EE 473 PROJECT

GUITAR CHORD RECOGNITION

Ahmet Can Ozbek

I. PROBLEM STATEMENT

The purpose of our project is to implement a program, that reads a guitar chord as an input, recognizes the notes that are present in the chord through required frequency analysis steps and finally gives the chord as an output. Thereby, it takes a '.wav' file as an input and print outs to corresponding chord.

II. INTRODUCTION

- Chords sounds are generated by 'Guitar Pro 5' software. Our inputs are '.wav' files.

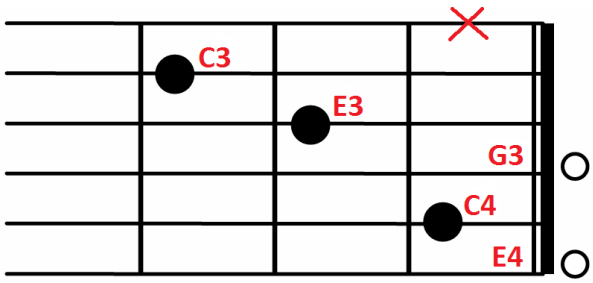
III. METHODOLOGY

The main logic behind our project is to use Fast Fourier Transform, thus detect the peaks in the spectrum. Then we apply comparisons between the peak frequency values of the spectrum and our database of chords. Finally, through this matching process, we are able to determine the corresponding chord.We have a set of 24 different chord sounds and we are able recognize them.

For this project we are just aiming to recognize chords which consist of just 3 notes.

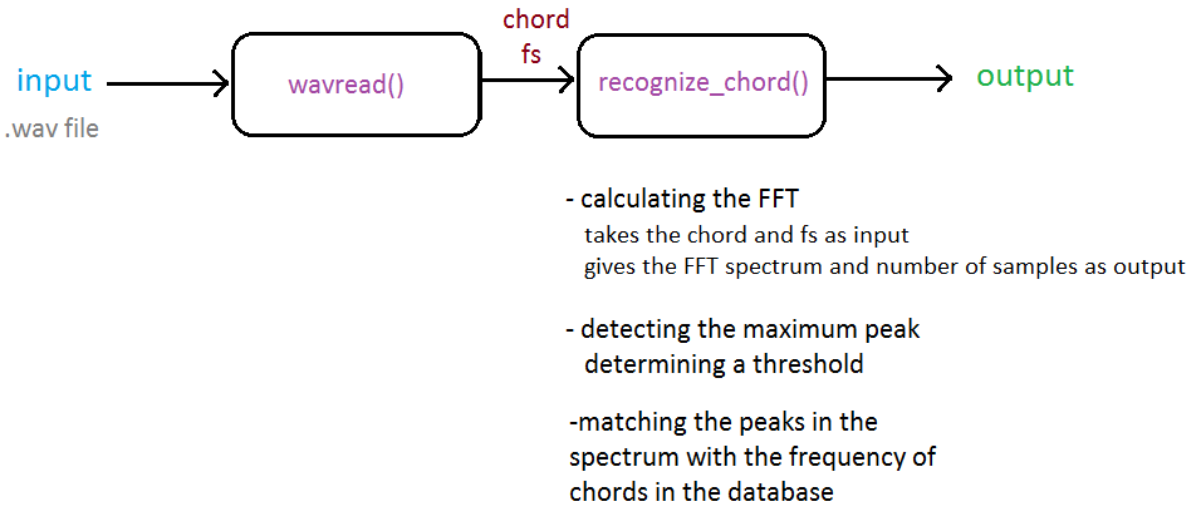
However, to play some chords in guitar, you are required to play more than 3 strings simultanously. This does not mean that you are playing more then 3 different notes. Some of those notes are the octaves of those 3 notes that constitue the chord.

For example to play C chord, you are actually hitting 5 strings. However the notes you are hitting actually are C3, E3, G3, C4 and E4. As we can see C4 and E4 are just the octaves of C3 and E3. For this example in the project we are looking for the presence of frequencies of C3, E3 and G3 in the spectrum of the input chord sound to detect if the input is a C chord or not. We do this procedure for each 24 chord.



