PROJECT REPORT
ON

# ECL301 : DIGITAL SIGNAL PROCESSING

## "Real-Time Audio Data Maneuvering"

**Submitted to**
**IIIT NAGPUR**

**Under the Guidance of**
**Dr. ANKIT BHURANE**

**DEPARTMENT OF**
**ELECTRONICS AND COMMUNICATION ENGINEERING**
**IIIT NAGPUR**
**2019-2020**


By

BT18ECE025 ANIRUDDHA GAWATE
BT18ECE026 ANUJ LATHI
BT18ECE030 SHRIJEET KHARCHE

# Table of Contents

# 1 INTRODUCTION

## 1.1 OBJECTIVE

The main objective is to perform certain functions on the data acquired on real time basis , the function comprises of filtering , plotting , certain graphical outputs and determining continuously the value of input frequency. Also keeping into the consideration all the function should be performed on Real-time computation.

## 1.2 ABSTRACT

With the development of telecommunication technology over the last decades, the request for digital information compression has increased dramatically. In many applications, such as high-quality audio transmission and storage, the target is to achieve audio and speech signal codings at the lowest possible data rates, in order to offer cheaper costs in terms of transmission and storage the real-time computing concept comes forward. A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness but also upon the time in which it is performed. In a real-time digital signal processing (DSP) process, the analyzed (input) and generated (output) samples can be processed (or generated) continuously in the time it takes to input and output the same set of samples independent of the processing delay. It means that the processing delay must be bounded even if the processing continues for an unlimited time.

The "Real-Time Audio Data Manoeuvring" divided into 3 phases, which carefully guides in order to achieve the desired output. Real-time audio data collection and their specific plot visualization is the primary goal. The secondary goal is to determine input frequency (peak) to help filtering certain frequencies along with the time required.

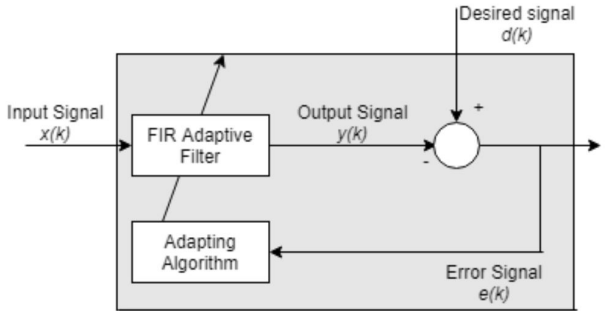
## 2 METHODOLOGY

**Phase I : Real-Time audio visualization** is a phase in which we basically get the input from the microphone mimicking the process of vocal communication with the system. In this the inputs is collected in an array which then with the help of suitable libraries to plot the real-time audio data graph.

As seen, the plot is displayed continuously on the web page. The plot updates every second. The code for web page (HTML) is CODE 1.2

We have also performed the graphical representation of symbol (#) (fig 1) just to have an different way to express the output and it also resembles to the amplifiers display.

**Phase II : Real-Time audio Maneuvering** Adaptive filters are digital filters whose coefficients change with an objective to make the filter converge to an optimal state. As the filter adapts its coefficients, the mean square error (MSE) converges to its minimal value. At this state, the filter is adapted and the coefficients have converged to a solution. The filter output, y(k), is then said to match very closely to the desired signal, d(k). When you change the input data characteristics, sometimes called filter environment, the filter adapts to the new environment by generating a new set of coefficients for the new data.




First goal was to get the frequency of the input signal so the system gets information about the frequency value getting collected in array and plotted. We can see the output below along with the time taken the process to happen(fig 3). This is the first part of the final code(CODE 2.1). The code performs the function of getting the frequency value of input signal to the system.

Then we applied the low pass filter along with the value of frequency as input and the low pass filter has made a different function below to smooth the process.

Now the process goes on, in for loop to plot all the real-time data continuously and save the output plot files in the folder named (filtered_images). Here, the plot

does not go to HTML page but saved for future applications.

