INTERNSHIP REPORT

TOPIC: Writing OS in Assembly language, Audio Processing

Sl.no	NAME	SCHOLAR NUMBER
1.	P V V S S Santosh Lahari	171114095
2.	Pragada Sivani	171114127
3.	Mansi Singh	171114093

WRITING OS IN ASSEMBLY LANGUAGE

Operating system is a system software that manages and helps in the communication path from user to the hardware. It provides the common services for the computer programs. NASM(netwide assembler) is an assembler and disassembler for 8086 microprocessors. As the PC starts it starts through the boot of floppy or the CD-ROM. The .asm(assembly language file) file consisting of the booting code needs to be converted to .flp or .iso. This was achieved with the help of NASM. The OS was tested with the help of VirtualBox software, which helps in the booting of OS virtually to the PC and execute the services provided.

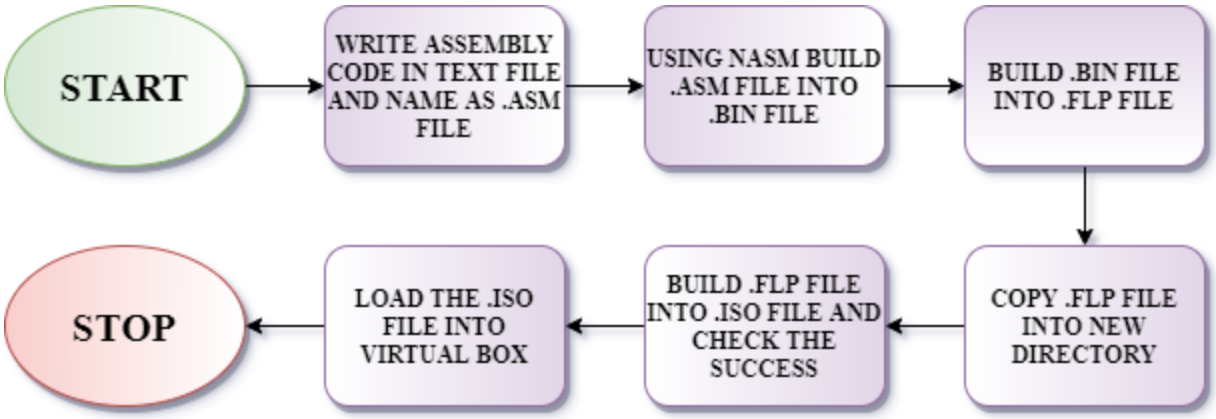
With MikeOS[1] as a stepping platform, it helped us in the understanding of the procedure of programming an OS from scratch. The code written on the first sector of the floppy disc is the bootloader or the bootstrap. Bootstrap is the program that initializes the OS during the startup of the device. The bootloader initialises the RAM, the buffer in which the OS is loaded and it further instructs in the execution of the sectors of the floppy. Then it proceeds to execute the kernel.bin. Kernel.bin contains the OS commands and this helps in the calling and running the services, other programs. Kernel.bin is where the interface is coded and the colors of the dialog boxes, giving the user an opportunity to select the required application that they want to access. It has all the files included in it that might be called and which can be executed, which are .bin or .bas files. The .bin and .bas files have specific instructions in them to perform calculations or run a game etc.

Worked on basic OS which prints basic strings upon booting and the understanding of the working of the mikeOS and how it can be edited. The edit is to be done in the kernel.bin file, with the addition of the new application that is to be included in the OS, this updated kernel.bin

[1] "MikeOS - SourceForge." http://mikeos.sourceforge.net/. Accessed 24 Jul. 2020.

is to be copied to the floppy. The rest of the memory segments, initializing can be kept similar to mikeOS.

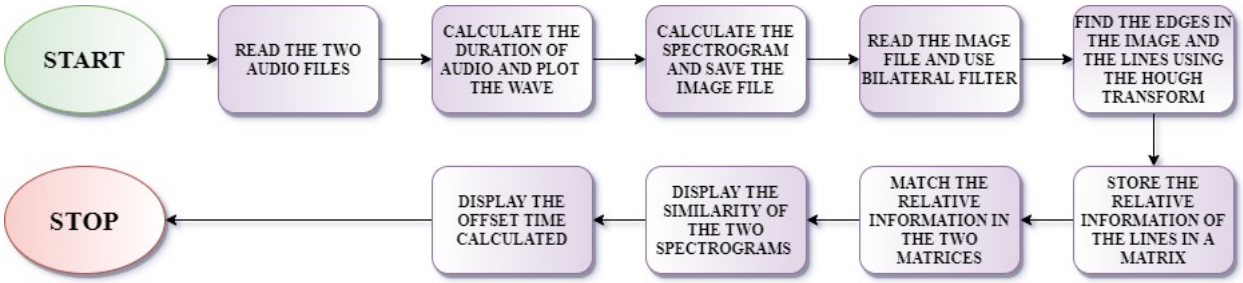
BLOCK DIAGRAM



AUDIO PROCESSING

Audio fingerprinting[2] is one of the major applications in audio recognition. Shazam is one of the algorithms that is used in audio fingerprinting. The Algorithm involves calculating the spectrogram of an audio file. Subsequently locating the peaks in the spectrogram of the audio because the peaks are least affected by noise. Then proceed to hash the information of the peaks to get a robust.

