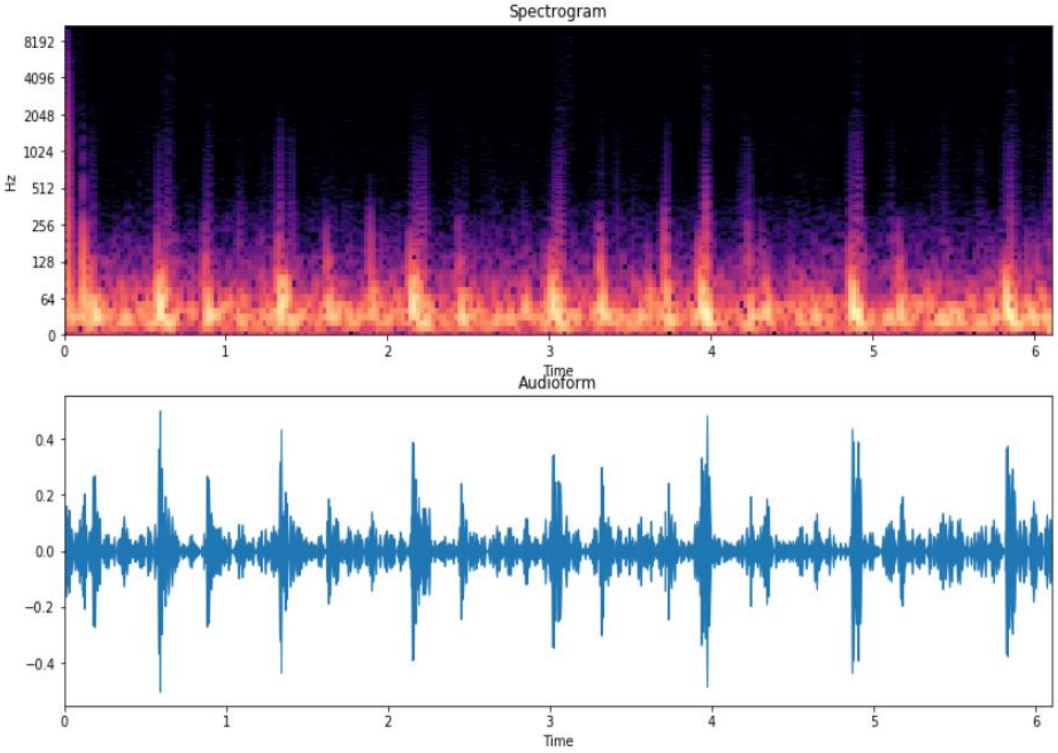
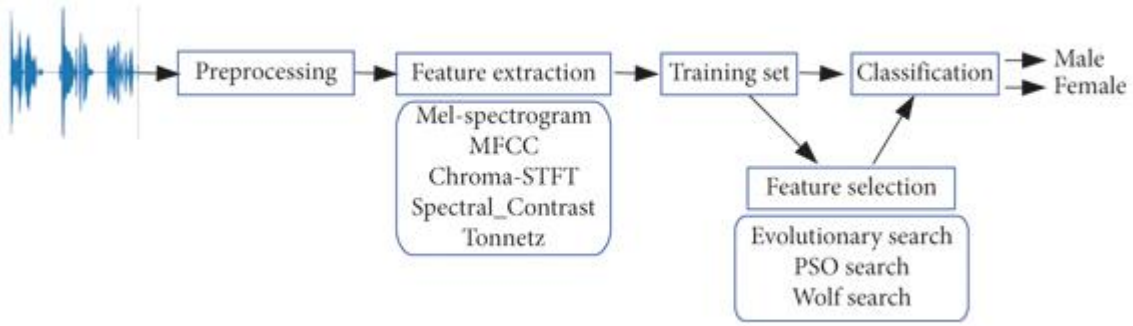

Spectrogram
8192
4096
2048
1024
512
256
128
64
0
Hz
0
1
2
3
4
5
6
Time
Audioform
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6
Time


Preprocessing
Feature extraction
Mel-spectrogram
MFCC
Chroma-STFT
Spectral_Contrast
Tonnetz
Training set
Classification
Male
Female
Feature selection
Evolutionary search
PSO search
Wolf search

