Fall 2020

EE 210
Mini-Project #: 02

Z-transform & Sampling Process

Last Name: **Aldacher**
First Name: **Muhammad**
ID: **011510317**
Date: **11/7/2020**

TABLE OF CONTENTS

Section 1: Abstract

This project is about designing a digital filter that filters out certain noise frequencies in an input signal using MATLAB. The design process includes finding the Z-transform of the filter's impulse response and finding the locations of the poles and zeros in the z-plane to filter out the disturbing tones. Digital signal processes like discrete convolution, Z-transform, and digital filtering are used to clean up the input voice signal and produce the desired output spectrum. This project gives a deeper understand of how IIR digital filters are designed using Z-transform and how digital filters can be used in future signal-processing applications.

Index Terms—Z-Transform, Discrete-time, IIR Filters, Signal-Processing, Matlab.

Section 2: Introduction

Z-transform is a discrete-time transform that is based on discrete-time Fourier transform (DTFT) and is commonly used in the process of designing digital filters. It is considered like the Laplace transform but in the discrete time domain. Z-transform takes a discrete input whose data is sampled by a sampling frequency of f_S and gives a continuous output in the z-domain. The causality or non-causality of a system defines the region of convergence (ROC) of the Z-transform, which specifies the values of z that allows the summation of the Z-transform given in equation (1) to converge. In digital filter design, the poles and zeros of the Z-transform of the filter's impulse response H(z) define the properties of the filter like the cut-off frequencies, the quality, and its stability. In the next sections, the steps of designing a digital filter used to cleanup an input signal that contains disturbing tones is discussed.

$$X(Z) = \sum_{n=-\infty}^{\infty} x[n] \cdot Z^{-n} \qquad (1)$$

Section 3: Design

This section will explain the process of designing the digital filter using MATLAB to filter out the disturbing tones in an input audio file. The MATLAB codes used are provided in appendix A.

3.1 Discrete-Time Fourier Transform

The first step is to perform discrete-time Fourier transform (DTFT) on the input signal and analyze its frequency response. The DTFT is defined by equation (2) and the digital frequency is defined by equation (3). By observing the frequency spectrum, the disturbing tones in the input signal can be spotted.

$$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-j\Omega n} \qquad (2)$$

$$\Omega = \frac{2\pi f}{fs} \qquad (3)$$

3.2 Filter Design

After identifying the disturbing tones in the input signal, a digital IIR notch filter is designed to remove these specific tones, as shown in Fig. 1. Assuming the disturbing tones are at frequencies f_1 and f_2, then the digital frequencies are calculated as shown in equations (4) and (5).

$$\Phi_1 = \frac{2\pi f_1}{fs} \tag{4}$$

$$\Phi_2 = \frac{2\pi f_2}{fs} \tag{5}$$

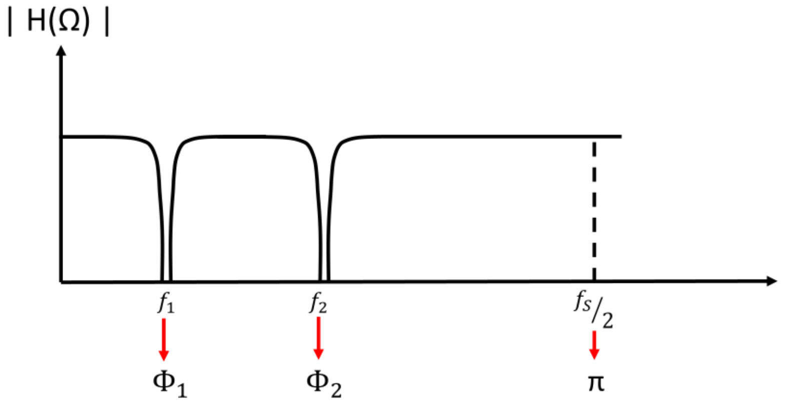


Fig.1. Magnitude of the frequency response of a notch filter to remove 2 disturbing tones

The digital frequencies are used to determine the angle at which the pole-zero pair is placed on the Z-plane, as shown in Fig. 2. The magnitude of the poles and zeros affect the sharpness of the notch and the amount of attenuation. The closer the zero and pole are to the unit circle, the sharper and more selective the notch gets. But as the pole-zero pair get closer to each other, the less attenuation happens at the specified frequency. To increase the attenuation, the order of the pole and zero can be increased. Equation (6) shows the transfer function for the designed filter with 4 poles and 4 zeros. The hand analysis of the design process is shown in appendix B.

$$H(z) = \frac{(z-Z_1).(z-Z_2).(z-Z_3).(z-Z_4)}{(z-P_1).(z-P_2).(z-P_3).(z-P_4)} \tag{6}$$

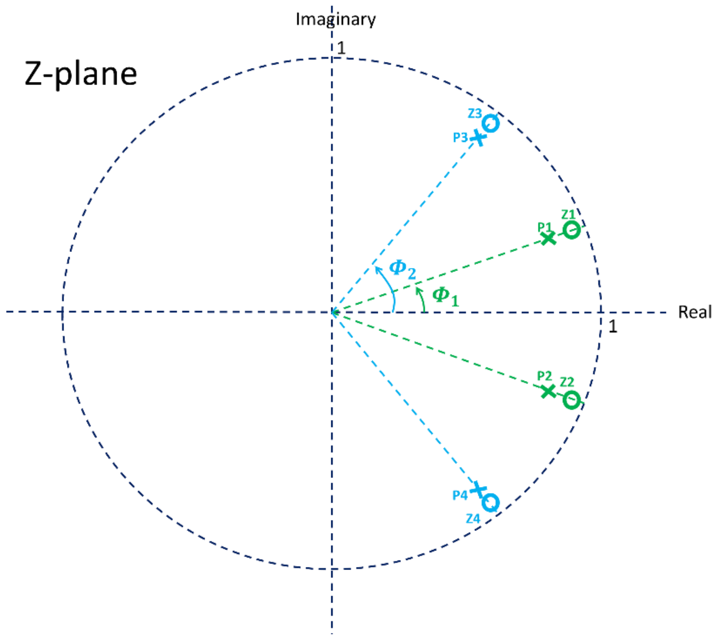
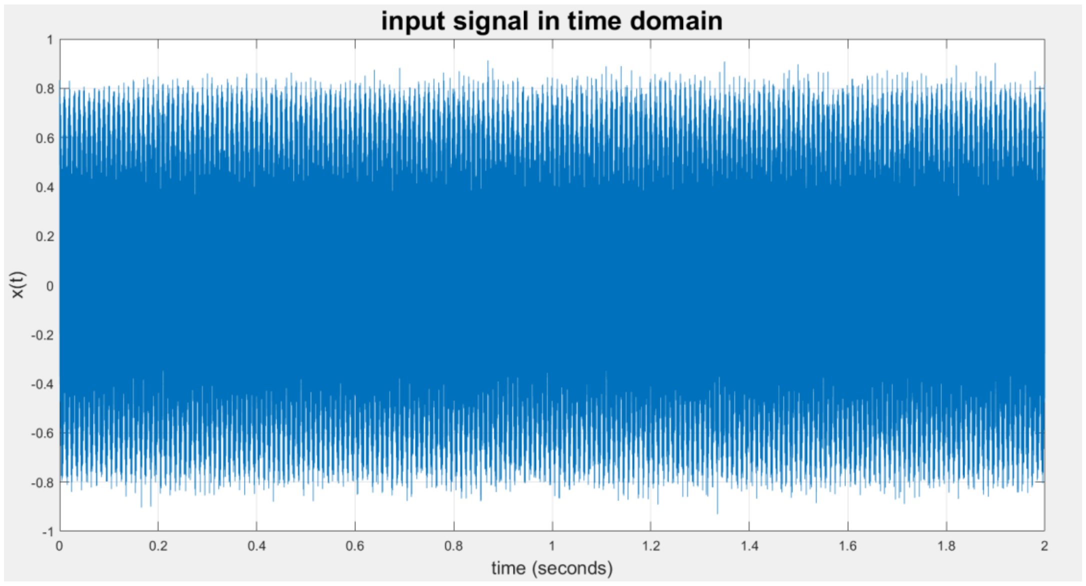