BLOCK DIAGRAM



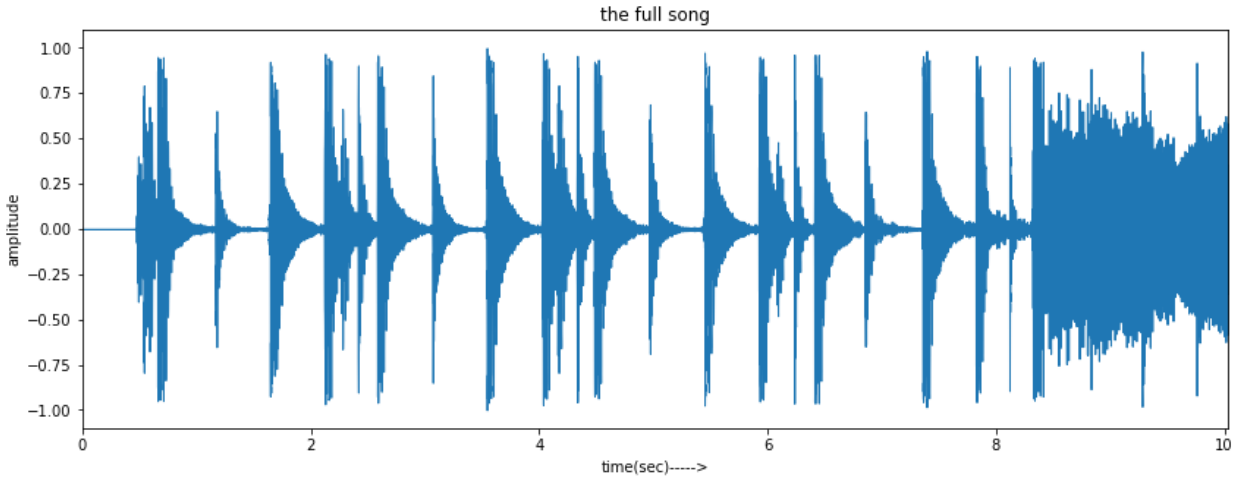
LOCAL PEAK IDENTIFICATION

We found the approach of taking local peaks in the spectrogram didn't give us the same peaks when we took the audio with offset in time. We did not take a recorded as in the later approach, we took an audio and gave offset in time compared to the original audio.

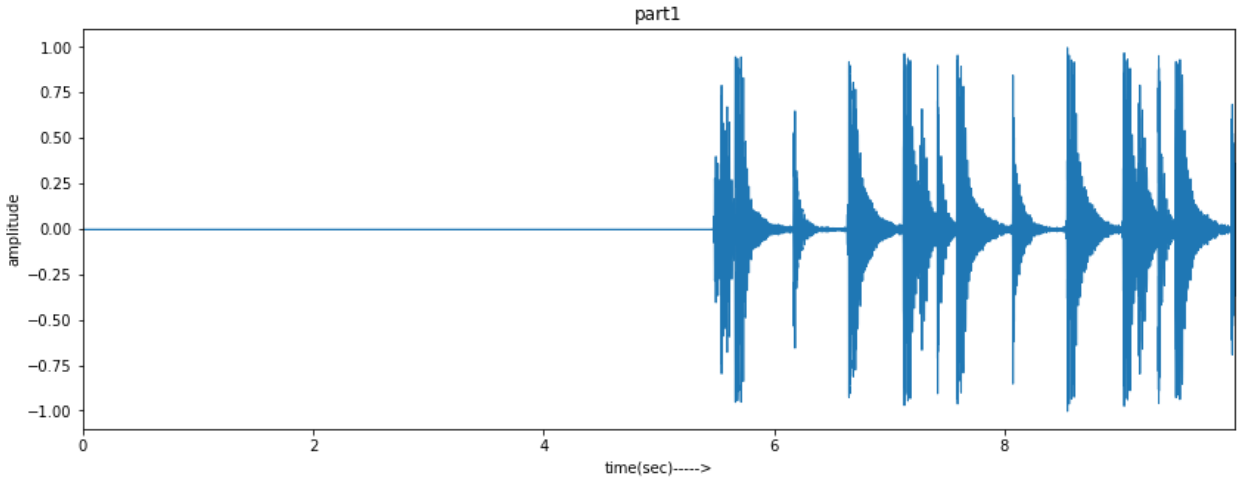
[2] "Audio Fingerprinting with Python and Numpy - Will Drevo." 15 Nov. 2013, https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/. Accessed 24 Jul. 2020.

Finding the local peak in a spectrogram, we treat the spectrogram as an image and try to get the location of the pixel which is the local maxima[3].

The original audio wave plot.

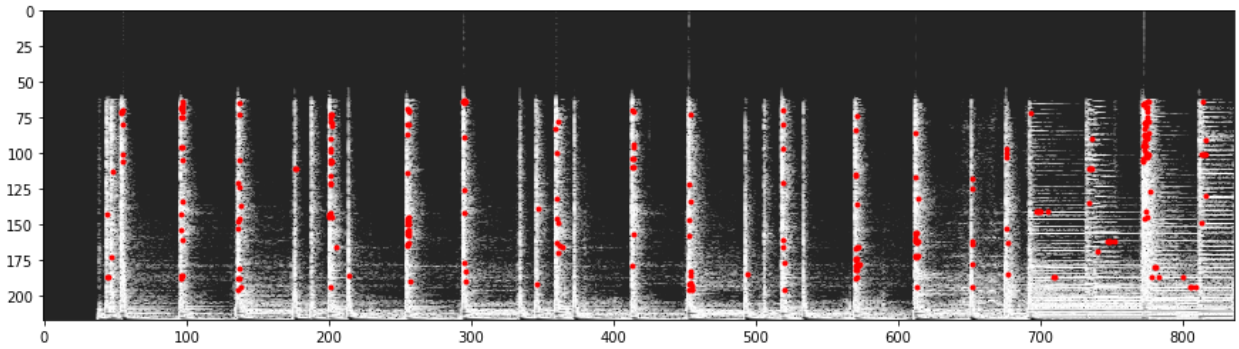


Wave plot of the audio with offset in time.