Sound Classification Using Deep Learning

Table of Content

Software Setup

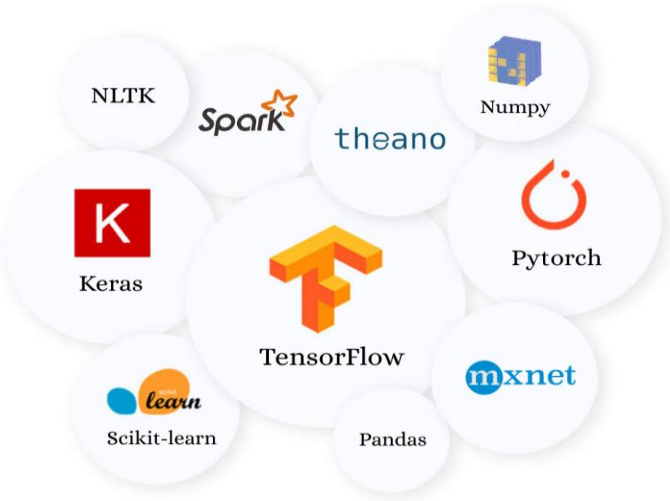


Figure 1 Techniques for Sound Classification

The software techniques that can be used to develop a sound classification system using deep learning are as following:

1. Tensor Flow: Tensor Flow is an open-source machine learning system developed by Google. It offers a wide range of tools for building and training deep learning models, including pre-built models for audio classification.
2. PyTorch: PyTorch is another open source machine learning framework developed by Facebook. Similar to Tensor Flow, it also offers a wide range of tools for building and training deep learning models.
3. Keras: Keras is a high-level neural network API that can run on top of Tensor Flow or another backend. It allows rapid experimentation and is user-friendly, making it a popular choice for building deep learning models.
4. Scikit-learn: scikit-learn is a machine learning library for Python that offers a wide range of tools for preprocessing, feature extraction, model selection, and evaluation. It can be used in conjunction with Tensor Flow or PyTorch to create a complete audio classification system.
5. Librosa: Librosa is a python library for music and audio analysis. It provides tools for loading audio files, extracting features, and visualizing audio data. It is commonly used in feature extraction for an audio classification system.

In addition to these frameworks and libraries, it's important to have a dataset of labeled sound samples for training the model. The dataset should be large enough and diverse enough to train the model effectively.

Proposed Voice System Components

- ### Feature Extraction

Feature extraction is the process of extracting relevant information from raw data in order to feed it into a machine learning model. In the context of voice gender classification, the raw data is typically audio in the form of .wav files. One popular method for feature extraction from audio data is the use of Mel-Frequency Cepstral Coefficients (MFCCs) and Mel Spectrogram.

Mel Spectrogram is a representation of audio that shows the spectral content of the audio signal over time. It is created by applying a Fourier transform to the audio signal, and then applying a non-linear transformation called the Mel Scale to the resulting spectrum. The Mel Scale is designed to better match the way humans perceive pitch, and it is particularly useful for speech and music analysis.

MFCCs are a set of coefficients that describe the shape of the Mel Spectrogram. They are derived from the log of the Mel Spectrogram by applying a discrete cosine transform (DCT). MFCCs capture the spectral information of the audio signal in a compact representation, and they are commonly used as features in speech and music recognition systems.

When using CNNs for voice gender classification, the Mel Spectrogram and MFCCs are commonly used as inputs to the network. The CNN can be trained to learn patterns in the Mel Spectrogram and MFCCs that are characteristic of different genders, and then make a final decision based on those patterns.

The importance of Mel Spectrogram and MFCCs in deep learning for voice gender classification is that they provide a more meaningful representation of the audio signal for the CNN to learn from. These features are specifically designed to capture the spectral information of the audio signal that is most relevant for speech and music analysis, which is important for voice gender classification. Moreover, these features are more robust to noise and other distortions, which is essential for real-world application.