Then the code goes on to make video out of all this saved plots files , so we get a video of all the plots in *.avi* format. As the number of signal plots is large and makes difficult to observe and spot changes. A video of input signals makes it easy to compare that the signal definitely had positive changes towards our goal.

**Process :**

To see the changes happen

Processed Part

1. CODE 2.3 should be performed giving the system a specific input .
2. Part (i) of code will determine the frequency value of input signal
3. Part (ii) of code will filter the frequency above and saving the plots.
4. Part (iii) of code will compile all the saved images in a folder named filtered_images converted to video.

Now, video we get is filtered_video.

Original Part

1. Now going for CODE 2.2 here part (i) will determine the frequency value of input signal.
2. Part (ii) of code will compile all the saved images in a folder named unfiltered_images converted to video.

Now, video we get is original_video.

Now, we have acquired two videos original_video and filtered_video, comparison is done side by side Ref. fig 4 and difference can be seen clearly. For the whole comparison both the videos are uploaded to drive link provided at last so the whole observation can be seen and observed properly.

**Phase III :** After the completion of above steps, the derived system can be taken forward in many other applications. Such as,

1. Noise or Interference Cancellation — Using an Adaptive Filter to Remove Noise from an Unknown System
2. Prediction — Predicting Future Values of a Periodic Signal
3. Inverse System Identification — Determining an Inverse Response to an Unknown System
4. Merging the derived system with encryption techniques.

(a) Steganography

(b) Watermarking

5. The collaboration of this technique with online conferences can reduce the data stream is a much effective way.
6. This process can further be extended to develop a virtual environment with the help of digital signal processing such that, the real time audio operations can be performed.

## 3 CODES

CODE 1.1 (Phase 1 : Real-Time Audio Visualization)

```
# Initialization of Libaraies and Variables.
import pyaudio as pad
import numpy as np
import pylab as plb
import time as t

Fs = 44100
F = int(Fs/20)
FORMAT = pad.paInt16
"""
Here,
    'F' variable specifies the number of frames per buffer.
    'Fs' variable defines time resolution of the recording device (Hz) / Sampling Frequency.
    'FORMAT' variable defines sampling size and format.
"""

# Function defination to display the plot.
def soundplot(Audio_in):
    t1=t.time()
    data = np.fromstring(Audio_in.read(F),dtype=np.int16)
    plb.plot(data)
    plb.title(i)
    plb.grid()
    plb.axis([0,len(data),-2**15,2**15])
    plb.savefig("03.png",dpi=50)
    plb.close('all')
    print("took %.02f ms"%((t.time()-t1)*1000))

# Main Code

if __name__=="__main__":
    s=pad.PyAudio()
```

```
33      Audio_in=s.open(format=FORMAT,channels=1,rate=Fs,input
          =True,
34                      frames_per_buffer=F)
35      for i in range(int(20*Fs/F)): #do this for 10 seconds
36          soundplot(Audio_in)
37      Audio_in.stop_stream()
38      Audio_in.close()
39      s.terminate()
```

CODE 1.2 (HTML Page)

```
1 <html>
2 <script language="javascript">
3 function RefreshImage(){
4 document.pic0.src="03.png?a=" + String(Math.random()
    *99999999);
5 setTimeout('RefreshImage()',50);
6 }
7 </script>
8 <body onload="RefreshImage()">
9 <img name="pic0" src="03.png">
10 </body>
11 </html>
```