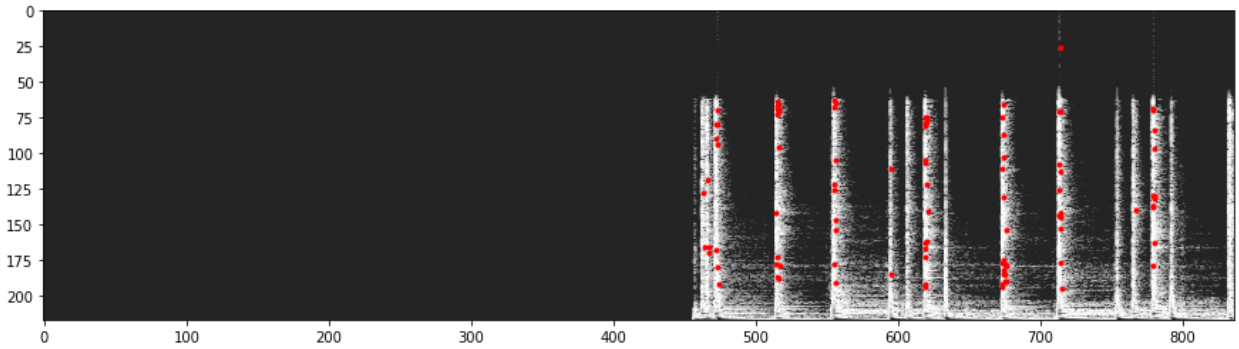


Local maxima of the original audio with no offset.

[3] "Finding local maxima — skimage v0.17.2 docs - Scikit-image." https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_peak_local_max.html. Accessed 24 Jul. 2020.



Local maxima of the audio with offset.



Just from the images we can tell that the local maxima or the peaks that are found in both the spectrograms are not in the similar locations. Hence when we proceed to match the spectrograms by taking relative information the results are not good. Hence we opted for the method described below.

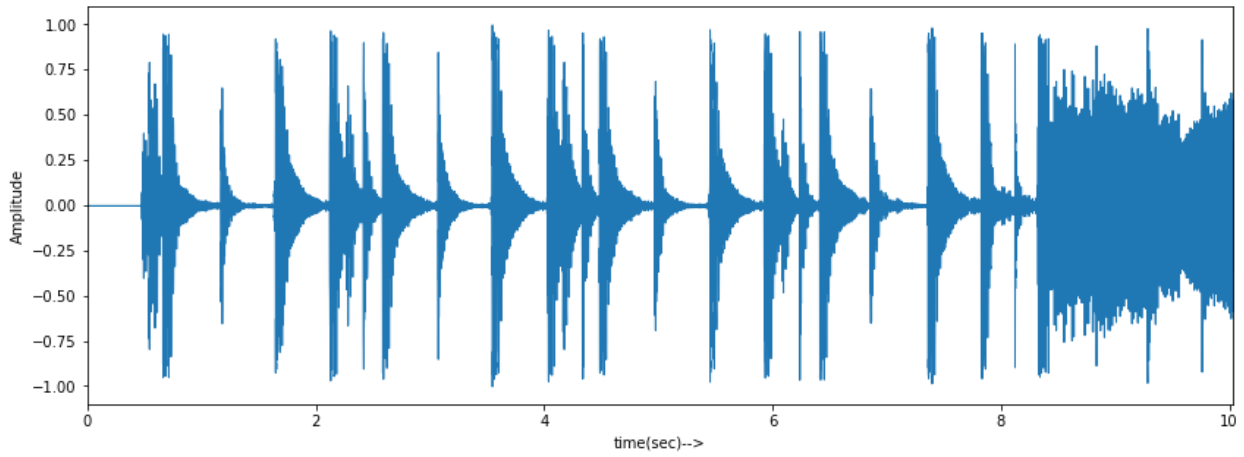
HOUGH TRANSFORM

Since finding the local peaks was not a very fruitful solution we opted to find the hough transform[4] of the spectrogram. We find the hough lines in the spectrogram. Taking all those lines we find the hash of each line and store it in a matrix. The main purpose of taking the hough transform is to compare two spectrograms and see the similarities in the spectrogram.

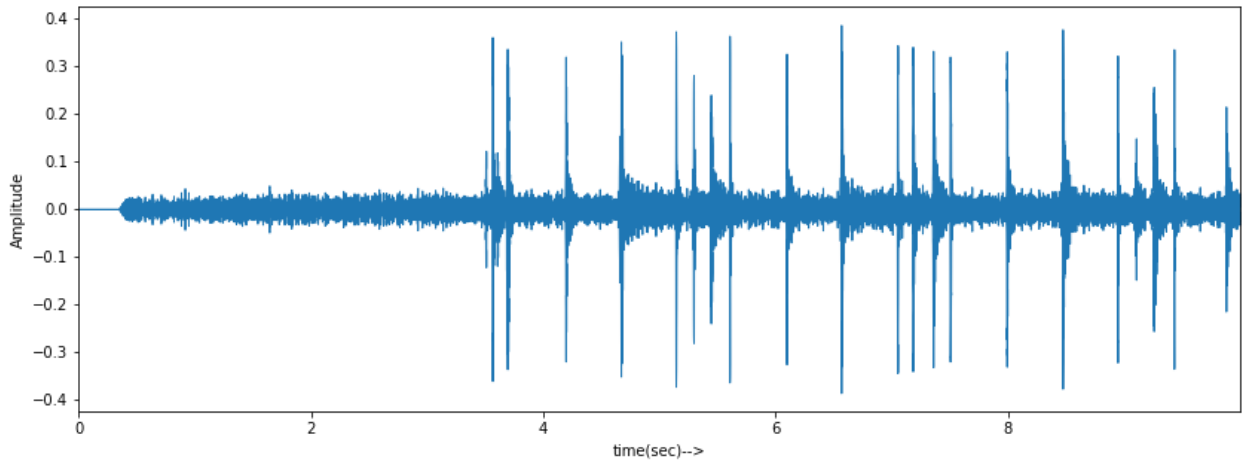
Hough transform helps us in finding the lines in the spectrogram with the help of voting procedure. Once we get the lines we have the information of the lines on the time axis and we store the relative information of the lines in a matrix. We get two such matrices, then we match the two matrices and see the similarity among the two spectrograms. We can then calculate the offset in the time of the recorded audio.