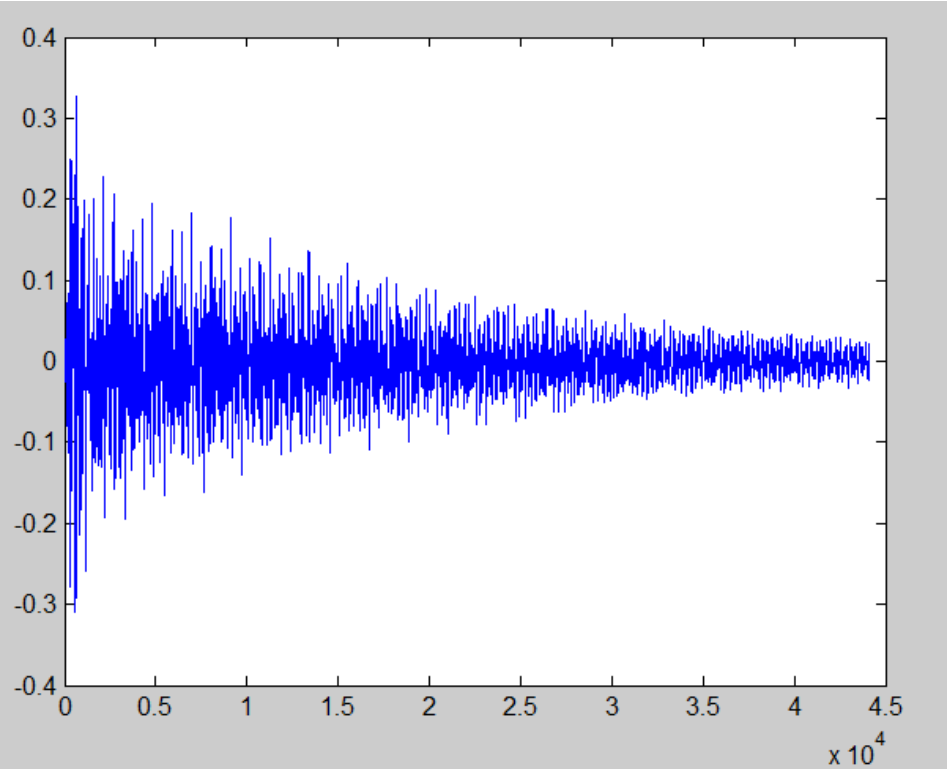
Before starting the process, we needed to create a database including the frequency information of 24 chords, each consisting of 3 notes. In order to do that, we had to determine the frequency values present in each chord and then save this data in MATLAB.

We decided to perform the operation as follows:

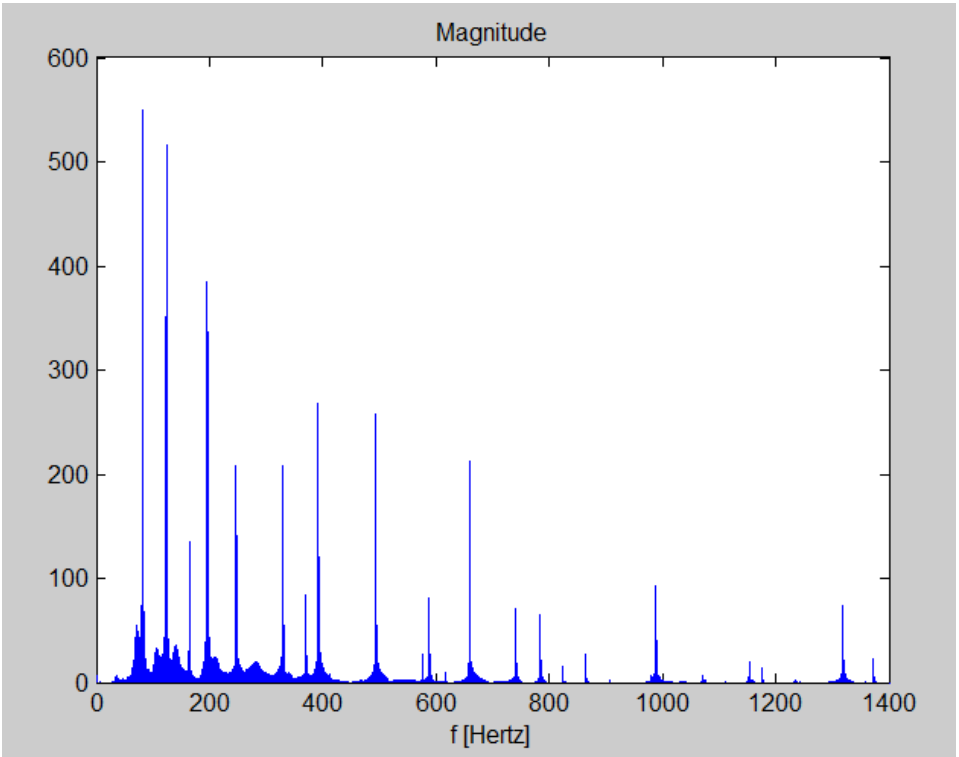


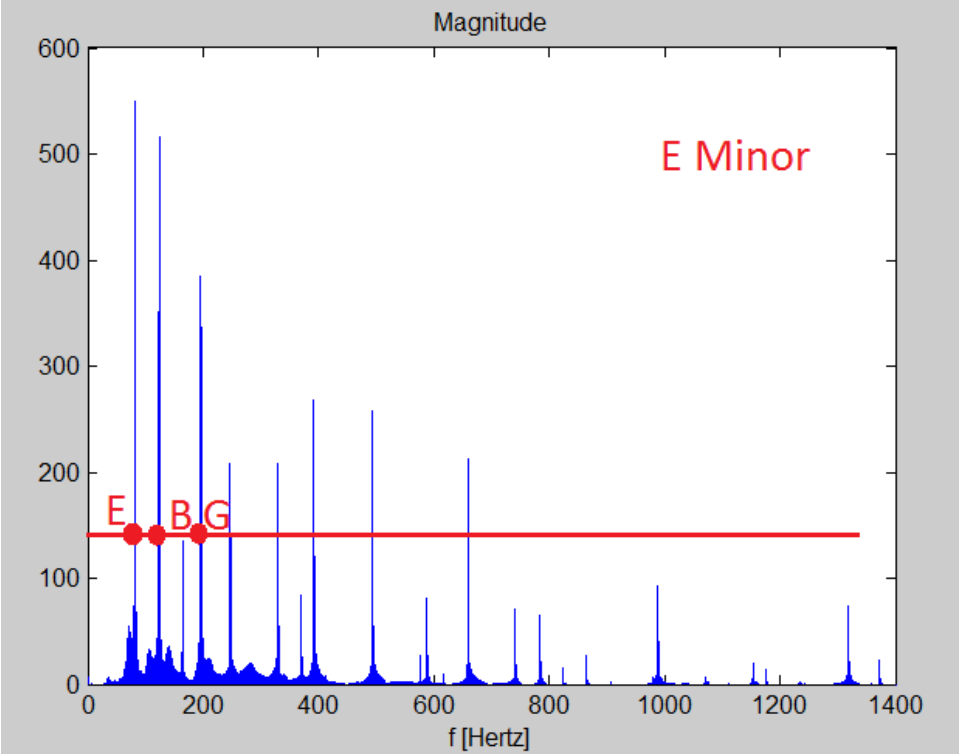
We can visualize the process with an example. Let's consider that our input chord is E minor.

Time Domain Representation



Frequency domain peaks corresponding to E minor are as follows:





IV. IMPLEMENTATION AND RESULTS

a) Creating the Database:

We need 24 chords, which are generated by 3 notes. So in order to define the chords, we need to assign frequency values to them.

The main idea of assigning frequency is as follows:

In guitar, together with majors, minors and sharps, there are 12 notes overall. In every single step taken to the higher direction, the current frequency value is multiplied by $2^{1/12}$, so the frequency increases.

For instance, if we want to find the frequency of the third fret, we have to multiply the open string frequency by $2^{1/12}$ to the power of 3, which is $2^{1/4}$.