Fig.2. Pole-Zero locations of the notch filter that removes 2 disturbing tones

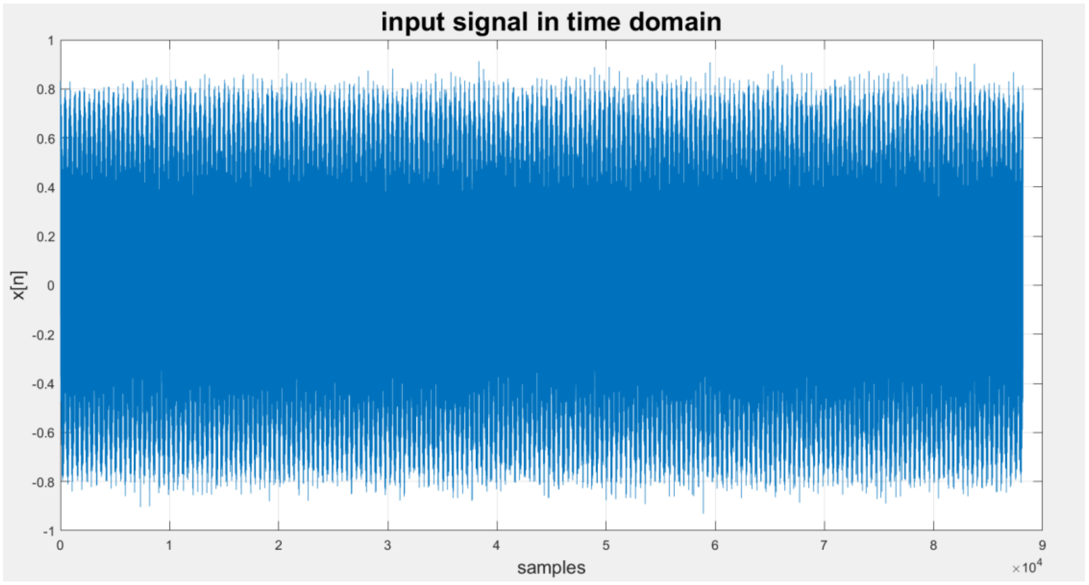
Section 4: Results and Discussion

4.1 Basic Design

[a] The input signal x[n] is an audio signal of a musical note with some disturbing tones added to it that make the musical note hard to be heard. What is heard is a very strong and annoying high-pitched tone covering up the musical note. [b] Fig. 3 shows the input signal plotted in time domain. The sampling rate f_S of this signal is 44100 Hz and its length is 2 seconds. From the time domain plots, it is difficult to tell any information about the disturbing tones.



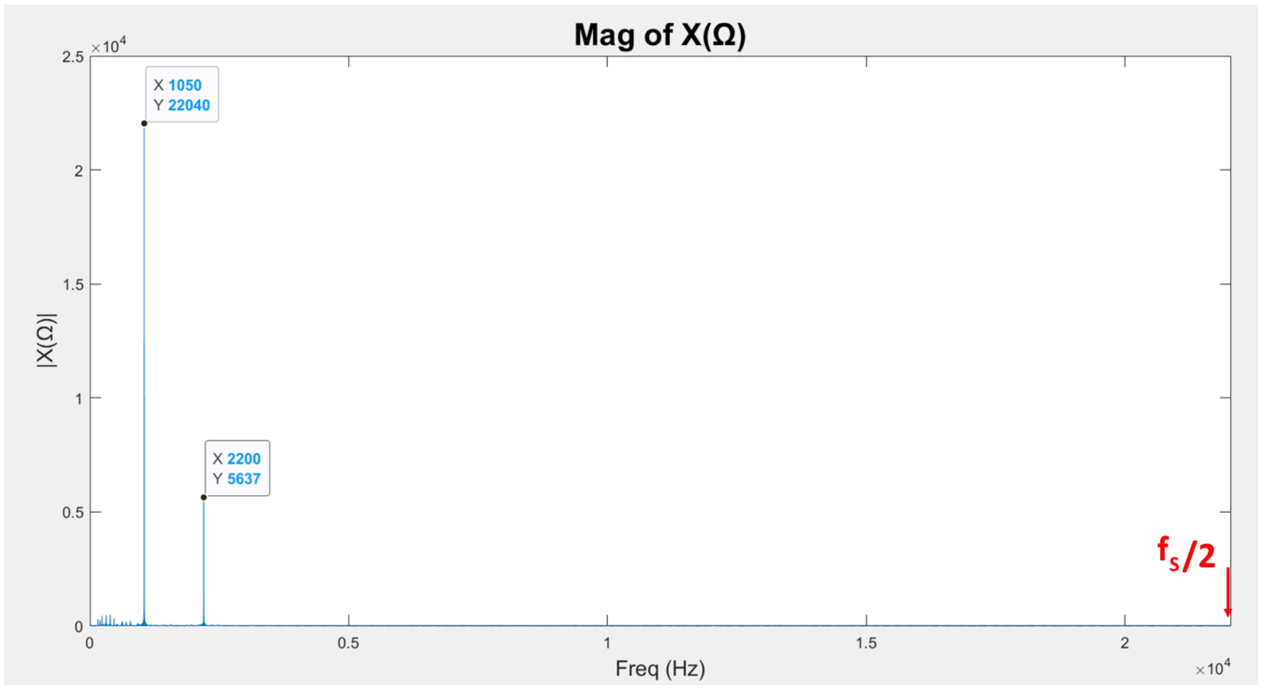
(a)



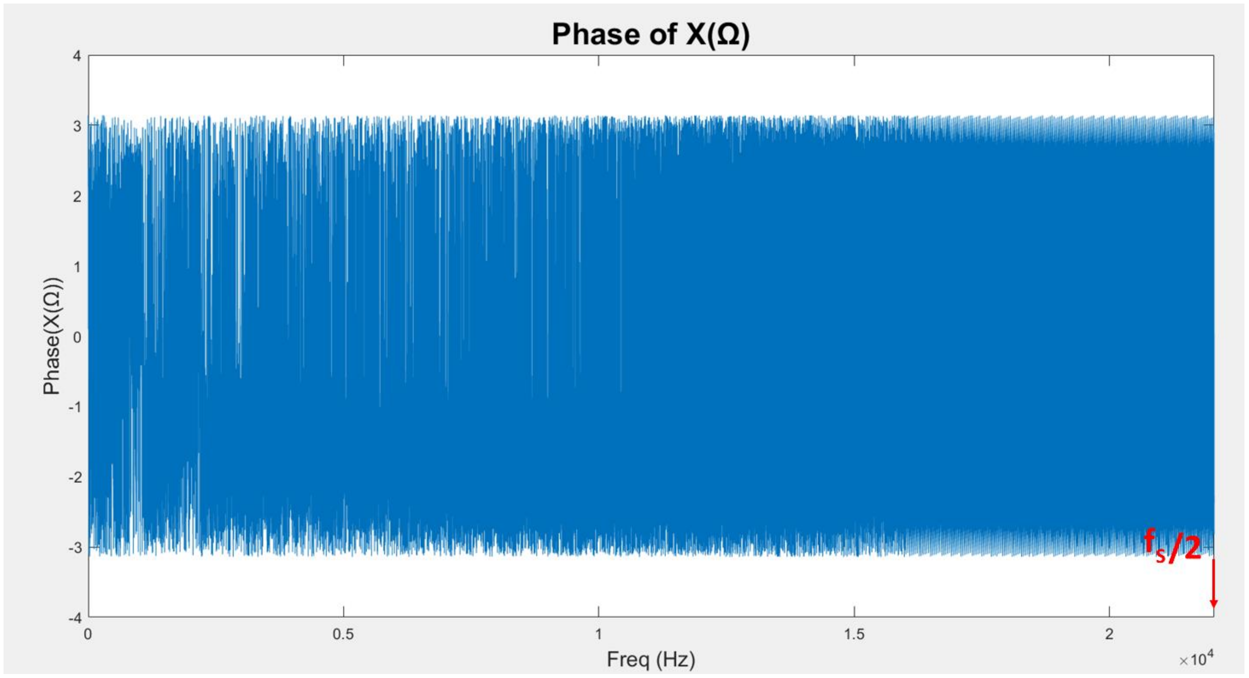
(b)

Fig.3. Input signal x[n] in time domain (a) vs time, (b) vs samples

[c] Using the DTFT code made in mini-project 01, the spectrum of x[n] can be obtained. The frequency response of the input audio signal, X(Ω), vs frequency f is shown in Fig. 4, and vs the digital frequency Ω in Fig. 5. [d] From the spectrum magnitude of the audio signal, it is clear that there are 2 disturbing tones of high magnitude at frequencies 1050 Hz and 2200 Hz. Those are equivalent to Ω = 0.15 rad and 0.313 rad, respectively.