Wave plot of the original audio.

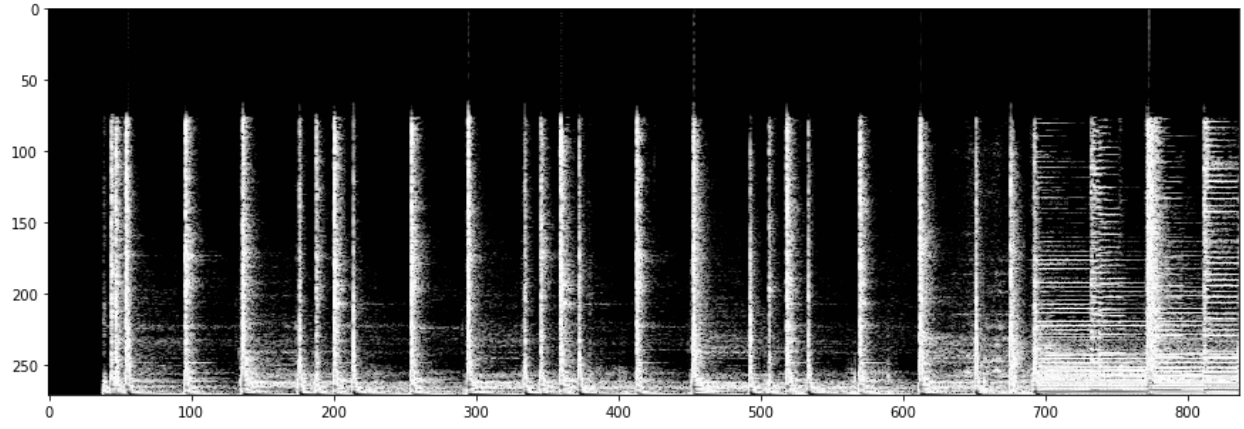
[4] "Compare two spectogram to find the offset where they match ...." 14 Apr. 2011, https://stackoverflow.com/questions/5651725/compare-two-spectogram-to-find-the-offset-where-they-match-algorithm. Accessed 24 Jul. 2020.



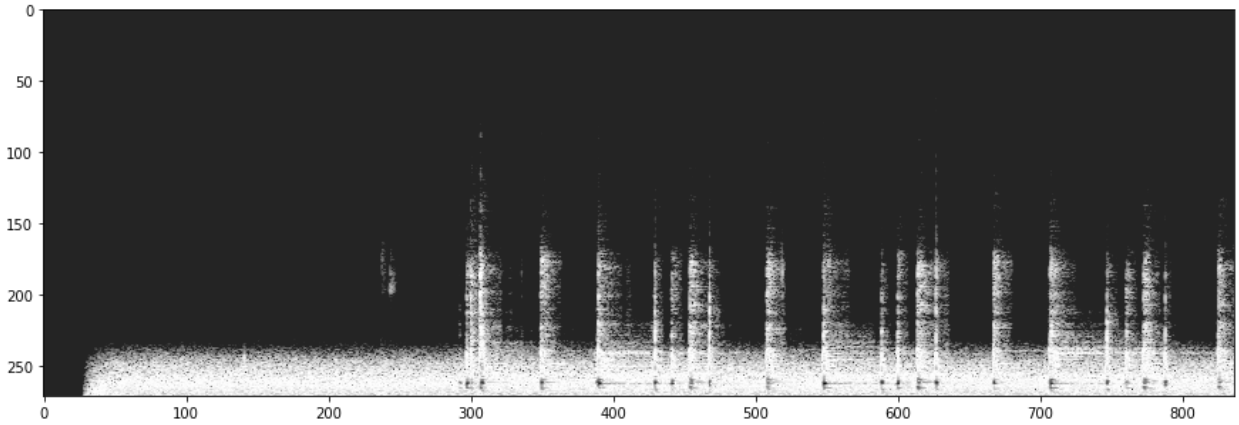
Wave plot of the phone recorded audio of the same audio as the original. As we can see there is offset in time.



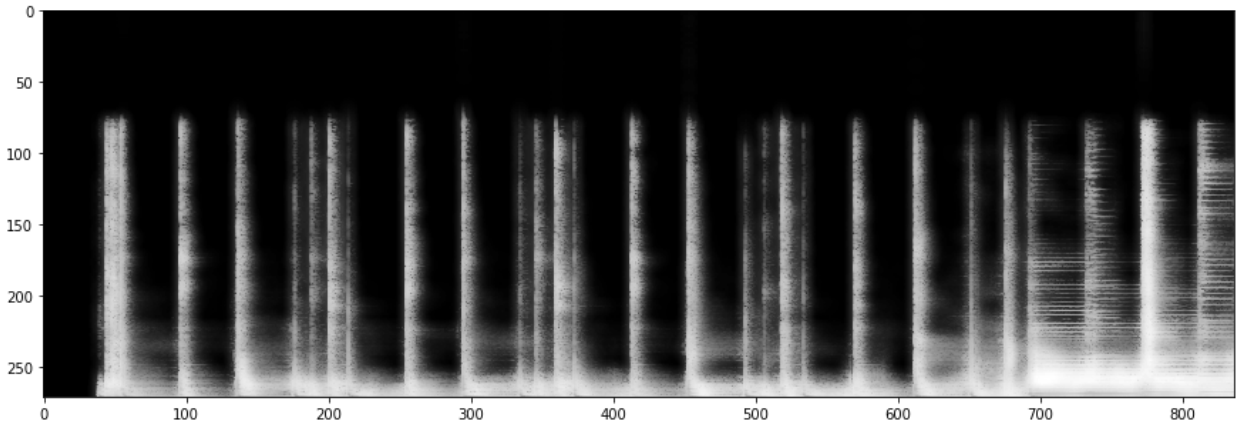
Spectrogram plot of the original audio.



