duplicate ones, very difficult. The MIDI module of *pygame* is responsible for reading MIDI signal from a MIDI keyboard, which includes a velocity value. This restricted how the visualizer could determine the start and the end of any notes being played, since only a velocity value of 0 represents the keys being released.

In the final implementation, I can only implement the part that the visualizations would disappear when they reach the piano, while making a compromise such that all visualizations of a chord would start dropping down whenever one of the keys is released. This is a setback because in real piano playing, keys are often not being released together. However, for visualizing the voicing of the chord being played, I think this is still enough.

Also, in *PyQt5*, to make the GUI interactive, i.e., responding to any input that the users make, trigger must be used. The trigger mechanism works like this: when users make an input, a signal is transmitted to a corresponding trigger, in which the trigger would make changes to the GUI. However, this means that for each visualization created, a trigger must also be created and connected to the visualization, while being connected by a note on the piano. This makes creating two or more visualizations for the same note very difficult, since this would require creating many sets of new triggers and visualizations on-the-fly. Therefore, I also make a compromise in the final implementation that no repeated notes would be played until the visualization for the note has disappeared.

e. <u>Audio File Frequency Extractor</u>

To find the chords being played in an audio file, one possible way is to find all the frequencies of the notes in the audio file, and then to use the chord identifying algorithm described above. However, when a note is played on a musical instrument, harmonics would also be produced. It means that more than one frequency would be detected, with the lowest being the fundamental frequency and the higher ones being the harmonics.

Since the frequencies of the harmonics would be multiples of the fundamental frequency, I originally wanted to implement the frequencies extractor in a way that regards only the frequencies that are not multiples of any other frequencies as notes being played, while other frequencies could be understood as harmonics and hence be eliminated. Also, since chords are often sounded in downbeat, I also thought of using onset detection and detect chords whenever an onset begins.

However, it turns out that in reality, the fundamental frequency may be missing while the perceived pitch remains the same [4]. This makes my original idea infeasible on real life audio sample from instruments. Therefore, I also make a compromise here such that the implementation of the frequency extractor would work for chords generated by synthetic sine waves only.

To extract the frequencies, Fourier transform must be used on the audio sample. Luckily, *librosa* has a built-in function *librosa.piptrack* for this purpose of extracting frequencies [5]. The output of this function is two *numpy* arrays, storing the frequencies and the magnitudes at each bin respectively. To find the frequencies, and hence the pitches, appeared in the audio sample, it suffices to find the nonzero entries in the frequency array. Then the frequencies can be converted back to MIDI numbers and the chord identifying algorithm described above can be used.

Conclusion

In conclusion, I really learnt a lot from completing this course project: came up with a chord identifying algorithm, learnt a Python package *PyQt5* for GUI development in Python, discovered that the fundamental frequency could be missing and although in a very restricted manner, found a way to detect chords generated by pure sine waves.

Although I have encountered many bugs and unexpected behaviors while completing the code for the project, most of them are solved by finding references on Stack Overflow [6][7][8] as well as looking at the documentations of the Python packages used carefully, and I really learnt a lot in this trial-and-error process.

To be honest, the end product of the project is not perfect. However, I believe that the project could definitely be improved to achieve a more applicable product, for example to include chord recognitions for real audio samples. Nonetheless, I have really tried hard to finish the product as complete as possible and learnt a lot in this journey.

Reference

[1] *PyChord*. (2020). Yuma Mihira. Accessed: Dec. 20, 2020. [Online]. Available: https://github.com/yuma-m/pychord

[2] *music21.* (2020). Michael Cuthbert. Accessed: Dec. 20, 2020. [Online]. Available: https://github.com/cuthbertLab/music21

[3] *Synthesia.* (2020). Synthesia LLC. Accessed: Dec. 20, 2020. [Online]. Available: https://www.synthesiagame.com/

[4] J. Schnupp, E. Nelken, A. King. "Missing fundamentals. Periodicity and Pitch | Auditory Neuroscience." Auditory Neuroscience. http://auditoryneuroscience.com/pitch/missing-fundamentals (accessed Dec. 20, 2020).

[5] librosa development team. "librosa.piptrack — librosa 0.8.0 documentation." librosa. https://librosa.org/doc/main/generated/librosa.piptrack.html (accessed Dec. 20, 2020).

[6] M. Byers. "unique - Python: Uniqueness for list of lists - Stack Overflow." Stack Overflow. https://stackoverflow.com/a/3724558 (accessed Dec. 20, 2020).

[7] L3viathan. "python - Checking if list is a sublist - Stack Overflow." Stack Overflow. https://stackoverflow.com/a/35964184 (accessed Dec. 20, 2020).

[8] fviktor. "python - QPropertyAnimation doesn't work with a child widget - Stack Overflow." Stack Overflow. https://stackoverflow.com/a/6953965 (accessed Dec. 20, 2020).