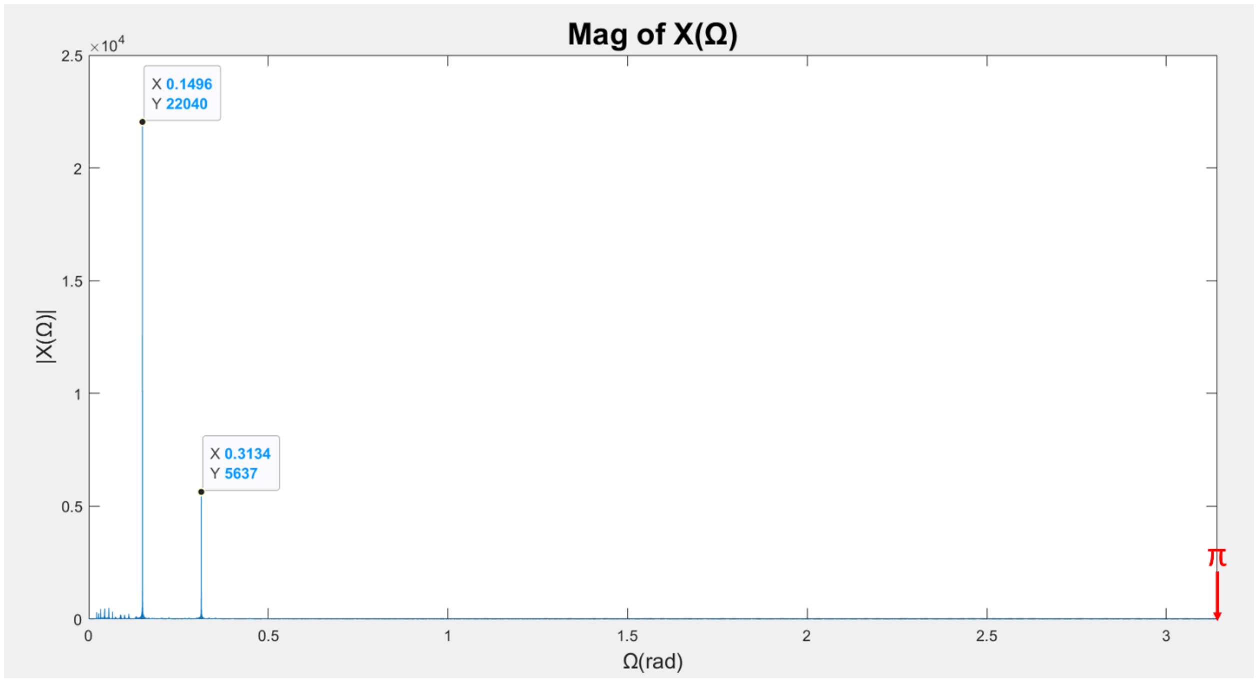


(a)

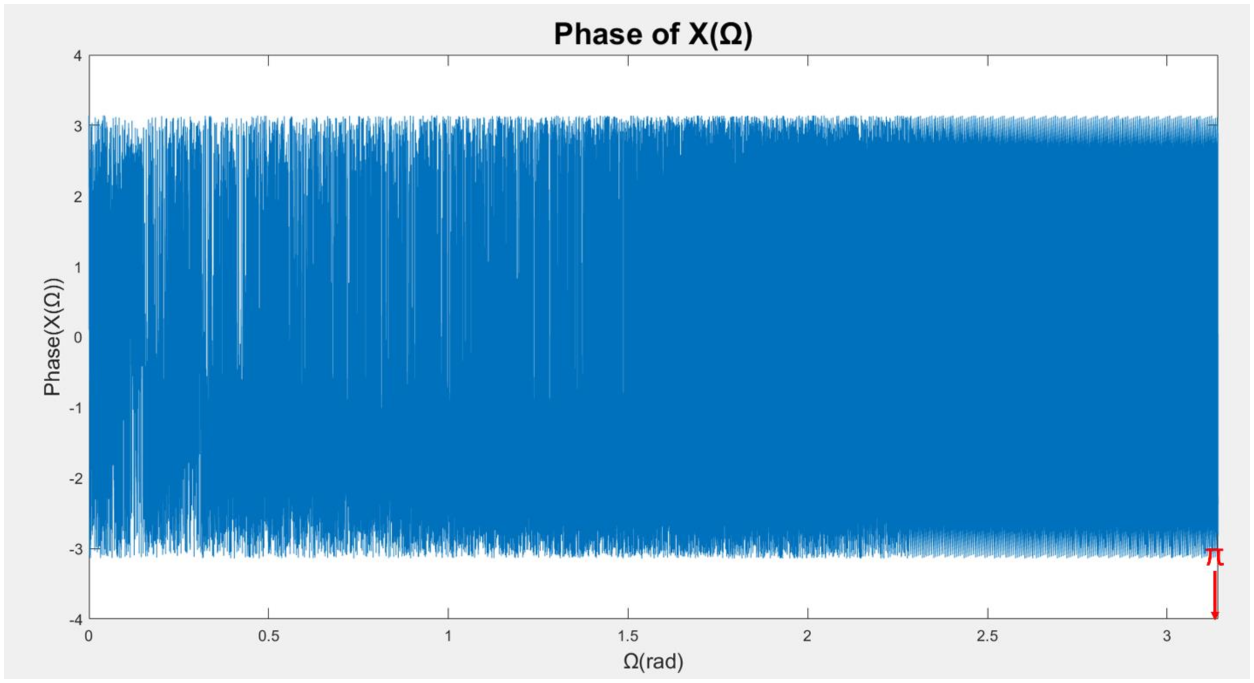


(b)

Fig.4. Frequency response of the input signal X(Ω) vs frequency f (a) Magnitude, (b) Phase



(a)



(b)

Fig.5. Frequency response of the input signal X(Ω) vs digital frequency Ω (a) Magnitude, (b) Phase

[e] An IIR notch filter is designed to attenuate these 2 disturbing tones, by creating zero-pole pairs in the Z-plane (or the complex plane). These pairs are within the unit circle for the filter to be stable and at angles equal to the digital frequencies ($\Omega = 0.15$ rad and 0.313 rad). [f] The transfer function of the notch filter is defined by equation (7). The Z_n and P_n represent the zeros and poles respectively and are defined as in (8) and (9). [g] Using the "conv" function in MATLAB

to perform polynomial multiplications, the coefficients vectors N and D are obtained as the coefficients of the numerator and the denominator of H(z) respectively and are as shown in (10) and (11) .

$$H(z) = \frac{N(z)}{D(z)} = \frac{(z-Z_1).(z-Z_2).(z-Z_3).(z-Z_4)}{(z-P_1).(z-P_2).(z-P_3).(z-P_4)} \quad (7)$$

$$Z_{1,2} = 0.99\, e^{\pm j\, 0.15} \ \ ,\ Z_{3,4} = 0.99\, e^{\pm j\, 0.313} \quad (8)$$

