International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Enhancing the Quality of Sound using Fast Fourier Transform in Python

Criton Jose Kallukkaran[1], Yashraj Morde[2]

[1, 2]MS Mechatronics, University of Glasgow, United Kingdom

Abstract: ***Audio information plays an important role in the increasing digital content that is available today, resulting in a need for methodologies that automatically analyse such content like music information retrieval, audio-visual analysis of online videos for content-based recommendation. This report presents the use of fast Fourier transform (FFT) to improve the quality of voice. Motivation: Sound is an essential part of life. While shooting a video and sound in an outdoor location, recording high quality audio can be difficult. Using FFT to increase the amplitudes of the harmonics to improve the quality of the voice and bringing it to a desired standard is essential.***

Keywords: fast Fourier transform, harmonics, fundamental frequencies, Noise reduction, Voice amplification

1. Introduction

Audio information plays an important role in the increasing digital content that is available today, resulting in a need for methodologies that automatically analyse such content like music information retrieval, audio-visual analysis of online videos for content-based recommendation. This report presents the use of fast Fourier transform (FFT) to improve the quality of voice.

Motivation: Sound is an essential part of life. While shooting a video and sound in an outdoor location, recording high quality audio can be difficult. Using FFT to increase the amplitudes of the harmonics to improve the quality of the voice and bringing it to a desired standard is essential.

2. SM58 microphone – Common vocal microphone

Harmonics enhancement in this report is done using SM58 microphone as a reference.

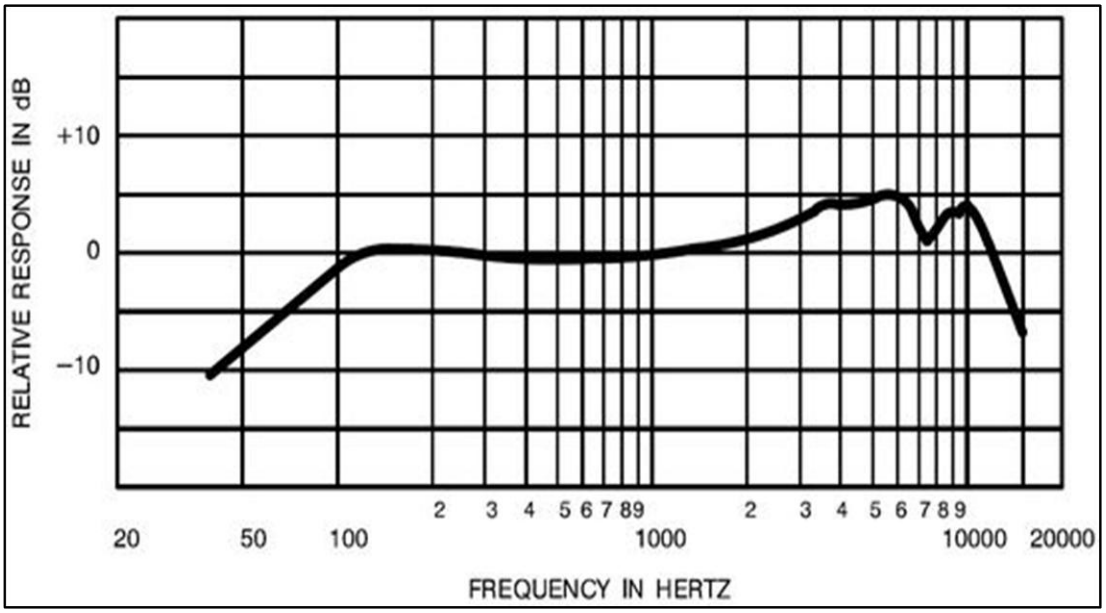

Figure 1: SM58 Frequency Response

Figure 1represents the frequency response for SM58 microphone. From the response graph frequencies below 100 Hz are attenuated. Frequencies between 100Hz and 1000Hz are taken as range of fundamental frequencies. Fundamental frequency is the lowest frequency in the signal. Harmonics are the boosted frequencies which lie between 1000Hz to 10,000Hz. Frequencies above 10000Hz are high frequency noise, hence the frequencies are attenuated again [0][[2]].

