

**Visvesvaraya National Institute of Technology, Nagpur**

**Department of Electronics and Communication Engineering**

**Mini Project Report**

**TITLE**

**Submitted to:** Dr. K Surender (Lab Instructor)

**Course:** Measurement and Instrumentation Lab

**Date:** 7/11/2023

**Submitted By:**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Name of Students** | **Roll No.** |
| 1 | PRABHUNE ATHARVA GAJANAN | BT21ECE065 |
| 2 | YATHARTH SUNEEL | BT21ECE066 |
| 3 | SANKARNARAYAN CHANDRASEKHAR | BT21ECE067 |
| 4 | GAIKWAD JAY KISHOR | BT21ECE059 |

* **Project Introduction :**

**Introduction to Audio Signal Processing :**Audio Signal Processing is a branch of signal processing that deals specifically with the manipulation of audio signals. An audio signal is a representation of sound, typically in the form of an electrical voltage for analog signals or as a series of discrete values for digital signals. Audio signal processing involves the analysis, synthesis, and modification

of audio signals for various purposes.

Audio Signal processing is a method where intensive algorithms, techniques are applied to audio signals. Audio signals are the representation of sound, which is in the form of digital and analog signals. Their frequencies range between 20 to 20,000 Hz, and this is the lower and upper limit of our ears. Analog signals occur in electrical signals, while digital signals occur in binary representations. This process encompasses removing unwanted noise and balancing the time-frequency ranges by converting digital and analog signals. It focuses on computational methods for altering the sounds. It removes or minimizes the overmodulation, echo, unwanted noise by applying various techniques into it.

With the help of Python Script using the Librosa library we have perform the vocal separation from an audio Signal. The script The script loads an example audio file, computes the spectrogram magnitude and phase, and then separates the vocals from

the accompanying instrumentation using a method based on cosine similarity and soft-masking.

* **Software Requirement :** Software used : Python IDE – Jupyter Notebook 6.5.4 Libraries used : NumPy  
   Matplotlib

Librosa

* **Project Description:**The main components and processes involved in the project are :   
   **1) Audio File Loading :**   
  librosa.load function is used to load an audio file This variable holds the path to the audio file that you want to load. In this code, the file path is set to 'C:/Users/17205/Downloads/File/hello.mp3'. The function takes the file path (audio\_file\_path) as its first argument. The function returns two values: audio\_data and sample\_rate,

1. audio\_data: This is a one-dimensional NumPy array containing the audio signal.
2. sample\_rate: This is an integer representing the number of samples per second (in Hertz) used to represent the audio.

**2) Spectrogram Computation :**   
The computation of the spectrogram magnitude and phase is done using the Librosa library. The librosa.stft function is used to compute the Short-Time Fourier Transform of the audio signal (audio\_data).The STFT represents how the frequency content of the signal changes over time by dividing the signal into short overlapping windows and performing a Fourier Transform on each window. Then with the help of librosa.magphase we extract the magnitude and phase. So, after these computations, we have the magnitude information (spectrum\_magnitude) that can be used to visualize the spectrogram, and the phase information (phase) that can be used in various audio processing tasks.