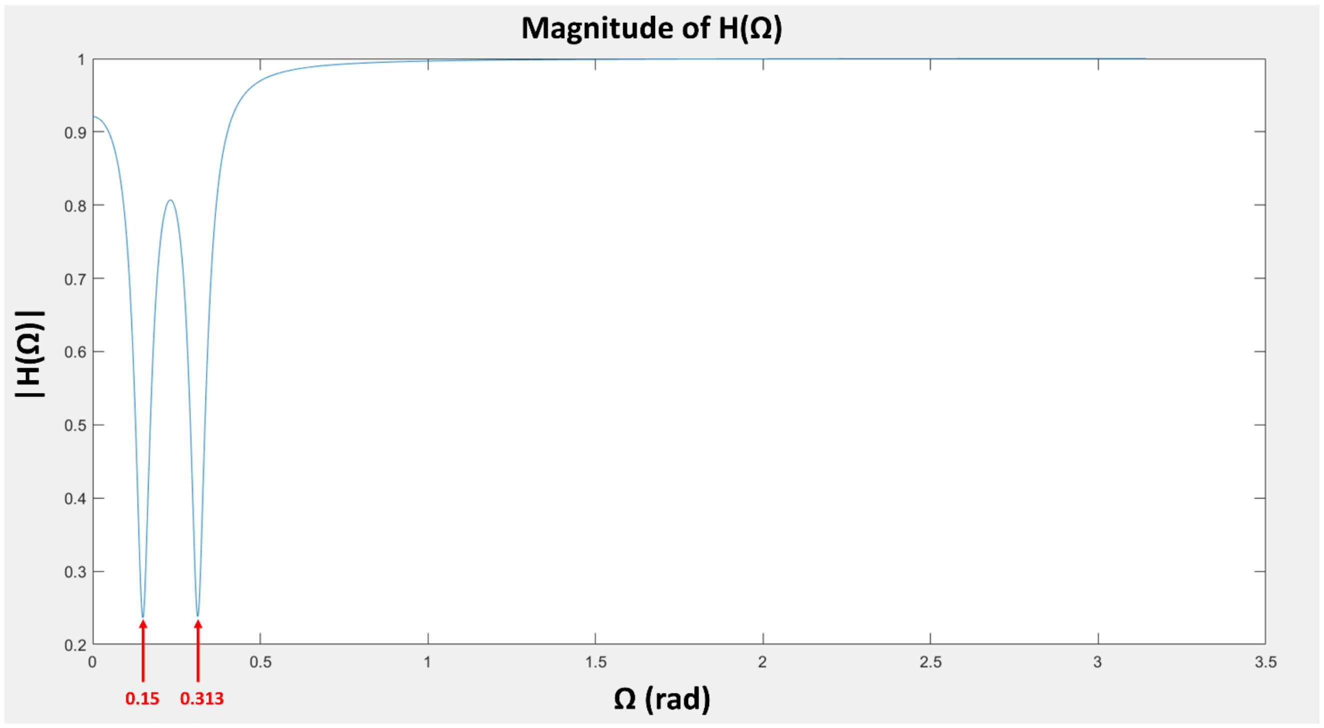
$$P_{1,2} = 0.96\, e^{\pm j\, 0.15} \ \ ,\ P_{3,4} = 0.96\, e^{\pm j\, 0.313} \quad (9)$$

$$N = [1.0000 \quad -3.8414 \quad 5.6479 \quad -3.7650 \quad 0.9606] \quad (10)$$

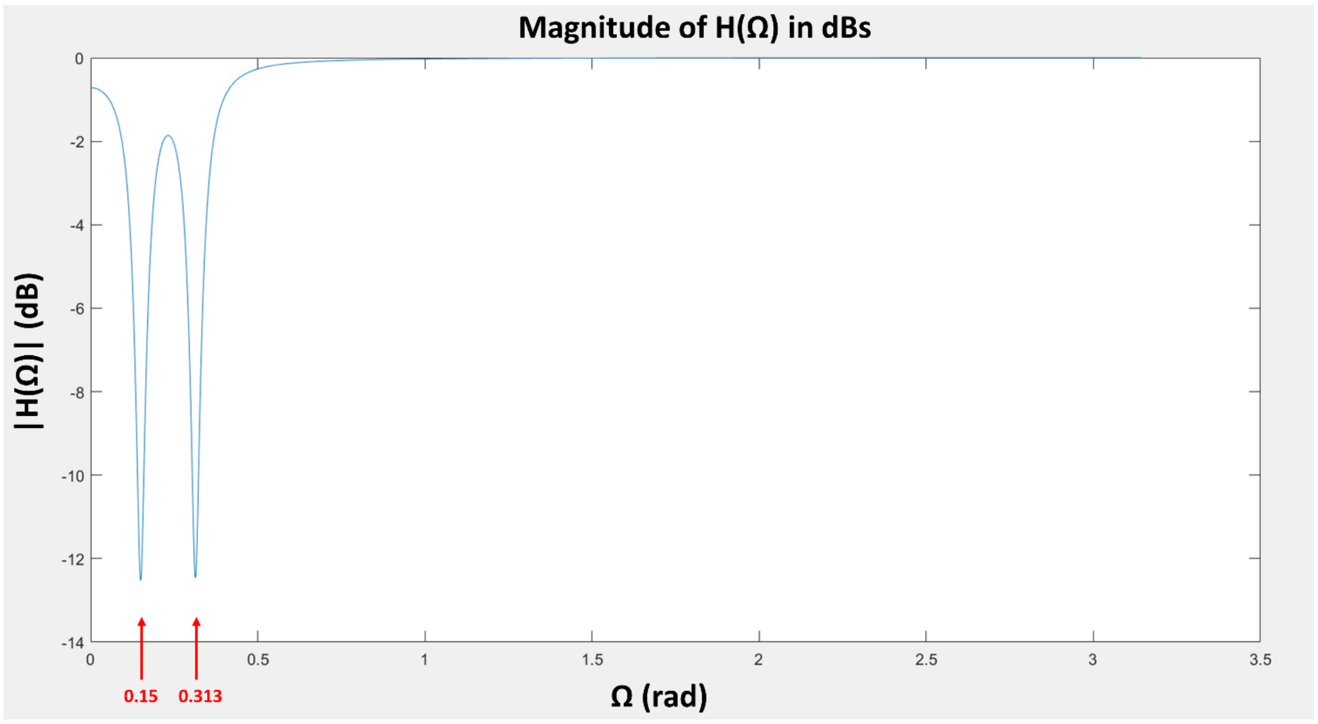
$$D = [1.0000 \quad -3.7250 \quad 5.3108 \quad -3.4330 \quad 0.8493] \quad (11)$$

[h] Fig. 6 shows the magnitude of the frequency response of the filter H(Ω) in normal scale and in dB scale, after normalizing the transfer function (so the maximum value is 1 or 0 dB). H(Ω) in dBs can be calculated using equation (12). Fig. 7 shows the phase of H(Ω). [i] The locations of the poles and the zeros are plotted in Fig. 8 using the “zplane” function in MATLAB.

$$H(\Omega)\ in\ dBs = 20.\log_{10}|H| \quad (12)$$



(a)



(b)

Fig.6. Magnitude of the filter's transfer function H(Ω) (a) in normal scale, (b) in dB scale