3. Results and Discussion

3.1 Time and Frequency domain – Recoded Audio

Complete Python code for voice enhancement is shown in

Appendix. The quality of voice is improved by using fast Fourier transform in Python. Audio sample is recorded at a sampling rate of 44kHz using a microphone. Multiple audio samples were recorded since certain audio samples were clipped which resulted in distortion on enhancement, hence the best audio sample was selected were noclipping was observed.

Recorded audio is plotted in time domain as shown in Figure 2.

t=np.linspace(0,len(data)/fs,len(data))where, len (data) represents the total number of samples from 0-start point to len(data)/fs – end point.

plt.plot(t,data)

t plotted on x-axis; data plotted on y-axis

Time domain is plotted for the total audio time [[3]].

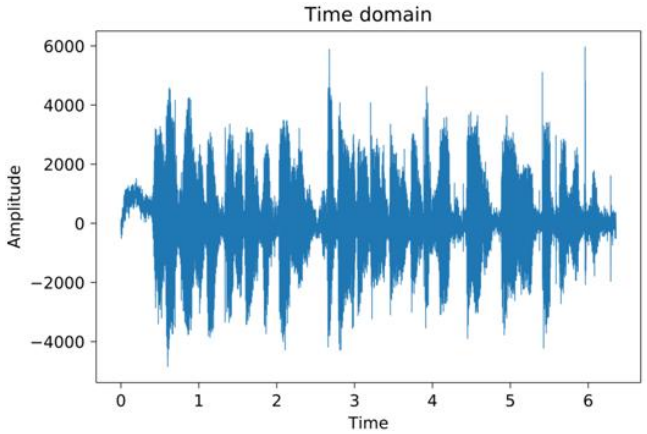

Figure 2: Original_Time domain

Frequency domain [[3]] is plotted as shown in Figure 3 to record the peak signals during the total period of observation. Red area highlights the range for fundamental frequencies (100Hz – 1000Hz) and yellow area highlights harmonics (1000Hz – 10000Hz) range of human voice. Generally, the first peak in the range of 100-1000Hz is considered as fundamental frequency, however there can be more than one fundamental frequencies in frequency spectrum.