CODE 2.1 (Real-time Frequency Determiner)

```
1 import pyaudio
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 np.set_printoptions(suppress=True) # don't use scientific
    notation
6
7
8 F = 4096 # number of data points to read at a time
9 Fs = 44100 # time resolution of the recording device (Hz)
10
```

```
p=pyaudio.PyAudio() # start the PyAudio class
stream=p.open(format=pyaudio.paInt16,channels=1,rate=Fs,
    input=True,
                frames_per_buffer=F) #uses default input
                   device

# create a numpy array holding a single read of audio data
for i in range(100): #to it a few times just to see
    data = np.fromstring(stream.read(F),dtype=np.int16)
    data = data * np.hanning(len(data)) # smooth the FFT
       by windowing data
    fft = abs(np.fft.fft(data).real)
    fft = fft[:int(len(fft)/2)] # keep only first half
    freq = np.fft.fftfreq(F,1.0/Fs)
    freq = freq[:int(len(freq)/2)] # keep only first half
    freqPeak = freq[np.where(fft==np.max(fft))[0][0]]+1
    print("peak frequency: %d Hz"%freqPeak)

    # uncomment this if you want to see what the freq vs
       FFT looks like
    #plt.plot(freq,fft)
    #plt.axis([0,4000,None,None])
    #plt.show()
    #plt.close()

# close the stream gracefully
stream.stop_stream()
stream.close()
p.terminate()
```

CODE 2.2 (Phase 2 : Real-Time Audio "ORIGINAL")

```
import pyaudio as pad
import numpy as np
import pylab as plb
import time as t
#import wavio
from scipy.signal import butter, filtfilt, iirnotch,
```

```
    savgol_filter
import cv2
import os


Fs = 44100
F = int(Fs/20)
FORMAT = pad.paInt16
"""
Here,
    'F' variable specifies the number of frames per buffer.
    'Fs' variable defines time resolution of the recording device (Hz) / Sampling Frequency.
    'FORMAT' variable defines sampling size and format.
"""
def soundplot(Audio_in):
    t1=t.time()
    data = np.fromstring(Audio_in.read(F),dtype=np.int16)
    plb.figure(i)
    plb.plot(data)
    plb.title(i)
    plb.grid()
    plb.axis([0,len(data),-2**16/2,2**16/2])
    for v in plb.get_fignums():
            plb.figure(v)
            plb.savefig('zfigure%d.png' % v)
            img = cv2.imread('zfigure%d.png' % v)
            path = 'unfiltered_images'
            cv2.imwrite(os.path.join(path , 'zfigure%d.png') %v,img)
            cv2.waitKey(0)
    plb.close('all')
    image_folder = 'unfiltered_images'
    video_name = 'unfilter_video.avi'

    images = [img for img in os.listdir(image_folder) if img.endswith(".png")]
```

```
40      frame = cv2.imread(os.path.join(image_folder, images
            [0]))
41      height, width, layers = frame.shape
42
43      video = cv2.VideoWriter(video_name, 0, 1, (width,
            height))
44
45      for image in images:
46          video.write(cv2.imread(os.path.join(image_folder,
                image)))
47
48      cv2.destroyAllWindows()
49      video.release()
50      print("took %.02f ms"%((t.time()-t1)*1000))
51        #for a in range(10): #to it a few times just to see
52
53
54 # Main Code
55 if __name__=="__main__":
56      s=pad.PyAudio()
57      Audio_in=s.open(format=FORMAT,channels=1,rate=Fs,input
            =True,
58                      frames_per_buffer=F)
59      for i in range(int(10*Fs/F)): #do this for 10 seconds
60          soundplot(Audio_in)
61
62
63      Audio_in.stop_stream()
64      Audio_in.close()
65      s.terminate()
```