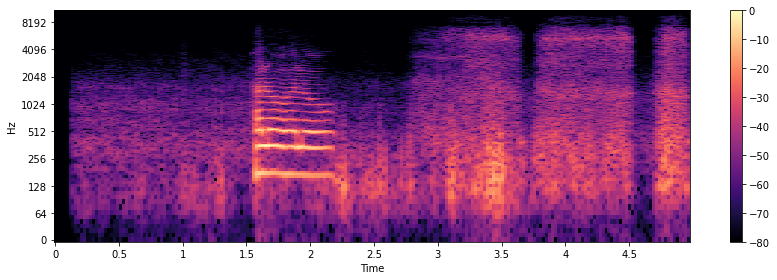
**3) Vocal Separation Algorithm :**The separation of vocals from instrumentation involves several steps, including the use of cosine similarity and soft-masking techniques. The librosa.decompose.nn\_filter function is used to compare frames of the magnitude spectrogram using cosine similarity. The metric='cosine' parameter specifies the cosine similarity metric.The width parameter sets the minimum distance between similar frames. The aggregate=np.median parameter specifies that the aggregated value for similar frames is the median of those frames. This step helps to suppress sparse/non-repetitive deviations from the average spectrum. The output of the filter is then element-wise compared with the original spectrum magnitude using np.minimum. This ensures that the output of the filter is not greater than the input spectrum, assuming signals are additive. Soft-masking is applied to the filtered spectrum using the librosa.util.softmask function. Two masks are created: mask\_instrument and mask\_vocal.These masks are based on the filtered spectrum and the difference between the original spectrum and the filtered spectrum, with additional margin adjustments.The final step involves multiplying the masks with the original spectrum magnitude. foreground represents the vocal component, and background represents the instrumental component.

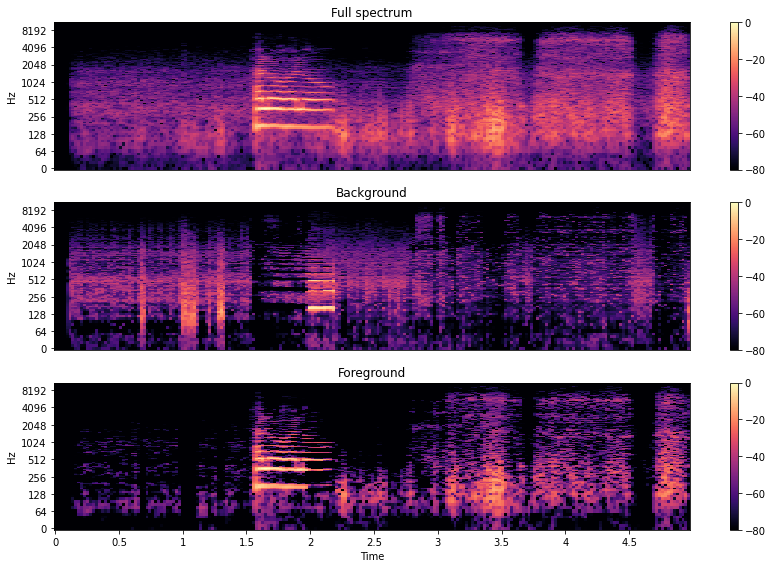
**4) Plotting and Visualization :**

The visualization process involves creating a multi-subplot figure where the original spectrum, background (instrumental), and foreground (vocal) components are displayed separately. The use of Librosa's specshow function ensures proper visualization of spectrograms, and Matplotlib is used for overall figure management. The resulting plot provides a visual representation of the separated vocal and instrumental components for the specified time frame. We can also adjust the figure size, layout, and plot titles according to specific preferences and requirements. The saved image ('Signal\_Background\_Foreground\_Separated.jpg') can be used for documentation or analysis.