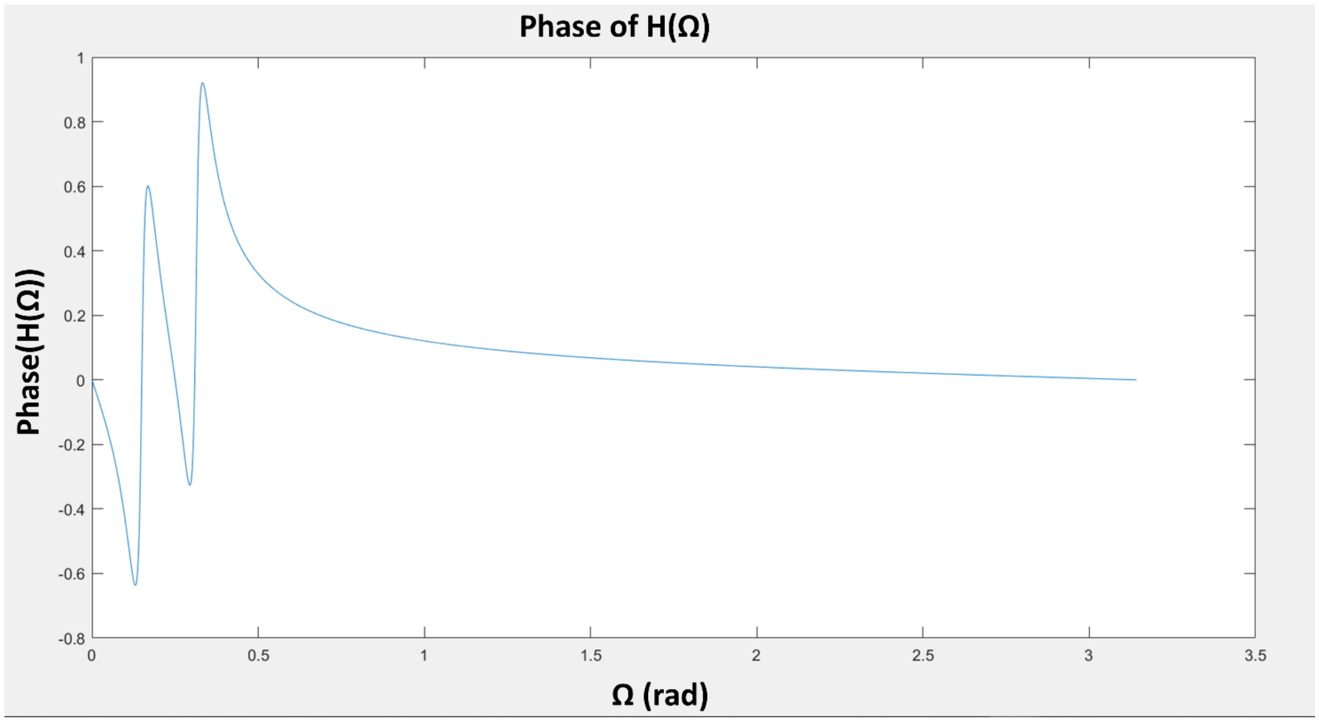


Fig.7. Phase of the filter's transfer function H(Ω)

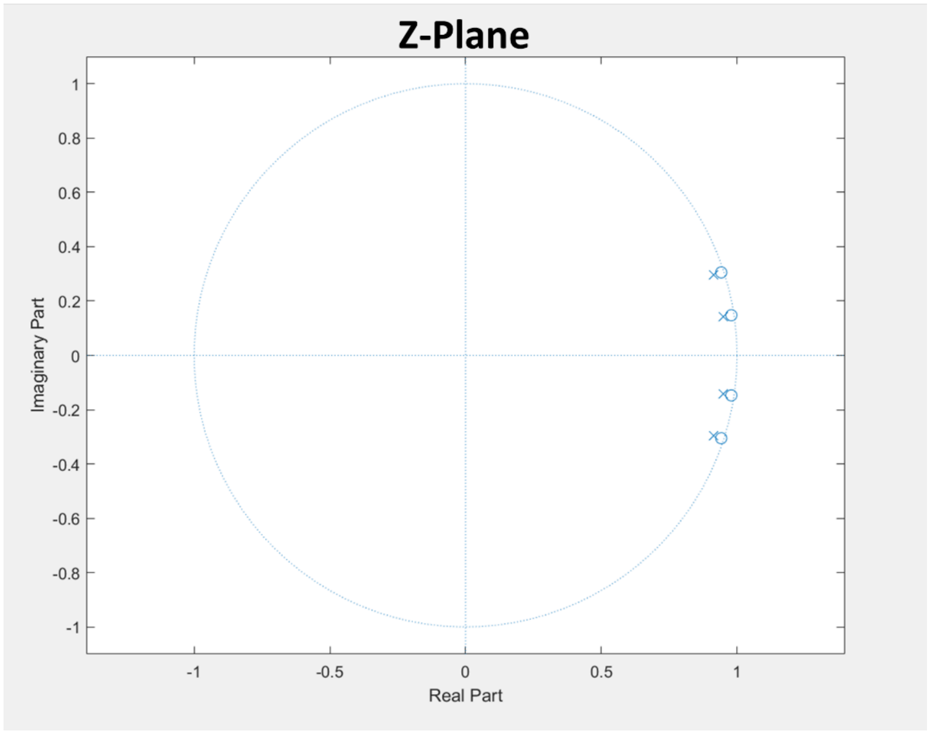
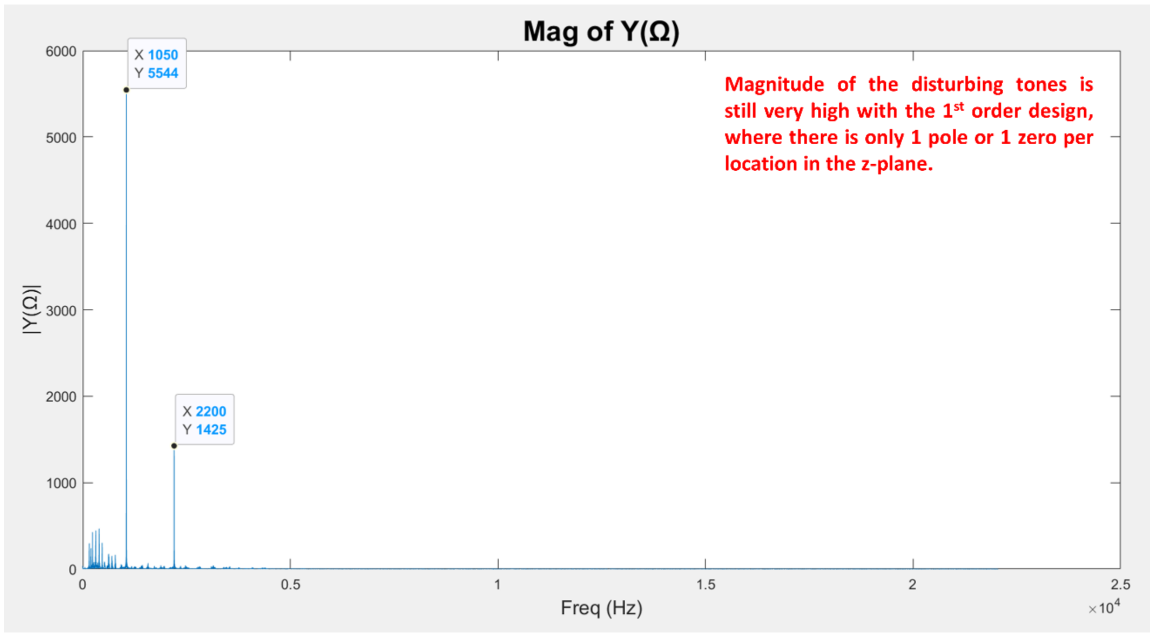


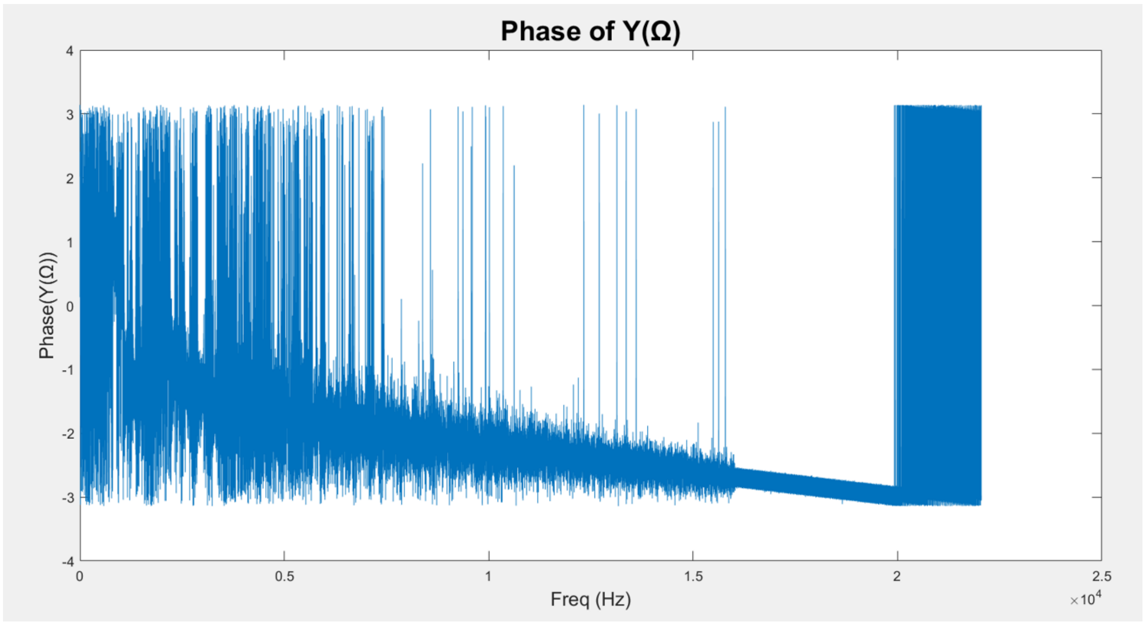
Fig.8. Locations of the poles and the zeros of H(Ω) in the z-plane