The open string frequencies in guitar:

E - 82.4
A - 110
D - 146.8
G - 196
B - 246.9
E - 329.6

In this manner, we performed this operation to every single note and finally obtained the table below.

```
C chord - C E G
C - 131 Hz
E - 164.8
G - 196

Cm chord - C D# G
C - 131
D#- 155.5
G - 196

C# Chord - C# F G#
C# - 138.6
F - 174.6
G#- 207.6

C#m Chord - C# E G#
C# - 138.6
E - 164.8
G# - 207.6
-----------------
D Chord - D F# A
D - 146.8
A - 220
F#- 370

Dm Chord - D F A
D - 146.8
A - 220
F - 349.2

D# Chord - D# G A#
D# - 155.5
G - 392
A# - 233

D#m chord - D# F# A#
D# - 155.5
F# - 370
A# - 233

-----------------
```

```
E chord - E G# B
E - 82.4
B - 123.5
G#- 207.7

Em chord - E G B
E - 82.4
B - 123.5
G - 196

-----------------

F chord - F A C
F - 87.3
A - 110
C - 261.6

Fm chord - F G# C
F - 87.3
C - 130.8
G#- 207.7

F# chord - F# A# C#
F# - 92.5
A# - 233
C# - 138.6

F#m chord - F# A C#
F# - 92.5
A - 110
C#- 277.13
-------------------

B chord - B D# F#
B - 123.5
D#- 311.1
F#- 185

Bm chord - B D F#
B - 123.5
D - 146.8
F#- 370
```

```
G chord - G B D
G - 98
B - 123.5
D - 146.8

Gm chord - G A# D
G - 98
A# - 116.5
D - 146.8

G# chord - G# C D#
G# - 103.8
C - 130.8
D# - 155.5

G#m chord - G# B D#
G# - 103.8
B - 123.5
D# - 155.5

-------------------

A chord - A C# E
A - 110
C# - 277.1
E - 164.8

Am chord - A C E
A - 110
C - 261.6
E - 164.8

A# chord - A# D F
A# - 116.5
D - 146.8
F - 349.2

A#m chord - A# C# F
A# - 116.5
C# - 277.1
F - 349.2
```

Afterwards, with 'import_chordnotesdata' and 'import_chordsdata' functions, we uploaded this database and saved the information in 'chord_notes{}' variables, which we will be using later on for the matching process.

b) Fast Fourier Transform

The main function of this part is to take Fast Fourier Transform of the input chord given.

'my_fft_plot_abs' function takes the input chord and the sampling frequency f_s as inputs, and returns the discrete spectrum, f scale and the number of samples as outputs.

```
function [X,f,n_of_samples] = my_fft_plot_abs(x,fs)
%Parameters:
%x : is the function to calculate its fourier
%fs: sampling frequency of fs
```

Since in FFT, the number of the samples should be a power of two, first we use 'nextpow2' function to find this value and then we take the transform. Afterwards, we use 'fftshift' function to shift zero-frequency component to the center of the spectrum.

```
NFFT = 2^nextpow2(length(x)); %number of samples

%calculating fourier transform
X = fft(x,NFFT);
X=fftshift(X);
```

In the next step, we arrange f-axis such that, it is divided to the number of samples between – $f_s/2$ and + $f_s/2$. Then we draw the magnitude of the Fast Fourier Transform.

```
%arranging f axis
f = fs/2 * linspace(-1,1,NFFT);

X = abs(X);

stem(f,abs(X),'marker','none');
title('Magnitude');
xlabel('f [Hertz]');
```

The function below plots the phase of the Fast Fourier Transform. This part was only for our information.

```
function [X,f] = my_fft_plot_phase(x,fs)
%Parameters:
%x : is the function to calculate its fourier
%fs: sampling frequency of x

%calculating fourier transform
X = fft(x);
X=fftshift(X);

%arranging f axis
f = fs/2 * linspace(-1,1,length(x));

plot(f,angle(X));
title('Phase Spectrum');
xlabel('f [Hertz]');
```

c) Recognizing the Chord:

This part performs the main function of the project.

First of all, it uses 'import_chordnotesdata' function and activates the database we created. Then, 'my_fft_plot_abs' function takes the Fourier Transform.

```
function [] = recognize_chord(input_chord,fs)

import_chordnotesdata();

[spectrum f n_of_samples] = my_fft_plot_abs(input_chord,fs);
```