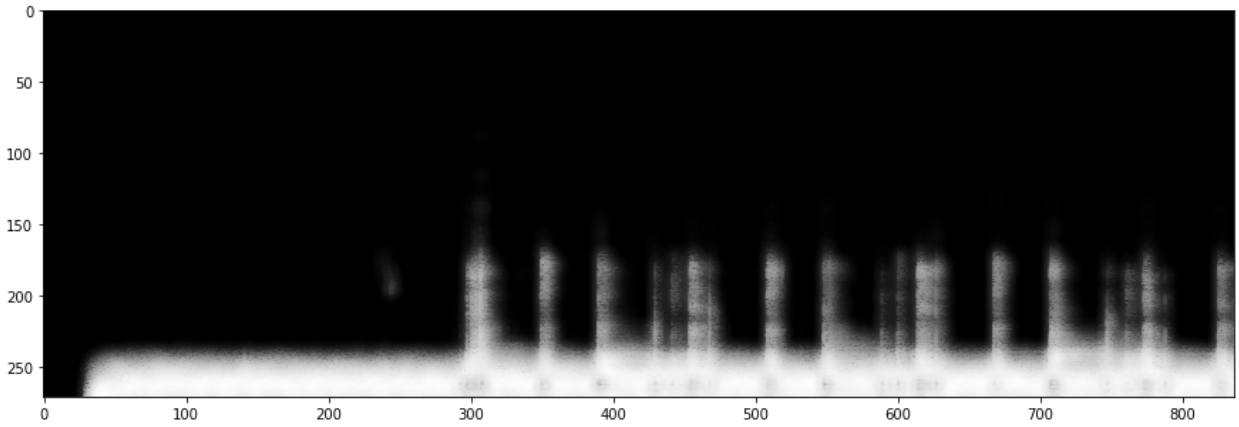
Spectrogram plot of the recorded audio.



We take the bilateral filter because we have to find the edges and it is easier with a filtered, smoothened image. This is the filtered image of the spectrogram of the original audio.

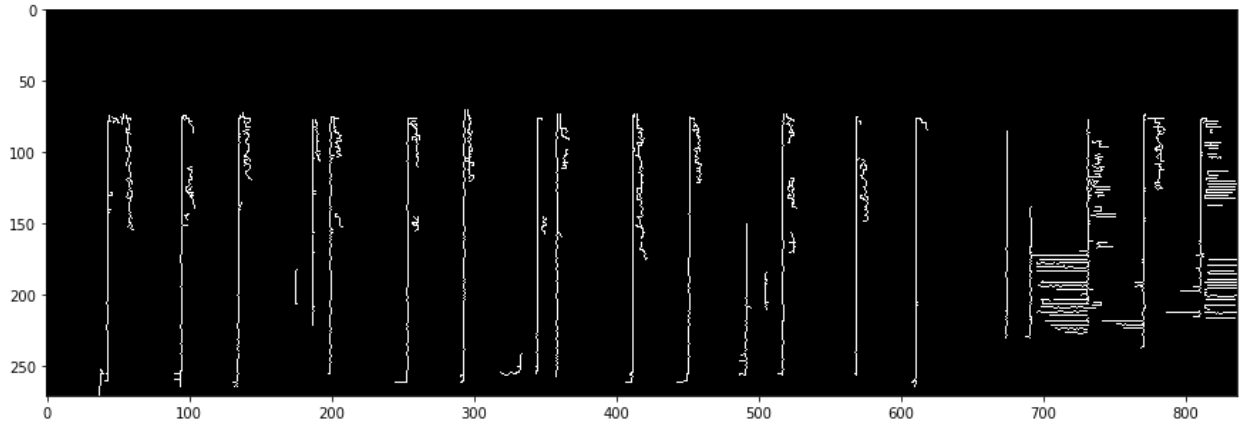


Filtered image of the spectrogram of the recorded audio.

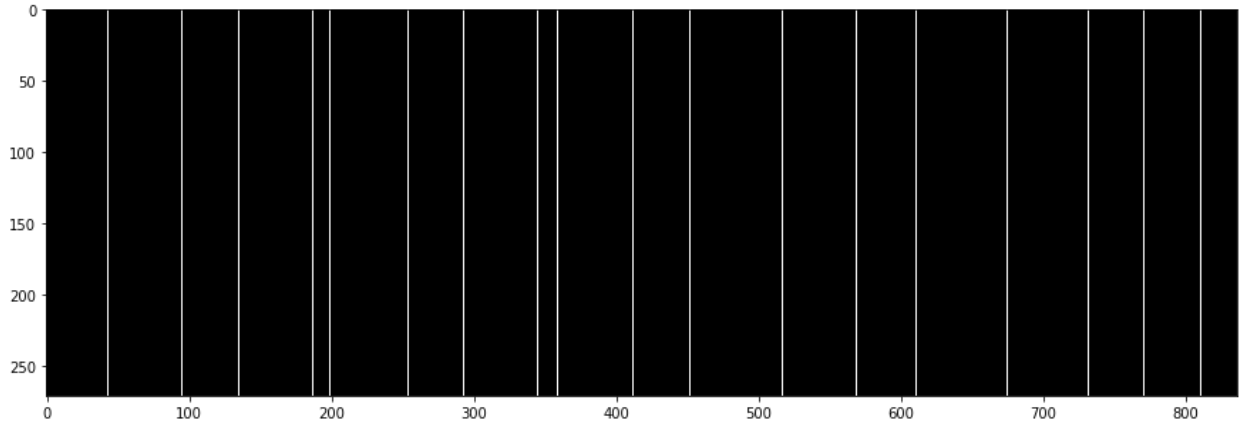


We calculate the edges and then plot the hough lines of the detected edges.