CODE 2.3 (Phase 2 : Real-Time Audio "PROCESSED")

```
1 # Initialization of Libaraies and Variables.
2 import pyaudio as pad
3 import numpy as np
4 import pylab as plb
5 import time as t
```

```
6 import wavio
7 from scipy.signal import butter, filtfilt, iirnotch,
    savgol_filter
8 import cv2
9 import os
10
11
12 Fs = 44100
13 F = int(Fs/20)
14 FORMAT = pad.paInt16
15 """
16 Here,
17     'F' variable specifies the number of frames per buffer
       .
18     'Fs' variable defines time resolution of the recording
         device (Hz) / Sampling Frequency.
19     'FORMAT' variable defines sampling size and format.
20 """
21
22 # Function defination to display the plot.
23 def soundplot(Audio_in):
24     t1=t.time()
25     data = np.fromstring(Audio_in.read(F),dtype=np.int16)
26
27     data = data * np.hanning(len(data)) # smooth the FFT
       by windowing data
28     fft = abs(np.fft.fft(data).real)
29     fft = fft[:int(len(fft)/2)] # keep only first half
30     freq = np.fft.fftfreq(F,1.0/Fs)
31     freq = freq[:int(len(freq)/2)] # keep only first half
32     freqPeak = freq[np.where(fft==np.max(fft))[0][0]]+1
33     print("peak frequency: %d Hz"%freqPeak)
34     if(freqPeak >= 1300):
35         butter_lowpass(1300,F,data)
36         #wavio.write("recorded.wav",data,F,sampwidth=2)
37         plb.figure(i)
38         plb.plot(data)
39         plb.title(i)
```

```
40            plb.grid()
41            plb.axis([0,len(data),-2**16/2,2**16/2])
42          # plb.savefig("03.png",dpi=50)
43            for v in plb.get_fignums():
44                plb.figure(v)
45                plb.savefig('zfigure%d.png' % v)
46                img = cv2.imread('zfigure%d.png' % v)
47                path = 'filtered_images'
48                cv2.imwrite(os.path.join(path , 'zfigure%d.png
                      ') %v,img)
49                cv2.waitKey(0)
50            plb.close('all')
51            print("took %.02f ms"%((t.time()-t1)*1000))
52
53      else:
54          plb.figure(i)
55          fig=plb.plot(data)
56          plb.title(i)
57          plb.grid()
58          plb.axis([0,len(data),-2**16/2,2**16/2])
59          for v in plb.get_fignums():
60            plb.figure(v)
61            plb.savefig('zfigure%d.png' % v)
62            img = cv2.imread('zfigure%d.png' % v)
63            path = 'filtered_images'
64            cv2.imwrite(os.path.join(path , 'zfigure%d.png')
                  %v,img)
65            cv2.waitKey(0)
66          plb.close('all')
67          image_folder = 'filtered_images'
68          video_name = 'filter_video.avi'
69
70          images = [img for img in os.listdir(image_folder)
              if img.endswith(".png")]
71          frame = cv2.imread(os.path.join(image_folder,
              images[0]))
72          height, width, layers = frame.shape
73
```

```
74          video = cv2.VideoWriter(video_name, 0, 1, (width,
              height))
75
76          for image in images:
77              video.write(cv2.imread(os.path.join(
                  image_folder, image)))
78
79          cv2.destroyAllWindows()
80          video.release()
81          print("took %.02f ms"%((t.time()-t1)*1000))
82          #for a in range(10): #to it a few times just to
              see
83
84
85
86 def butter_lowpass(cutoff, sample_rate, data,order=6):
87  """
88      Parameters
89      ----------
90      cutoff : int or float
91          frequency in Hz that acts as cutoff for filter.
92          All frequencies below cutoff are filtered out.
93
94      sample_rate : int or float
95          sample rate of the supplied signal
96      order : int
97          filter order, defines the strength of the roll-off
98          around the cutoff frequency. Typically orders
              above 6
99          are not used frequently.
100         default : 2
101     Returns
102     --------
103     out : tuple
104         numerator and denominator (b, a) polynomials.
105     """
106  nyq = 0.5 * sample_rate
107  normal_cutoff = cutoff / nyq
```

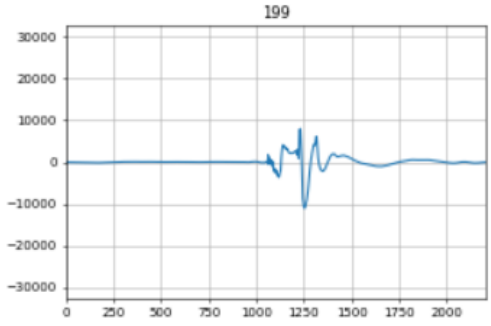
```
108  b, a = butter(order, normal_cutoff, btype='low', analog=
         True)
109  data = filtfilt(b, a, data)
110  print("filtered")
111  return data
112
113
114 # Main Code
115 if __name__=="__main__":
116      s=pad.PyAudio()
117      Audio_in=s.open(format=FORMAT,channels=1,rate=Fs,input
           =True,
118                      frames_per_buffer=F)
119      for i in range(int(10*Fs/F)): #do this for 10 seconds
120          soundplot(Audio_in)
121
122
123      Audio_in.stop_stream()
124      Audio_in.close()
125      s.terminate()
```