[j] To perform the filtering process on x[n] and produce a cleaner output y[n], the "filter" function of MATLAB is used according to equation (13). [l] The frequency response of y[n] in Fig. 9 shows that the disturbing tones still exist but are slightly attenuated. [k, m] Using the "sound" function of MATLAB to playback the filtered output, it is apparent that the tones are attenuated, but still loud enough to cover the music. To increase the attenuation of these tones, higher-order poles and zeros must be used, as will be shown in the next sub-section.

$$y = filter(N, D, x) \qquad (12)$$



(a)



(b)

Fig.9. Frequency response of the output signal Y(Ω) vs frequency (a) Magnitude, (b) Phase

4.2 Higher-Order Design

[n] The notch filter is re-designed to have higher-order poles and zeros at the same locations mentioned in the previous sub-section to increase the attenuation of the disturbing tones. To have 5th order poles and 5th order zeros per location, 5 filters of transfer function H(z) are cascaded as shown in Fig. 10 to create the new transfer function in equation (13).

$$H_{new}(z) = \frac{N(z)}{D(z)} = \frac{(z-Z_1)^5.(z-Z_2)^5.(z-Z_3)^5.(z-Z_4)^5}{(z-P_1)^5.(z-P_2)^5.(z-P_3)^5.(z-P_4)^5} \qquad (13)$$

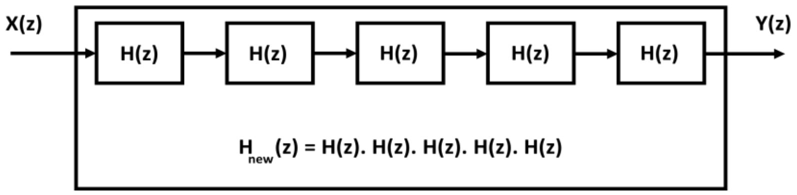
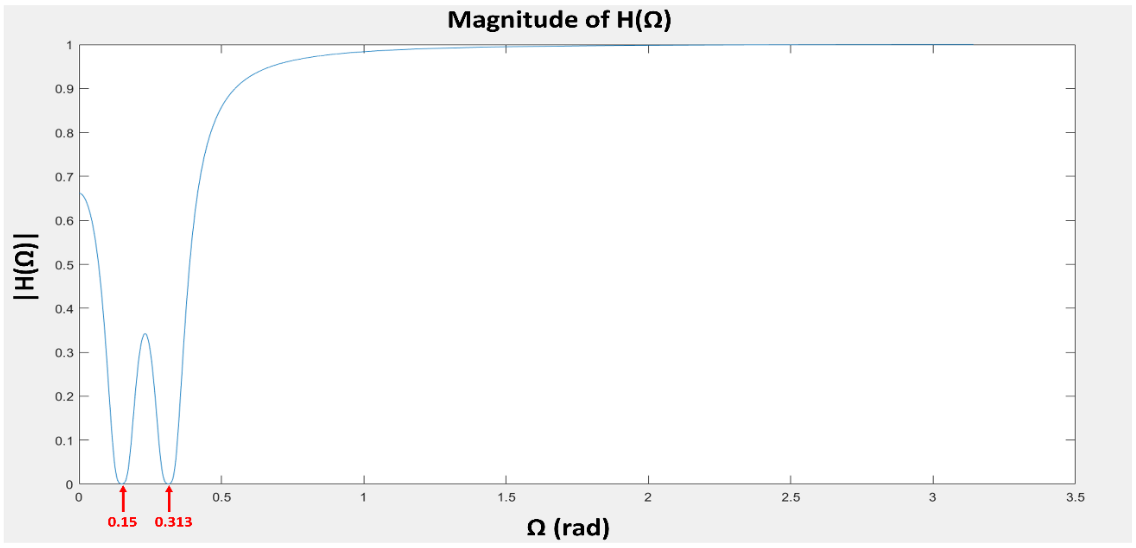
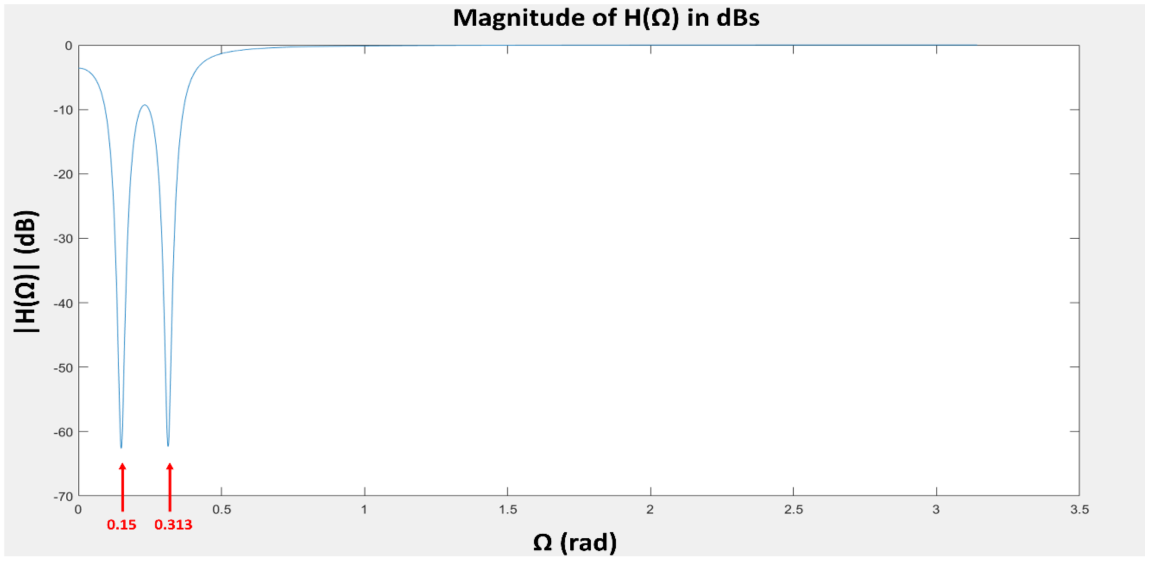


Fig.10. Cascaded Filters

Fig. 11 shows the magnitude of the frequency response of the new filter. The attenuation at the disturbing frequencies is increased 5 times from 12.5 dB to 62.5 dB. The output produced by filtering x[n] with the cascaded filters is shown in Fig. 12. The disturbing tones are almost completely attenuated and can hardly be heard when playing back the new output.