dataf=np.fft.fft(data) # using FFT to convert into frequency domain

fd=abs(dataf) # considering absolute values

x=fd[0:int((len(fd)/2)-1)] # Half-samples consideration

plt.xscale('log') # log scale x-axis or frequency axis

faxis=np.linspace(0,fs/2,len(x)) # frequency axis

plt.plot(faxis,20*np.log10(x/len(data))) #amplitude in decibels

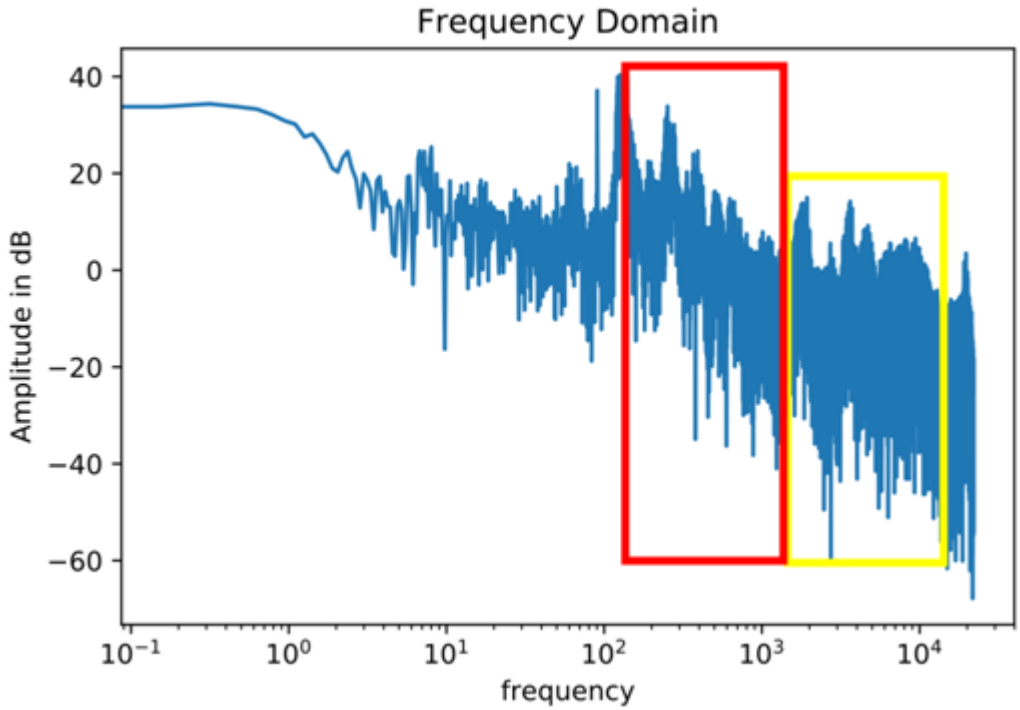

Figure 3: Original_Frequency Domain

3.2 Amplified Harmonics and Noise reduction

k1=int(len(dataf)/fs*1000) #index place for 1000Hz

k2=int(len(dataf)/fs*10000) #index place for 10000Hz

n1=int(len(dataf)/fs*0.1) #index place for 0.1Hz

n2=int(len(dataf)/fs*90) #index place for 90Hz

n3=int(len(dataf)/fs*10001) #index place for 10001Hz

```
n4=int(len(dataf)/fs*25000)                    #index place
for 25000Hz
dataf[n3:n4]=dataf[n3:n4]/60                   # Noise
reduction
dataf[int(len(dataf)-n4):int(len(dataf)-
n3)]=dataf[int(len(dataf)-n4):int(len(dataf)-n3)]/60
                                    # Mirroring to reduce noise
dataf[n1:n2]=dataf[n1:n2]/60
dataf[int(len(dataf)-n2):int(len(dataf)-
n1)]=dataf[int(len(dataf)-n2):int(len(dataf)-n1)]/60
                                    # Mirroring to reduce
noise
dataf[k1:k2]=dataf[k1:k2]*10                   # Voice
amplification
dataf[int(len(dataf)-k2):int(len(dataf)-
k1)]=dataf[int(len(dataf)-k2):int(len(dataf)-k1)]*10
plt.plot(faxis,20*np.log10(dataf[0:int(len(dataf)/2)-
1]/len(dataf)))
```

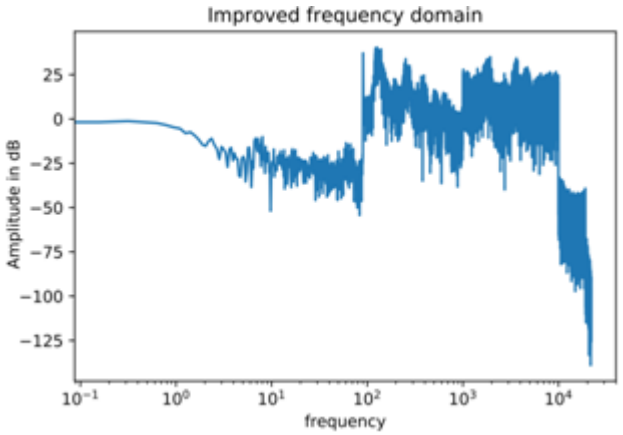

Figure 4: Harmonics Amplification and noise reduction

3.3 Time domain representation - Enhanced

```
enhanced=np.fft.ifft(dataf)        #Inverse fourier transform to
transform to time domain
clr=np.real(enhanced)              # Real part extraction
audio = clr.astype(np.int16)       #Convert to 16 bit data
plt.plot(t,clr)                    #Plotting in time domain
wavfile.write('improved.wav',fs,audio)#Saving enhanced
audio file
```