Edges of the original audio spectrogram.

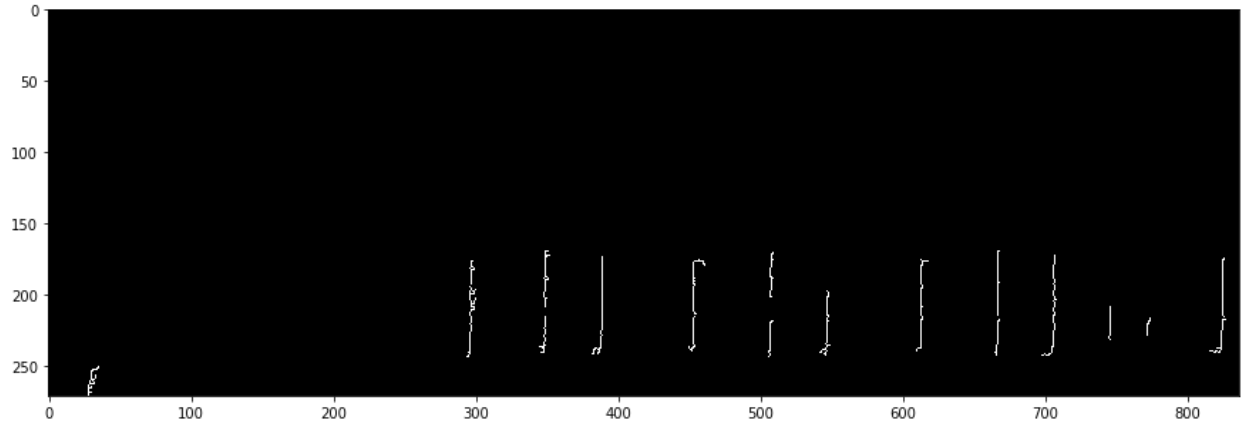


The hough lines of the above edge detected image of the spectrogram.

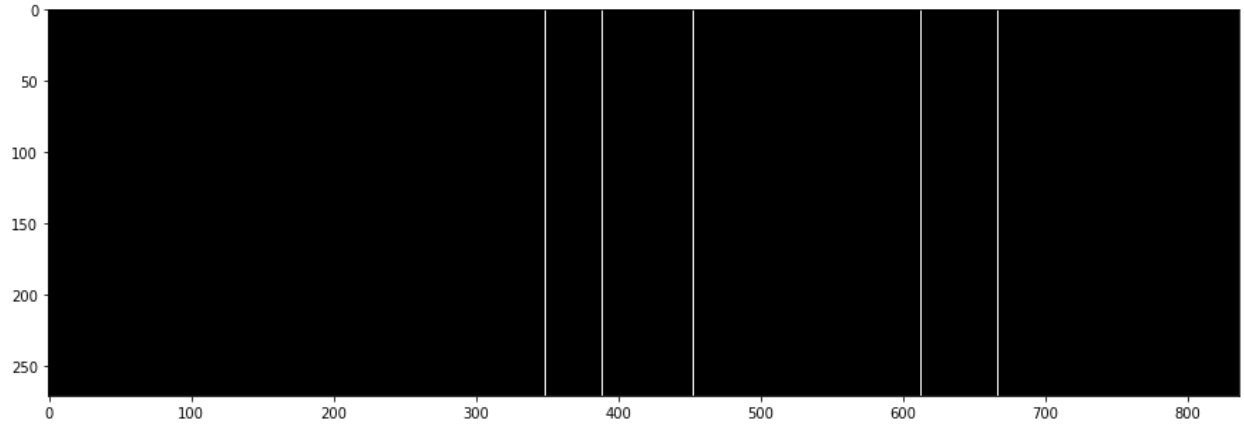


For the above image we store the information of the lines on the x axis that is the time axis.
Number of lines: 18
Matrix of the time information:
[0.51538794 1.13864777 1.61807841 2.24133825 2.38516744 3.04438457
3.51182945 4.13508928 4.30289001 4.9381356 5.41756625 6.19664104
6.81990087 7.32330304 8.09039207 8.77358073 9.24102561 9.72045625]
We proceed to take the relative information and store it into another matrix.

Similarly that of the recorded audio. This is the edges of the recorded audio spectrogram.



The hough lines of the above edge detected image of the spectrogram.



For the above image we store the information of the lines on the x axis that is the time axis.
Number of lines: 5
Matrix of the time information: [4.16965352 4.64755078 5.41218638 7.32377539 7.96893668]
Then we proceed to take the relative information and store it into another matrix.

We then match the two relative information matrices and calculate the similarity and the offset in the time.