* **Code and Output :**

|  |
| --- |
| ##################  # Standard imports  **from** **\_\_future\_\_** **import** print\_function  **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **librosa**  **import** **librosa.display**  #############################################  # Load an example audio file with vocals.  audio\_file\_path = 'C:/Users/17205/Downloads/File/hello.mp3'  audio\_data, sample\_rate = librosa.load(audio\_file\_path, duration=**120**)  # Compute the spectrogram magnitude and phase  spectrum\_magnitude, phase = librosa.magphase(librosa.stft(audio\_data))  #######################################  # Display a 5-second slice of the spectrogram  time\_frame\_slice = slice(\*librosa.time\_to\_frames([**0**, **5**], sr=sample\_rate))  plt.figure(figsize=(**12**, **4**))  librosa.display.specshow(librosa.amplitude\_to\_db(spectrum\_magnitude[:, time\_frame\_slice], ref=np.max),  y\_axis='log', x\_axis='time', sr=sample\_rate)  plt.colorbar()  plt.tight\_layout()  ###########################################################  # The wiggly lines above are due to the vocal component.  # Our goal is to separate them from the accompanying  # instrumentation.  # The steps below outline the process.  # We'll compare frames using cosine similarity, and aggregate similar frames  # by taking their (per-frequency) median value.  # To avoid being biased by local continuity, we constrain similar frames to be  # separated by at least 2 seconds.  # This suppresses sparse/non-repetetitive deviations from the average spectrum,  # and works well to discard vocal elements.  filtered\_spectrum = librosa.decompose.nn\_filter(spectrum\_magnitude,  aggregate=np.median,  metric='cosine',  width=int(librosa.time\_to\_frames(**2**, sr=sample\_rate)))  # The output of the filter shouldn't be greater than the input  # if we assume signals are additive. Taking the pointwise minimium  # with the input spectrum forces this.  filtered\_spectrum = np.minimum(spectrum\_magnitude, filtered\_spectrum)  ##############################################  # The raw filter output can be used as a mask,  # but it sounds better if we use soft-masking.  # We can also use a margin to reduce bleed between the vocals and instrumentation masks.  # Note: the margins need not be equal for foreground and background separation  margin\_instrument, margin\_vocal = **2**, **10**  power = **2**  mask\_instrument = librosa.util.softmask(filtered\_spectrum,  margin\_instrument \* (spectrum\_magnitude - filtered\_spectrum),  power=power)  mask\_vocal = librosa.util.softmask(spectrum\_magnitude - filtered\_spectrum,  margin\_vocal \* filtered\_spectrum,  power=power)  # Once we have the masks, simply multiply them with the input spectrum  # to separate the components  foreground = mask\_vocal \* spectrum\_magnitude  background = mask\_instrument \* spectrum\_magnitude  ##########################################  # Plot the same slice, but separated into its foreground and background  # sphinx\_gallery\_thumbnail\_number = 2  plt.figure(figsize=(**12**, **8**))  plt.subplot(**3**, **1**, **1**)  librosa.display.specshow(librosa.amplitude\_to\_db(spectrum\_magnitude[:, time\_frame\_slice], ref=np.max),  y\_axis='log', sr=sample\_rate)  plt.title('Full spectrum')  plt.colorbar()  plt.subplot(**3**, **1**, **2**)  librosa.display.specshow(librosa.amplitude\_to\_db(background[:, time\_frame\_slice], ref=np.max),  y\_axis='log', sr=sample\_rate)  plt.title('Background')  plt.colorbar()  plt.subplot(**3**, **1**, **3**)  librosa.display.specshow(librosa.amplitude\_to\_db(foreground[:, time\_frame\_slice], ref=np.max),  y\_axis='log', x\_axis='time', sr=sample\_rate)  plt.title('Foreground')  plt.colorbar()  plt.tight\_layout()  plt.savefig('Signal\_Background\_Foreground\_Seperated.jpg')  plt.show() |

****

****

* **Application:   
    
  1) Speech Enhancement :**In applications like voice recording or conference calls,

vocal separation can be used to enhance the clarity of speech by isolating and adjusting the vocal

**2) Automatic Speech Recognition (ASR):**Vocal separation can improve the accuracy of ASR systems by providing cleaner speech signals for recognition.

**3) Karaoke Systems:**   
Karaoke systems can benefit from vocal separation to provide customizable playback options, allowing users to sing along with or without the original vocals.

**4)** **Music Analysis and Genre Classification:**Researchers and music analysts can use vocal separation for detailed analysis of musical elements and genre classification.

* **Conclusion:**With the help of Python and its various libraries like NumPy, Matplotlib and Librosa we were able to able to separate the vocals from the overall spectrum . The project involves various steps like loading an audio file, computing its spectrogram and utilizing combination of cosine similarity filtering and soft-masking techniques to separate vocals from instrumentation. In terms of application this project exceeds the expectation because if we observe carefully then almost every field can make use of the project so far. So overall this project tends to provide more flexibility to the user according to its need . Additional features can also be included in the future work.