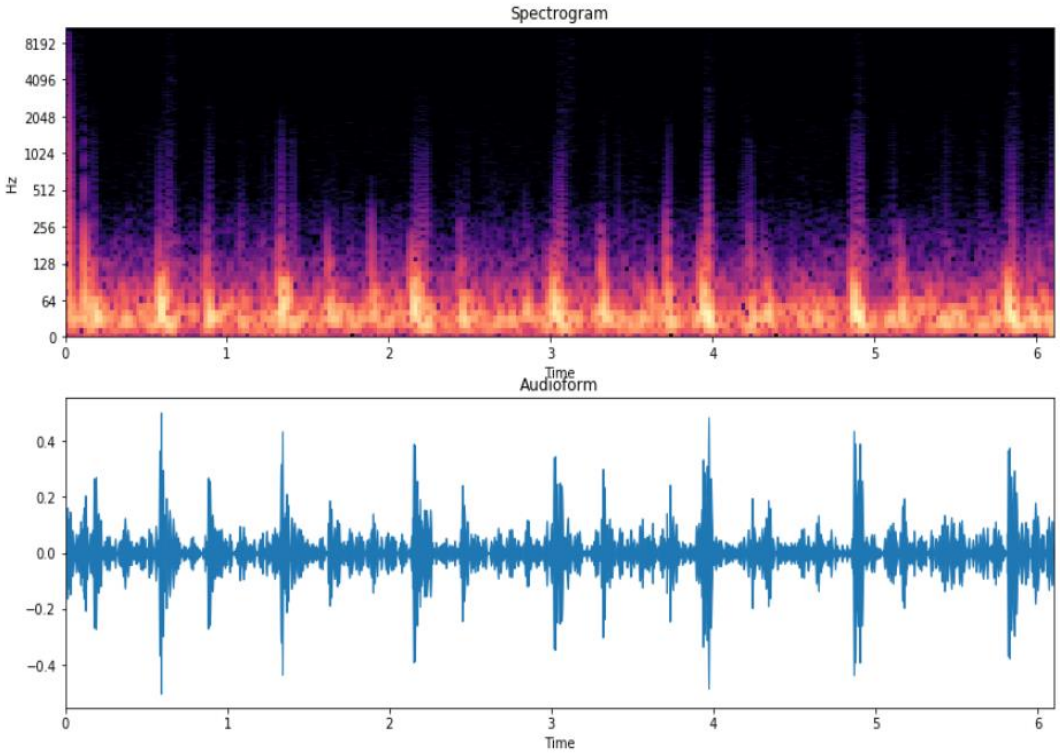


Figure 2 Mel Spectrogram of below audio signal

- **Model Development**

Our proposed DCNN Model performed well on test dataset classifying the sound as of either Male, Female or too much noise with 91% accuracy. Our High-Level Model has following Architecture.