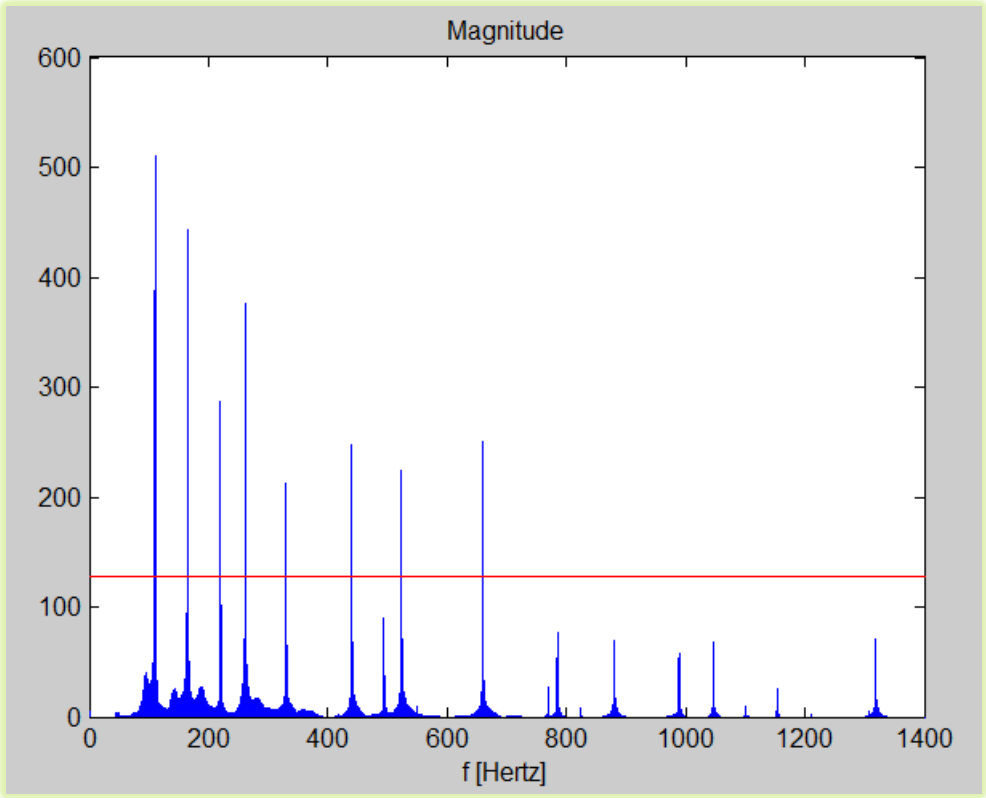
In this part, first of all we detect the maximum value in the spectrum and define a variable called 'level', which is one fourth of this maximum peak value in the spectrum and this will be our threshold for the comparison process in the next part. The peaks in the specrum which are lower than this level are disregarded and not used for matching process to recognize the chord.

```
level = max(spectrum)/4;
%draw red line at level
plot(1:1400,level,'r');
```

After that, we define a new variable called ‘peak_indexes’, which corresponds to the index of spectrum peaks that are above the threshold, so ‘level’. In order to avoid the indexes which correspond to negative frequency values, we extracted just the necessary part(positive side of the frequency axis) from peak_indexes.

```
peak_indexes = find(spectrum>level);
peak_indexes = peak_indexes(size(peak_indexes)/2 + 1:size(peak_indexes),1);
peaks = f(peak_indexes);
```

The spectrum peaks together and the red line drawn



Peaks of A-minor

We continue with the note detection part, in which we run a comparison and match the peaks of the input chord with the chords in our database.

We define a new threshold value, this time as a lower bound for the difference between the peaks frequency values of our input chord, and the peak frequency values defined in our database.

We run two for loops, one with index 24, since we have 24 chords to compare overall, and the other one with 3, since the comparison will end when we detect 3 notes, so 3 peaks.

Additionally, we have a control parameter called ‘count’, which causes the for loops to end, when 3 suitable notes are detected.

The main logic of this matching process is as follows; for each of the 24 chords in our database, we take the difference between the peak values of them and the input chord. (We also check the second harmonic – this will be explained in the ‘problems’ section).

If the minimum value among these differences is smaller than the threshold value defined, this means that the peak compared, so the note compared, is included in our chord. Thus, the control value is increased by 1. This loop ends when the control value reaches 3, so when 3 peaks are detected.

```
%we obtained peaks
%matching
threshold = 2;
count = 0;
for c=1:24
    for i=1:3
        if( min(abs(chord_notes{c}(i) - peaks)) <threshold )...
         || (min(abs(chord_notes{c}(i)*2 - peaks)) <threshold )
              count = count + 1;
        end

    end
```

After we are done with the comparison, we move to the part where we match our chord with the corresponding one in our database, and print out its name on the screen.

```
    if(count ==3)
        chord_number = c;
        disp(get_chordname(chord_number));
        count = 0;
        break;break;break;
    end
    count = 0;
end

end
```

d) The Commands by the User:

The user should perform the steps below to reach the output.