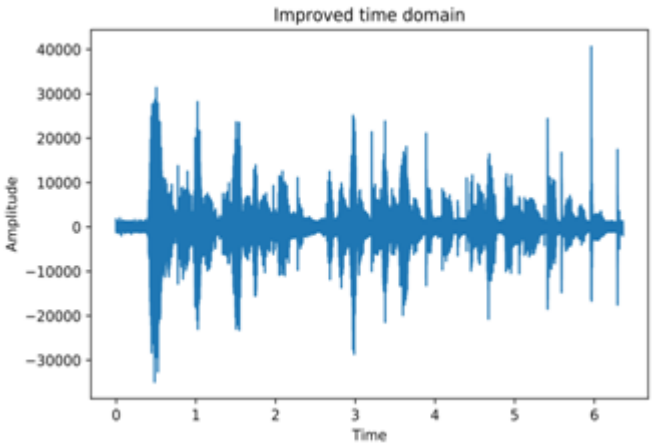

Figure 5: Enhanced voice sample

4. Conclusion

Audio was recorded at 44kHz sampling rate and the harmonics (1000Hz – 10000 Hz) were enhanced with a scaling factor of 10 in order to improve the quality of recorded voice. In order to get a similar frequency curve as SM58 microphone, frequency below 100 Hz (noise) and above 10001Hz (noise) where attenuated. The graphs were plotted in time and frequency domain. The enhanced audio sounded loud and clear as compared to the original recording, however some unwanted noise was heard.

References

[1] Comparing the Shure SM58 VS RODE NT-1A (Read This Before You Buy). (2019). Retrieved 27October 2019, from https://producerhive.com/buyer-guides/shure-sm58-vs-rode-nt-1a/
[2] (2019). Retrieved 27 October 2019, from https://www.scienceabc.com/pure-sciences/why-negative-decibels-are-a-thing.html
[3] University of Glasgow Moodle: Log in to the site. (2019). Retrieved 28 October 2019, from https://moodle.gla.ac.uk/course/view.php?id=18777

Appendix

```
import numpy as np                                  #importing libraries
import matplotlib.pyplot as plt                     #importing libraries
import scipy.io.wavfile as wavfile                  #importing libraries
fs,data=wavfile.read('original.wav')          #Reading recorded audio file
t=np.linspace(0,len(data)/fs,len(data))#sample to time
plt.figure(1)                                       #separate figure
plt.plot(t,data)                                #plotting in time domain
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Time domain')
dataf=np.fft.fft(data)                              #FFT function
fd=abs(dataf)                                   #Taking absolute values
x=fd[0:int((len(fd)/2)-1)]                      #Half Range
faxis=np.linspace(0,fs/2,len(x))                #Range frequency axis
plt.figure(2)                                       #separate figure
plt.xlabel('frequency')
plt.ylabel('Amplitude in dB')
plt.title('Frequency Domain')
```

```
plt.xscale('log')                    #plotting x-axis in logarithmic axis
plt.plot(faxis,20*np.log10(x/len(data)))# plotting frequency axis vs amplitude(dB)
k1=int(len(dataf)/fs*1000)                    #index place for 1000Hz
k2=int(len(dataf)/fs*10000)                        #index place for 10000Hz
n1=int(len(dataf)/fs*0.1)                    #index place for 0.1Hz
n2=int(len(dataf)/fs*90)                    #index place for 90Hz
n3=int(len(dataf)/fs*10001)                        #index place for 1001Hz
n4=int(len(dataf)/fs*25000)                        #index place for 25000Hz
dataf[n3:n4]=dataf[n3:n4]/60                    # Noise reduction
dataf[int(len(dataf)-n4):int(len(dataf)-n3)]=dataf[int(len(dataf)-n4):int(len(dataf)-n3)]/60
dataf[n1:n2]=dataf[n1:n2]/60
dataf[int(len(dataf)-n2):int(len(dataf)-n1)]=dataf[int(len(dataf)-n2):int(len(dataf)-n1)]/60
dataf[k1:k2]=dataf[k1:k2]*10                    # hamonics amplification
dataf[int(len(dataf)-k2):int(len(dataf)-k1)]=dataf[int(len(dataf)-k2):int(len(dataf)-k1)]*10
plt.figure(3)                                    # separate figure
plt.plot(faxis,20*np.log10(dataf[0:int(len(dataf)/2)-1]/len(dataf))# frequency vs amplitude(db)
plt.xlabel('frequency')
plt.ylabel('Amplitude in dB')
plt.title('Improved frequency domain')
plt.xscale('log')                    #plotting x-axis in logarithmic axis
enhanced=np.fft.ifft(dataf)                    #IFFT function transform to time domain
clr=np.real(enhanced)                    # Real part extraction
audio = clr.astype(np.int16)                    #Convert to 16 bit data
plt.figure(4)                    #separate figure
plt.plot(t,clr)                    #plotting in time domain
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Improved time domain')
plt.show()
wavfile.write('improved.wav',fs,audio)        #writing enhanced audio file
```