CODE AND OUTPUT

The image outputs are used in the explanation above. We got the similarity of the two audios as 99.38% and offset time is 3.122 seconds.

```
import IPython.display as ipd
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
```

```
import cv2
import math
```

```
file_path1 = "Believer.wav"
file_path2 = "Believer_recorded.wav"
samples1, sampling_rate1 = librosa.load(file_path1, sr=None, mono=True, offset=0.0, duration=None)
samples2, sampling_rate2 = librosa.load(file_path2, sr=None, mono=True, offset=0.0, duration=None)
```

```
duration_of_sound1 = len(samples1)/sampling_rate1
duration_of_sound2 = len(samples2)/sampling_rate2
print(duration_of_sound1, "sec full sound")
print(duration_of_sound2, "sec part sound")
```

```
from IPython.display import Audio
Audio(file_path1)
```

```
Audio(file_path2)
```

```
plt.figure(figsize=(14,5))
librosa.display.waveplot(y=samples1, sr=sampling_rate1)
plt.xlabel("time(sec)-->")
plt.ylabel("Amplitude")
plt.show()
```

```
plt.figure(figsize=(14,5))
librosa.display.waveplot(y=samples2, sr=sampling_rate2)
plt.xlabel("time(sec)-->")
plt.ylabel("Amplitude")
plt.show()
```

```
fig_size1 = 15
X1 = librosa.stft(samples1)
Xdb1 = librosa.amplitude_to_db(abs(X1))
plt.figure(figsize=(fig_size1,5))
ax = plt.axes()
ax.set_axis_off()
```

```
librosa.display.specshow(Xdb1, sr=sampling_rate1, x_axis='time',y_axis='hz')
#plt.colorbar()
plt.savefig('Believerfull.png', bbox_inches='tight', transparent=True, pad_inches=0.0 )
```

```
fig_size2 = 15
X2 = librosa.stft(samples2)
Xdb2 = librosa.amplitude_to_db(abs(X2))
plt.figure(figsize=(fig_size2,5))
ax = plt.axes()
ax.set_axis_off()
librosa.display.specshow(Xdb2, sr=sampling_rate2, x_axis='time',y_axis='hz')
#plt.colorbar()
plt.savefig('Believerrecorded.png', bbox_inches='tight', transparent=True, pad_inches=0.0 )
```

```
image1 = cv2.imread('Believerfull.png')
image_gray1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(fig_size1,5))
plt.imshow(image_gray1, cmap='gray')
```

```
image2 = cv2.imread('Believerrecorded.png')
image_gray2 = cv2.cvtColor(image2,cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(fig_size2,5))
plt.imshow(image_gray2, cmap='gray')
```

```
bilateral1 = cv2.bilateralFilter(image_gray1, 15,75,75)
plt.figure(figsize=(fig_size1,5))
plt.imshow(bilateral1, cmap='gray')
```

```
bilateral2 = cv2.bilateralFilter(image_gray2, 15,75,75)
plt.figure(figsize=(fig_size2,5))
plt.imshow(bilateral2, cmap='gray')
```

```
edges1 = cv2.Canny(bilateral1, 100, 200)
lines1 = cv2.HoughLines(edges1,1,np.pi/180,100)
plt.figure(figsize=(fig_size1,5))
plt.imshow(edges1, cmap='gray')
```

```
edges2 = cv2.Canny(bilateral2, 100, 200)
lines2 = cv2.HoughLines(edges2,1,np.pi/180,50)
plt.figure(figsize=(fig_size2,5))
plt.imshow(edges2, cmap='gray')
```

```
img1 = image_gray1*0
if lines1 is not None:
    for i in range(0, len(lines1)):
        rho = lines1[i][0][0]
        theta = lines1[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a*rho
        y0 = b*rho
        pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
        pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
        cv2.line(img1, pt1, pt2, (255,255,255), 1)
plt.figure(figsize=(15,5))
plt.imshow(img1,cmap='gray')
```

```
img2 = image_gray2*0
if lines2 is not None:
    for i in range(0, len(lines2)):
        rho = lines2[i][0][0]
        theta = lines2[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a*rho
        y0 = b*rho
        pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
        pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
        cv2.line(img2, pt1, pt2, (255,255,255), 1)

plt.figure(figsize=(15,5))
plt.imshow(img2,cmap='gray')
```

```
#img = img.astype(np.uint8)
#img1 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
np.shape(img1)
r,c = np.shape(img1)
division = duration_of_sound1/c
```

```
lines_total1 = int(np.sum(img1[100,:])/255)
print(lines_total1)
lines_time1 = np.zeros(lines_total1)
temp=0
for i in range(0,c):
    if img1[100,i] == 255:
        lines_time1[temp]=i
        temp=temp+1
print(lines_time1)
lines_time1 = lines_time1*division
print(lines_time1)
```

```
18
[ 43.  95. 135. 187. 199. 254. 293. 345. 359. 412. 452. 517. 569. 611.
 675. 732. 771. 811.]
[0.51538794 1.13864777 1.61807841 2.24133825 2.38516744 3.04438457
 3.51182945 4.13508928 4.30289001 4.9381356  5.41756625 6.19664104
 6.81990087 7.32330304 8.09039207 8.77358073 9.24102561 9.72045625]
```

```
np.shape(img2)
r,c = np.shape(img2)
division = duration_of_sound2/c
lines_total2 = int(np.sum(img2[100,:])/255)
print(lines_total2)
lines_time2 = np.zeros(lines_total2)
temp=0
for i in range(0,c):
    if img2[100,i] == 255:
        lines_time2[temp]=i
        temp=temp+1
print(lines_time2)
lines_time2 = lines_time2*division
print(lines_time2)
```

```
5
[349. 389. 453. 613. 667.]
[4.16965352 4.64755078 5.41218638 7.32377539 7.96893668]
```

```
hash_1 = np.zeros((lines_total1-1,lines_total1-1))
for i in range(0,lines_total1-1):
    temp = lines_time1[i]