# 4 OUTPUT / RESULT

```
0000 00000
0001 00000
0002 00000
0003 00151
0004 00232
0005 00277
0006 00356
0007 00436
0008 00308
0009 00335
0010 00313
0011 00375
0012 00344
0013 00357
0014 03657 ##
0015 04698 ###
0016 05396 ####
0017 04285 ###
0018 02424 #
0019 01255
0020 01048
0021 00964
0022 00330
0023 00320
0408 02782 ##
0409 03582 ##
0410 04042 ###
0411 03349 ##
0412 01599 #
0413 02828 ##
0414 04663 ###
0415 04008 ###
0416 03031 ##
0417 03538 ##
0418 04184 ###
0419 04344 ###
0420 03838 ##
0421 01725 #
0422 01536 #
0423 02843 ##
0424 02710 ##
0425 02248 #
0426 03392 ##
0427 04434 ###
0428 04568 ###
0429 05093 ###
```

Figure 1: OUTPUT 1.1 (Phase 1 : Real-Time Audio Visualization)

OR




```
D:\Python\Test.py:20: DeprecationWarning: The binary mode of fromstring is deprecated,
as it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 321.57 ms
D:\Python\Test.py:20: DeprecationWarning: The binary mode of fromstring is deprecated,
as it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 379.46 ms
D:\Python\Test.py:20: DeprecationWarning: The binary mode of fromstring is deprecated,
as it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 271.46 ms
D:\Python\Test.py:20: DeprecationWarning: The binary mode of fromstring is deprecated,
as it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 207.92 ms
D:\Python\Test.py:20: DeprecationWarning: The binary mode of fromstring is deprecated,
as it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 333.10 ms
```

Figure 2: OUTPUT 1.1 (Phase 1 : Real-Time Audio Visualization)

```
D:\Python\end_testing.py:25: DeprecationWarning: The binary mode of fromstring is deprecated, as
it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 441.14 ms
peak frequency: 1321 Hz
filtered
D:\Python\end_testing.py:25: DeprecationWarning: The binary mode of fromstring is deprecated, as
it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 400.98 ms
peak frequency: 1081 Hz
D:\Python\end_testing.py:25: DeprecationWarning: The binary mode of fromstring is deprecated, as
it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 329.83 ms
peak frequency: 821 Hz
D:\Python\end_testing.py:25: DeprecationWarning: The binary mode of fromstring is deprecated, as
it behaves surprisingly on unicode inputs. Use frombuffer instead
  data = np.fromstring(Audio_in.read(F),dtype=np.int16)
took 1270.66 ms
peak frequency: 1061 Hz
```