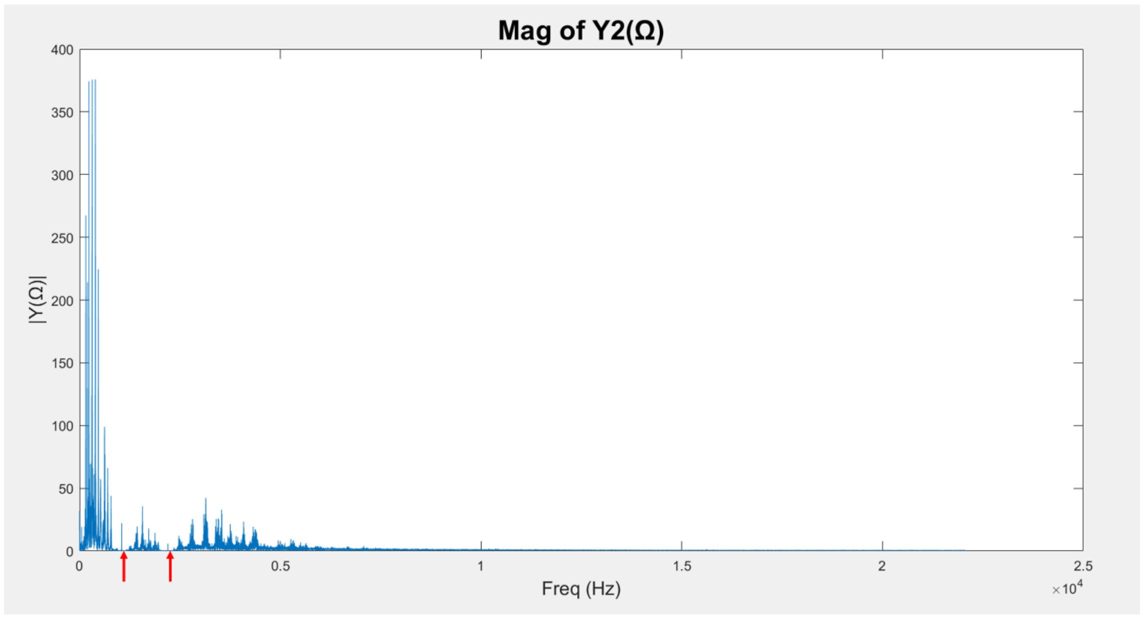


(a)

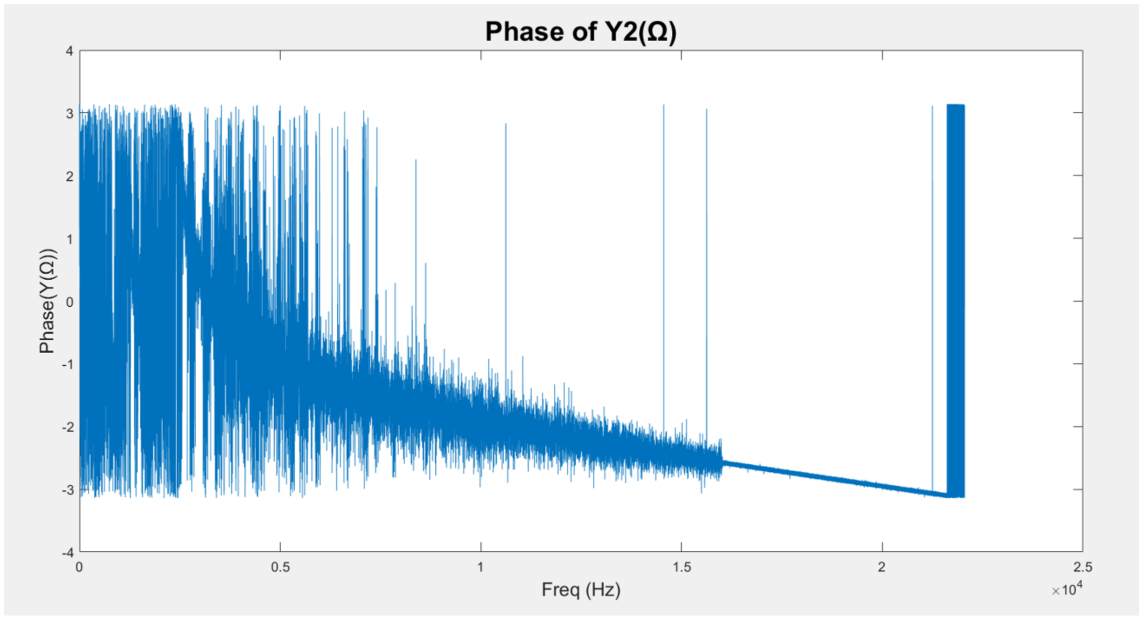


(b)

Fig.11. Magnitude of the new filter's transfer function (a) in normal scale, (b) in dB scale



(a)



(b)

Fig.12. Frequency response of the new output signal $Y_2(\Omega)$ vs frequency (a) Magnitude, (b) Phase

Section 5: Conclusion

[o] This project is about designing an IIR digital filter to remove the noise tones from an audio signal. The design process includes performing discrete-time Fourier transform to spot the disturbing notes in the input's spectrum, then determining the zero and pole locations in the z-domain for the transfer function of the notch filter before producing the cleaner output signal. The learning outcomes of this project include understanding how the Z-transform is used to implement an IIR filter and how the location of the zero-pole pair can affect the filtering process. The closer the pole and zero are to each other and to the unit circle, the more selective the filter becomes but the less attenuation the filter offers. To increase the attenuation of the disturbing frequencies, the order of the poles and the zeros need to increase. During the coding process, MATLAB functions in the signal-processing toolbox like "conv" and "filter" are utilized to put the whole system together and produce a clean output by filtering the input signal. This project can be further expanded to design IIR filters for more complex systems and applications.

Appendix A: MATLAB codes

A.1: Filter Design – Main Code:

```
clc; clear all; close all;
[x,fs] = audioread('mini_proj.wav');
t=0:(1/fs):(length(x)/fs); t=t(1:end-1);

figure; plot(x);title('input signal in time domain',"FontSize",20); grid on;
ylabel('x[n]',"FontSize",14); xlabel('samples',"FontSize",14);

figure; plot(t,x);title('input signal in time domain',"FontSize",20); grid on;
ylabel('x(t)',"FontSize",14); xlabel('time (seconds)',"FontSize",14);

%-----------------------------------------
% Performing DTFT
[X, OM] = my_dtft(x);
X_mag = abs(X);
X_ph = angle(X);
freq = OM*fs/(2*pi);

% Spectrum Vs Frequency
figure; plot(freq,X_mag); title('Mag of X('+string(char(937))+')',"FontSize",20);
ylabel('|X('+string(char(937))+')|',"FontSize",14); xlabel('Freq (Hz)',"FontSize",14);
xlim([0 fs/2]);
figure; plot(freq,X_ph); title('Phase of X('+string(char(937))+')',"FontSize",20);
ylabel('Phase(X('+string(char(937))+'))',"FontSize",14); xlabel('Freq (Hz)',"FontSize",14);
xlim([0 fs/2]);

% Spectrum Vs Digital Frequency
figure; plot(OM,X_mag); title('Mag of X('+string(char(937))+')',"FontSize",20);
ylabel('|X('+string(char(937))+')|',"FontSize",14);
xlabel(string(char(937))+'(rad)',"FontSize",14);
xlim([0 pi]);
figure; plot(OM,X_ph); title('Phase of X('+string(char(937))+')',"FontSize",20);
ylabel('Phase(X('+string(char(937))+'))',"FontSize",14);
xlabel(string(char(937))+'(rad)',"FontSize",14);
xlim([0 pi]);

%-----------------------------------------
% Notch filter
f1 = 1050; f1n = f1/(fs/2);
Th1 = f1n*pi;

f2 = 2200; f2n = f2/(fs/2);
Th2 = f2n*pi;

NC1 = 0.99;   NC2 = 0.99;
DC1 = 0.96;   DC2 = 0.96;

OM = 0:0.0001:pi;
z = exp(j*OM);

N1 = z - NC1*exp(j*Th1); N2 = z - NC1*exp(-j*Th1); N3 = z - NC2*exp(j*Th2); N4 = z -
NC2*exp(-j*Th2);
D1 = z - DC1*exp(j*Th1); D2 = z - DC1*exp(-j*Th1); D3 = z - DC2*exp(j*Th2); D4 = z -
DC2*exp(-j*Th2);

Num = ((N1.*N2).^1).*((N3.*N4).^1);
Den = ((D1.*D2).^1).*((D3.*D4).^1);
H = Num./Den;
H_mag = abs(H)/(max(abs(H)));
H_ph = angle(H);
figure; plot(OM,H_mag)
figure; plot(OM,H_ph);
```