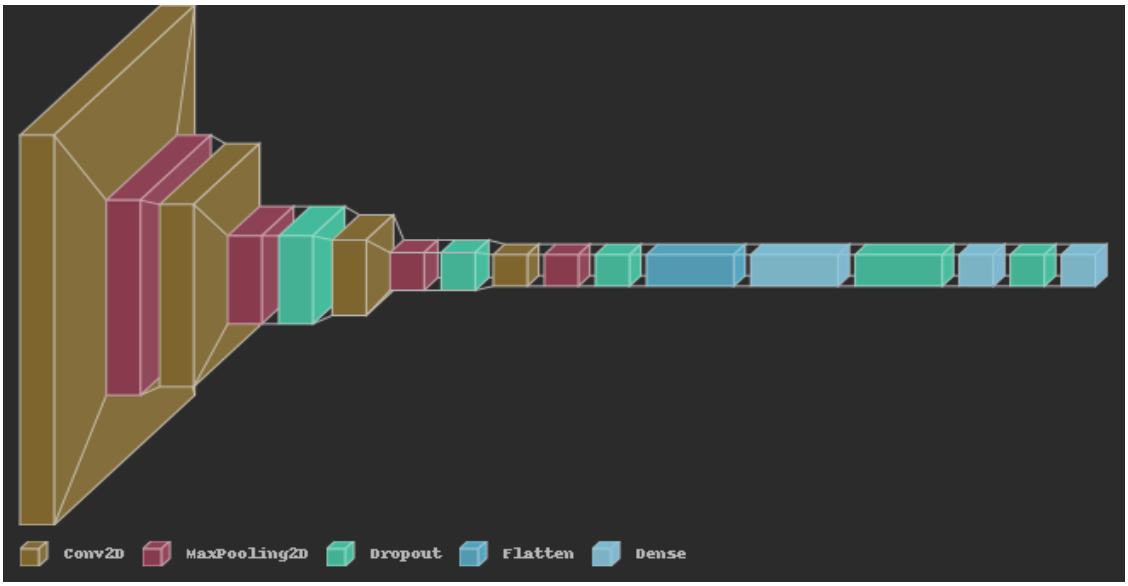


Figure 3 Layered View of Model

The first layer is a 2D convolution layer (Conv2D) with 32 filters and kernel size (3, 3). The input shape of this layer is (64, 64, 1), which represents the height, width, and number of channels of the input data. The activation function used is "ReLU", which means rectified linear unit. This is a common activation function used in CNNs because it helps the model learn non-linear relationships between input and output.

The next layer is the 2D max pooling layer (MaxPooling2D) with pool size (2, 2). This layer is used to reduce the spatial dimensions of the data, which helps reduce the number of parameters in the model and avoid overfitting.

The model then has 3 more Conv2D layers with 64, 128, and 128 filters respectively, each followed by a MaxPooling2D layer. Each of these layers increases the number of filters, which helps the model to learn more complex features. After each Conv2D layer there is a Dropout layer with rate 0.5 which drop out half of the neurons to prevent overfitting.

After the 4th MaxPooling2D layer, the data is flattened with the Flatten() layer. This converts the data from a 2D array to a 1D array so that it can be fed into a fully connected neural network (FCN). The FCN consist of 3 Dense layers with 512, 128, and 3 neurons respectively. The last Dense layer has 3 neurons and the activation function is 'softmax', which is commonly used for multi-class classification tasks. This will output the probability of each class.

Total parameters and input/output shape of each layer is displayed in table 1.

Layer	*Output Shape*	*Parameters*
Conv2D	(None, 62, 62, 32)	320
MaxPooling2D	(None, 31, 31, 32)	0
Conv2D	(None, 29, 29, 64)	18496
MaxPooling2D	(None, 14, 14, 64)	0
Dropout	(None, 14, 14, 64)	0
Conv2D	(None, 12, 12, 128)	73856
MaxPooling2D	(None, 6, 6, 128)	0
Dropout	(None, 6, 6, 128)	0
Conv2D	(None, 4, 4, 128)	147584
MaxPooling2D	(None, 2, 2, 128)	0
Dropout	(None, 2, 2, 128)	0
Flatten	(None, 512)	0
Dense	(None, 512)	262656
Dropout	(None, 512)	0
Dense	(None, 128)	65664
Dropout	(None, 128)	0
Dense	(None, 3)	387

Total parameters: 568,963 *Trainable parameters: 568,963* *Non-trainable parameters: 0*

Table 1 Model Parameters

The model gives 91.007 % Accuracy on unseen dataset.

- **Real Time Mic Recording**

Real-time microphone recording using PyAudio works by opening a stream of audio data from the microphone, and then continuously reading and processing this data in a loop. The PyAudio library provides a simple and consistent interface for recording and playing audio on a variety of platforms and allows you to specify a number of audio properties such as the sample rate, number of channels, and bit depth.

When you call the p.open() function, it sets up a new audio stream with the specified properties. The input=True argument tells PyAudio to open a recording stream rather than a playback stream. The frames_per_buffer argument specifies the number of audio frames that will be read from the stream at a time.

Once the stream is open, you can start recording by reading data from the stream in a loop. The stream.read() function reads a certain number of audio frames from the stream and returns them as a bytes object. In the example above, I are reading CHUNK number of frames at a time, which is defined as 1024. The frames list is used to store the audio data as it is being read.

Figure 4 Real time Sound Recording

After recording, the audio data is saved to a .wav file using the wave library. The wf.setnchannels(), wf.setsampwidth() and wf.setframerate() functions are used to set the number of channels, sample width and frame rate of the audio data respectively. Then wf.writeframes() writes the audio data to the .wav file.