Figure 3: Real-time Frequency Determiner

Unfiltered Sample Image | Filtered Sample Image

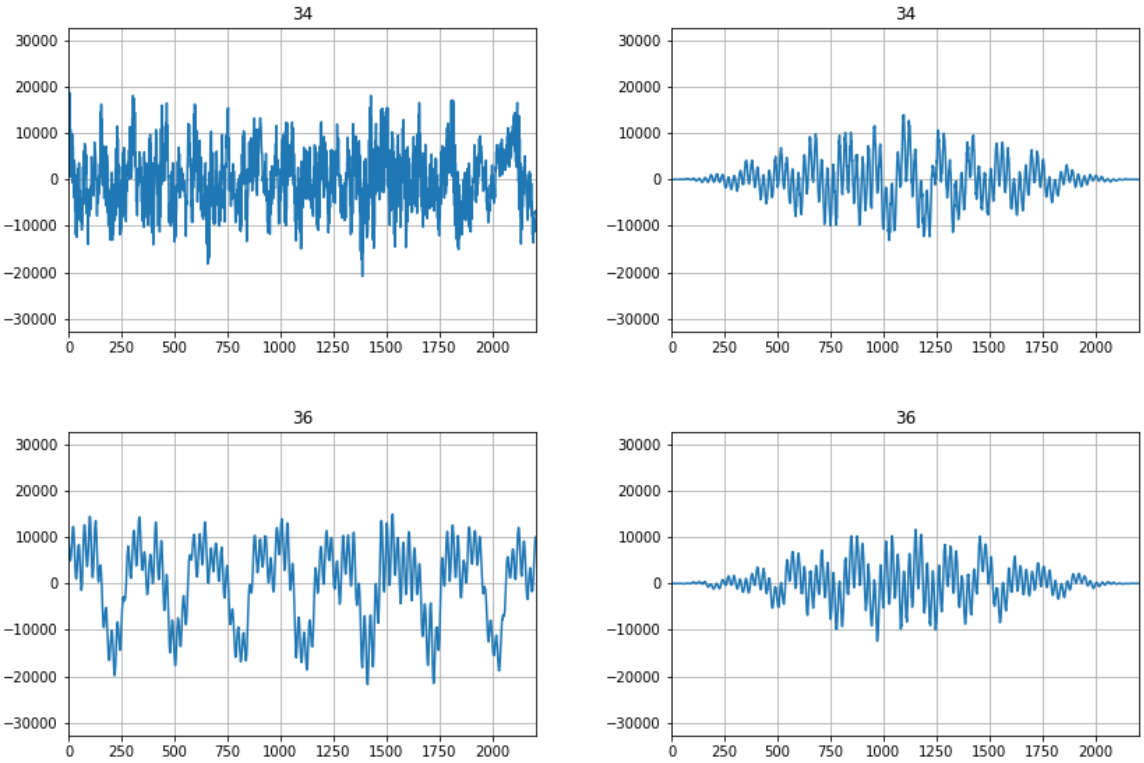



Figure 4: OUTPUT 2.1 (Phase 2 : Real-time Audio Filtering)

## 5 OBSERVATION

**Phase I :**

1. For the phase 1 case of testing, real time basis input output can be seen. As peak value exceeds certain frequency limits disturbance is observed.The observed disturbance is shown with symbol(#) to give a different look. (Ref. Figure 1)
2. To make the better visualization, the Real-Time plot of audio can be observed on the HTML page. Here, the plot of the input signal can be seen in real time which means it updates continuously as per the input. The python console also shows the time taken to plot. (Ref. Figure 2)

**Phase II :**

1. The frequency of the input signal is determined continuously and printed(Ref. Figure 3). Peak frequencies of particular intervals can be seen.
2. The filtering is applied immediately on the current real-time input and there is less amount of delay is observed with respect to simple audio processing.(Here, the delay is observed with the help of 'timestamp').
3. All the input value plots are saved into images then converted to video for better observation.
4. The phase 2 has two outputs from two codes, original and another processed. The processed is the "filtered_video". The two videos can be compared to see the maneuvering in the real time data.

---

*__Note :__ To check the descriptive video of the project i.e. to observe the difference between the original real-time input and the processed output, please click on the below hyperlink.

*Real-Time Audio Data Maneuvering*

---

## 6 CONCLUSION

The Real-time audio data maneuvering has been implemented and it can be used for further applications like Noise Cancellation, Encryption, etc.

# 7 REFERENCE


- PyAudio Documentation
  `https://people.csail.mit.edu/hubert/pyaudio/docs/#pyaudio.PyAudio.open`
- Adaptive Filter
  `https://pypi.org/project/adaptfilt/`
- Python Documentation
  `https://www.python-course.eu/index.php`
- Adaptive Filter Research
  `https://www.mathworks.com/help/dsp/ug/overview-of-adaptive-filters\-and-applications.html`