    #print(temp)
    temp1 = 0
    for j in range(i+1,lines_total1):
```

```
        hash_1[i,temp1] = lines_time1[j]-temp
        #print(lines_time1[j])
        temp1 = temp1+1
print(hash_1)
```

```
[[0.62325983 1.10269047 1.72595031 1.8697795  2.52899663 2.99644151
  3.61970134 3.78750207 4.42274767 4.90217831 5.6812531  6.30451293
  6.80791511 7.57500413 8.2581928  8.72563767 9.20506831]
 [0.47943064 1.10269047 1.24651967 1.9057368  2.37318167 2.99644151
  3.16424223 3.79948783 4.27891847 5.05799327 5.6812531  6.18465527
  6.9517443  7.63493296 8.10237784 8.58180848 0.        ]
 [0.62325983 0.76708903 1.42630616 1.89375103 2.51701087 2.68481159
  3.32005719 3.79948783 4.57856262 5.20182246 5.70522463 6.47231366
  7.15550232 7.6229472  8.10237784 0.         0.        ]
 [0.14382919 0.80304632 1.2704912  1.89375103 2.06155176 2.69679736
  3.176228   3.95530279 4.57856262 5.0819648  5.84905382 6.53224249
  6.99968736 7.479118   0.         0.         0.        ]
 [0.65921713 1.12666201 1.74992184 1.91772256 2.55296816 3.03239881
  3.8114736  4.43473343 4.9381356  5.70522463 6.38841329 6.85585817
  7.33528881 0.         0.         0.         0.        ]
 [0.46744488 1.09070471 1.25850543 1.89375103 2.37318167 3.15225647
  3.7755163  4.27891847 5.0460075  5.72919616 6.19664104 6.67607168
  0.         0.         0.         0.         0.        ]
 [0.62325983 0.79106056 1.42630616 1.9057368  2.68481159 3.30807142
  3.8114736  4.57856262 5.26175129 5.72919616 6.2086268  0.
  0.         0.         0.         0.         0.        ]
 [0.16780072 0.80304632 1.28247697 2.06155176 2.68481159 3.18821376
  3.95530279 4.63849145 5.10593633 5.58536697 0.         0.
  0.         0.         0.         0.         0.        ]
 [0.6352456  1.11467624 1.89375103 2.51701087 3.02041304 3.78750207
  4.47069073 4.9381356  5.41756625 0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.47943064 1.25850543 1.88176527 2.38516744 3.15225647 3.83544513
  4.30289001 4.78232065 0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.77907479 1.40233463 1.9057368  2.67282582 3.35601449 3.82345936
  4.30289001 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.62325983 1.12666201 1.89375103 2.5769397  3.04438457 3.52381521
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.50340217 1.2704912  1.95367986 2.42112474 2.90055538 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.76708903 1.45027769 1.91772256 2.39715321 0.         0.
```

```
 0.        0.        0.        0.        0.        0.
 0.        0.        0.        0.        0.       ]
[0.68318866 1.15063354 1.63006418 0.        0.        0.
 0.        0.        0.        0.        0.        0.
 0.        0.        0.        0.        0.       ]
[0.46744488 0.94687552 0.        0.        0.        0.
 0.        0.        0.        0.        0.        0.
 0.        0.        0.        0.        0.       ]
[0.47943064 0.        0.        0.        0.        0.
 0.        0.        0.        0.        0.        0.
 0.        0.        0.        0.        0.       ]]
```

```
hash_2 = np.zeros((lines_total2-1,lines_total2-1))
for i in range(0, lines_total2-1):
    temp = lines_time2[i]
    temp1=0
    for j in range(i+1,lines_total2):
        hash_2[i,temp1] = lines_time2[j]-temp
        temp1=temp1+1
print(hash_2)
```

```
[[0.47789725 1.24253286 3.15412186 3.79928315]
 [0.7646356  2.67622461 3.3213859  0.        ]
 [1.91158901 2.5567503  0.         0.        ]
 [0.64516129 0.         0.         0.        ]]
```

```
match = np.zeros((lines_total2-1,lines_total2-1))
time = np.zeros((lines_total2-1,lines_total2-1))
i = 0
i1 = 0
j = 0
j1 = 0
print('lines_total1 ',lines_total1-1)
print('lines_total2 ',lines_total2-1)
while i< lines_total2-1 and i1<lines_total2-1 and j<lines_total1-1 and j1<lines_total1-1:

    temp = hash_2[i,i1]
    temp1 = hash_1[j,j1]

    percent = 1-abs(temp1-temp)/temp
    if temp==0 and i<lines_total2-2:
        i=i+1
        i1=0
        temp = hash_2[i,i1]
```

```
    if temp1==0:
        j=j+1
        j1=0
        temp1 = hash_1[j,j1]
    if percent<0.96 and temp1>temp:
        j = j+1
        j1=0
    elif percent>0.96:
        match[i,i1] = percent
        time[i,i1] = abs(lines_time1[j]-lines_time2[i])
        i1 = i1+1
        j1 = j1+1

    elif percent<0.96 and temp1<temp:
        j1=j1+1

    if i1==lines_total2-1 and match[i,i1-1] != 0:
        i=i+1
        i1=0
```

```
n = np.sum(match>0)
similarity = np.sum(match)/n
print(similarity)
```

0.9938744973290772

```
m = np.sum(time>0)
offset_time = np.sum(time)/m
print(offset_time)
```

3.122225310673852

CONCLUSION

We practically wrote assembly language code for a basic OS and implemented it. We theoretically learnt how the OS boots upon the start of the device.

We worked on audio fingerprinting and were able to match recorded audio to the original audio with 99.38% similarity and calculated the offset time as 3.12seconds in the recorded audio, where the actual offset in time is 3.4seconds.