Then I can load .wav file using the librosa.load() function, which takes in the file path and returns the audio data as a NumPy array and the sample rate. The sample rate can be used to resample the audio data if needed.

To create the Mel spectrogram from the audio data, I use the librosa.feature.melspectrogram() function, which takes in the audio data and the sample rate, and returns the Mel spectrogram as a 2D NumPy array. The Mel spectrogram is a representation of the audio data in the frequency domain, where the y-axis represents frequency and the x-axis represents time.

To save the Mel spectrogram as an image, I use the librosa.display.specshow() function, which takes in the Mel spectrogram and the sample rate and plots the spectrogram. Then I use the matplotlib.pyplot.savefig() function to save the plot as an image file.

Once the Mel spectrogram is saved as an image, I load it and use it as input to our voice gender classification model. I resized the image to match the input size of our model and

possibly normalize the pixel values. The model then makes a prediction on the gender of the person based on the features learned during training.

- **Voice Gender Classifier**

Gender identification is one of the main problems of speech analysis today. Gender tracking from acoustic data i.e. pitch, median, frequency, etc. Deep learning provides promising results for classification problems in all areas of research. Voice gender classification is a task in which a machine learning model is trained to determine the gender of a speaker based on their voice. Deep learning, a type of machine learning that uses neural networks, is a popular approach for this task.

One way to approach voice gender classification using deep learning is to use a convolutional neural network (CNN) to extract features from the audio data, and then feed those features into a fully connected neural network (FCN) for classification. The CNN can be trained to learn patterns in the audio data that are characteristic of different genders, while the FCN can be trained to make a final decision based on those patterns.

Another approach is to use a recurrent neural network (RNN) to process the audio data. RNNs are particularly well suited for sequential data such as speech, as they can learn patterns in the data that span multiple time steps. In this case, the RNN can be trained to learn patterns in the speech that are characteristic of different genders, and then make a final decision based on those patterns.

It is important to note that the accuracy of a voice gender classification model depends on the quality of the dataset used for training. In particular, the dataset should be balanced and diverse, meaning that it should include a wide range of voices from different individuals, and it should be representative of the population that the model is intended to be used on.

Overall, deep learning approaches like CNN, RNN, and FCN have shown good performance on voice gender classification tasks.

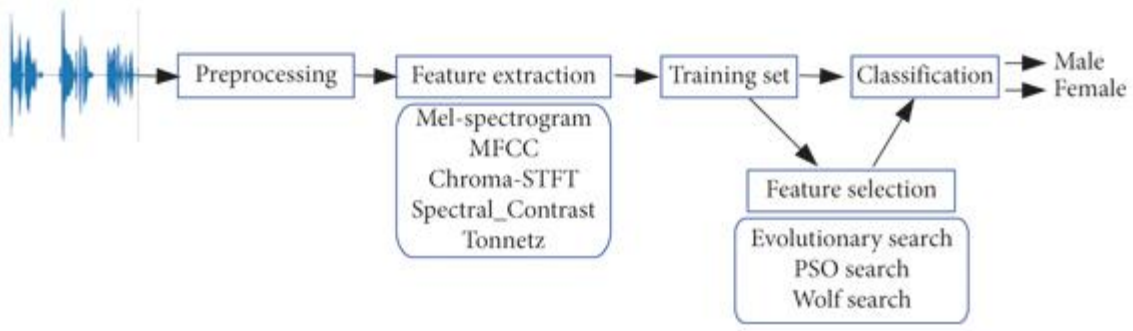


Figure 5 Overall Pipeline

I have chosen VoxCeleb and Urban 8 k dataset. VoxCeleb and Urban 8k dataset are two popular datasets used for speech recognition and other speech-related tasks.

The VoxCeleb dataset is a large collection of short audio clips of celebrities speaking, along with their corresponding transcriptions. The dataset was created by scraping YouTube videos of celebrities and contains over 150,000 clips from more than 6,000 different individuals. The clips vary in length, with some being as short as a few seconds and others being several minutes long. The VoxCeleb dataset is often used for tasks such as speaker verification, where the goal is to determine whether a given audio clip was spoken by a specific person, and speaker identification, where the goal is to identify the speaker of a given audio clip.