The second line reduces the dimension of the input from 2 to 1. In other words, makes it mono from stereo. So the input becomes suitable for the operation.

```
>> [chord fs nbits]=wavread('Em.wav');
>> chord=chord(:,1);
>> recognize_chord(chord,fs);


E minor
```

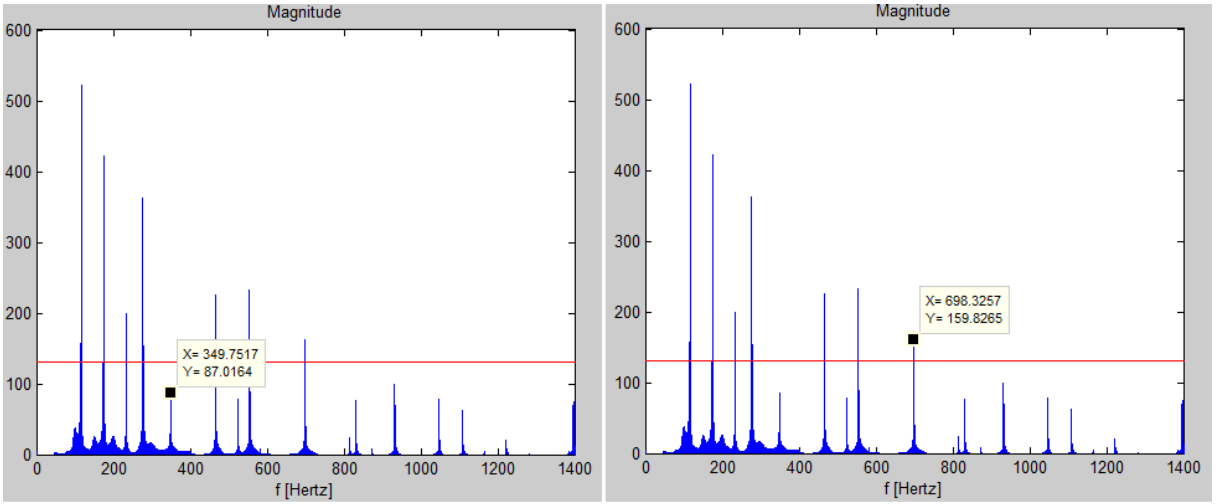
e) Problems Faced:

We had a problem during the recognition of A#minor chord. To recognize A#minor chord we are looking for the presence of the frequencies A#(116.5 Hz) , C#(277.1 Hz) , F (349.2 Hz).

In the spectrum we were able to detect the peaks corresponding to A# and C#, however the peak at F(349.2 Hz) was not high enough so it stayed below the level = max/4 line.

To solve this problem, we also looked at the harmonics of the notes because it is possible that peak at the fundamental frequency can be low and peak at the second harmonic can be high. Therefore we also considered the second harmonics of the notes.

By this way, we were able to detect A#minor chord.



As we can see in the figure above, F note (349.2 Hz) stays below the level line, so it is disregarded as a peak, however second harmonic of it 2*349.2 ~= 698.4 does stay above the line. So, by looking at second harmonics we solve this problem.

V. CONCLUSION

In our project, we were engaged with Guitar Chord Recognition. First, we built a database, during which we got familiar with the calculation of frequency values of music notes. Then we uploaded this data to MATLAB. Since we created our own library, we were ready to start with the main operation.

Our idea was to take the Fast Fourier Transform of our input chord, and then detect the peak values of it by comparing them with a threshold value. Finally, in the 'matching' part, we performed comparisons between the peak frequency values which were detected in the previous section and frequency of chords in the database. Thus, we were able to recognize the chord.

All in all, we are able to recognize each 24 chord sounds with %100 success.

VI. REFERENCES

- Recognizing Guitar Chords in Real Time – Micheal Goold *(http://undergraduate.csse.uwa.edu.au/year4/Current/Students/Files/2009/MichaelGoold/CorrectedDissertation.pdf)*