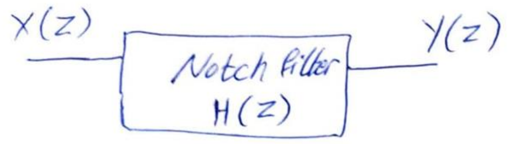
```
H_db = 20*log10(H_mag);
fq = OM*fs/(2*pi);
%figure; plot(fq,H_db);
figure; plot(OM,H_db);

%----------------------------------------
% Coefficients of H
n1 = conv([1,-1*NC1*exp(j*Th1)],[1,-1*NC1*exp(-j*Th1)]);
n2 = conv([1,-1*NC2*exp(j*Th2)],[1,-1*NC2*exp(-j*Th2)]);
N = conv(n1,n2)

d1 = conv([1,-1*DC1*exp(j*Th1)],[1,-1*DC1*exp(-j*Th1)]);
d2 = conv([1,-1*DC2*exp(j*Th2)],[1,-1*DC2*exp(-j*Th2)]);
D = conv(d1,d2)

figure; zplane(N,D)

%----------------------------------------
% Filtered Output (1st order poles & zeros)
y = filter(N,D,x);
[Y, OM] = my_dtft(y);

Y_mag = abs(Y);
Y_ph = angle(Y);
freq = OM*fs/(2*pi);

figure; plot(freq,Y_mag); title('Mag of Y('+string(char(937))+')',"FontSize",20);
ylabel('|Y('+string(char(937))+')|',"FontSize",14); xlabel('Freq (Hz)',"FontSize",14);
figure; plot(freq,Y_ph); title('Phase of Y('+string(char(937))+')',"FontSize",20);
ylabel('Phase(Y('+string(char(937))+'))',"FontSize",14); xlabel('Freq (Hz)',"FontSize",14);

sound(y,fs)

%----------------------------------------
%----------------------------------------
% Filtered Output (5th order poles & zeros)
Num = ((N1.*N2).^5).*((N3.*N4).^5);
Den = ((D1.*D2).^5).*((D3.*D4).^5);
H = Num./Den;
H_mag = abs(H)/(max(abs(H)));
figure; plot(OM,H_mag)

H_db = 20*log10(H_mag);
fq = OM*fs/(2*pi);
%figure; plot(fq,H_db);
figure; plot(OM,H_db);

%----------------------------------------
% Filtered Output (5th order poles & zeros)
y2 = filter(N,D,x);
y2 = filter(N,D,y2);
y2 = filter(N,D,y2);
y2 = filter(N,D,y2);
y2 = filter(N,D,y2); %5th order
[Y2, OM] = my_dtft(y2);

Y2_mag = abs(Y2);
Y2_ph = angle(Y2);
freq = OM*fs/(2*pi);

figure; plot(freq,Y2_mag); title('Mag of Y2('+string(char(937))+')',"FontSize",20);
ylabel('|Y('+string(char(937))+')|',"FontSize",14); xlabel('Freq (Hz)',"FontSize",14);
figure; plot(freq,Y2_ph); title('Phase of Y2('+string(char(937))+')',"FontSize",20);
ylabel('Phase(Y('+string(char(937))+'))',"FontSize",14); xlabel('Freq (Hz)',"FontSize",14);
sound(y2,fs)
```

A.2: <u>DTFT – "my_dtft" Function:</u>

```
function [X, OM] = my_dtft(x)

cnt=1;
for my_OM = 0:0.001:pi;
    my_x =0;

    for n = 1:length(x);
       my_x = my_x + x(n)*exp(-j*my_OM*n);
    end

   X(cnt) = my_x;
    OM(cnt) = my_OM;
   cnt = cnt+1;
end
end
```

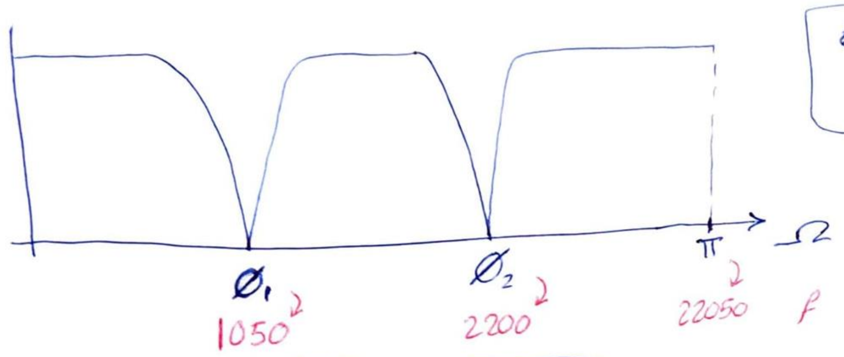
Appendix B: Hand Analysis

IIR filter design



$$\boxed{\begin{array}{l} * f_s = 44100 \longrightarrow 2\pi \\ \frac{f_s}{2} = 22050 \longrightarrow \pi \end{array}}$$

$$\boxed{\begin{array}{l} * \text{Noise Tones: } 1050 \text{ Hz} \\ \qquad \& \ 2200 \text{ Hz} \end{array}}$$



$22050 \longrightarrow \pi$
$1050 \longrightarrow \phi_1$

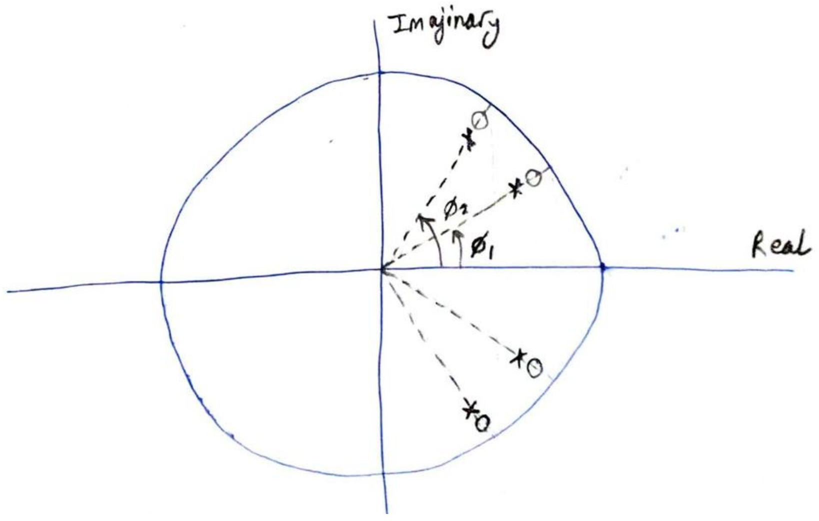
$$\phi_1 = \frac{1050}{22050}\pi \simeq 0.05\pi = 0.15 \text{ rad}$$

$$\phi_2 = \frac{2200}{22050}\pi \simeq 0.1\pi = 0.313 \text{ rad}$$

$$N(z) = (z - NC1\, e^{j\phi_1})(z - NC1\, e^{-j\phi_1}).(z - NC2\, e^{j\phi_2})(z - NC2\, e^{-j\phi_2})$$
$$= (z - 0.99\, e^{j0.15})(z - 0.99\, e^{-j0.15}).(z - 0.99\, e^{j0.313})(z - 0.99\, e^{-j0.313})$$

$$D(z) = (z - DC1\, e^{j\phi_1})(z - DC1\, e^{-j\phi_1}).(z - DC2\, e^{j\phi_2})(z - DC2\, e^{-j\phi_2})$$
$$= (z - 0.96\, e^{j0.15})(z - 0.96\, e^{-j0.15}).(z - 0.96\, e^{j0.313})(z - 0.96\, e^{-j0.313})$$

$$\boxed{H(z) = \frac{N(z)}{D(z)}}$$



References:

[1] A. Oppenheim and R. Schafer, “Discrete-Time Signal Processing”, 3rd Ed., Pearson, 2010.

[2] *MATLAB Signal Processing Toolbox* [Online], Available: https://www.mathworks.com/help/signal/index.html, Accessed: November 1, 2020.