The Urban 8k dataset is a collection of urban soundscapes and environmental recordings collected from various cities around the world. The dataset contains over 87,000 audio clips of various environmental sounds, such as vehicles, animals, and human speech. The clips vary in length and are typically several seconds long. The Urban 8k dataset is often used for tasks such as environmental sound classification, where the goal is to identify the type of sound in a given audio clip, and sound event detection, where the goal is to detect specific sound events in a given audio clip.

- ## Distance Calculation

Our Module of distance calculation depends on the model`s output. Based on audibility of Males, Females and Noise we can classify whether speaker is inside the room or outside the room.

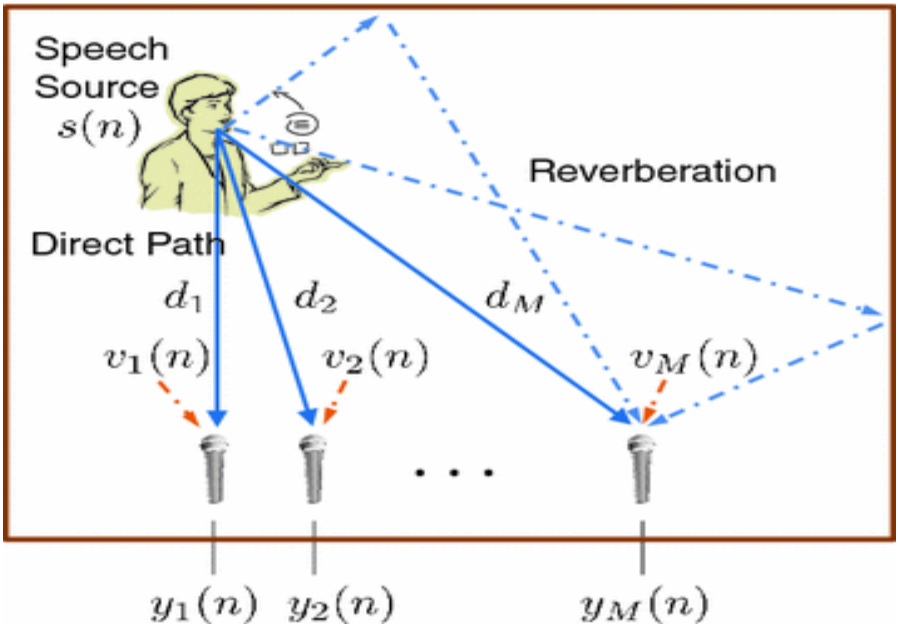


Figure 6 Sound Distance Estimation

This module uses the trained model to predict the gender of the speaker based on an input image (presumably a Mel spectrogram of an audio file). It uses the model.predict() function to get the predicted probability of the input image belonging to each class (male, female, or noise).

It then uses the np.argmax() function to find the class with the highest probability and assigns it as the final prediction. If the prediction is male, it checks the probabilities of the other classes (female, noise) to determine if the speaker is inside the room or far from the microphone, possibly outside the room. The same is done if the prediction is female. If the prediction is noise, it prints that the speaker is not audible.

The distance of the speaker from the microphone is determined by checking the probability of the final predicted class (male or female) and the probability of the noise class. If the final predicted class probability is high and the noise class probability is low, it is assumed that the speaker is inside the room. Otherwise, the speaker is assumed to be far from the microphone, possibly outside the room.

- **Angle/DOA Calculation**

This Module reads a .wav file named 'output.wav' and reads the audio data and sample rate using the wavfile.read function from the scipy library. The audio data is then converted to a numpy array.

Next, the code uses the signal.hilbert function from the scipy library to calculate the Hilbert transform of the audio data. The Hilbert transform is used to obtain the analytic signal from the audio data, which contains both the amplitude and phase information of the signal.

The code then calculates the phase angle of the analytic signal using the np.angle function from the numpy library. The phase angle is calculated in radians and then converted to degrees using the np.pi constant. The result is then printed.

The phase angle can be used to analyze the audio signal and extract information such as the fundamental frequency, which is the rate at which the phase angle changes over time.

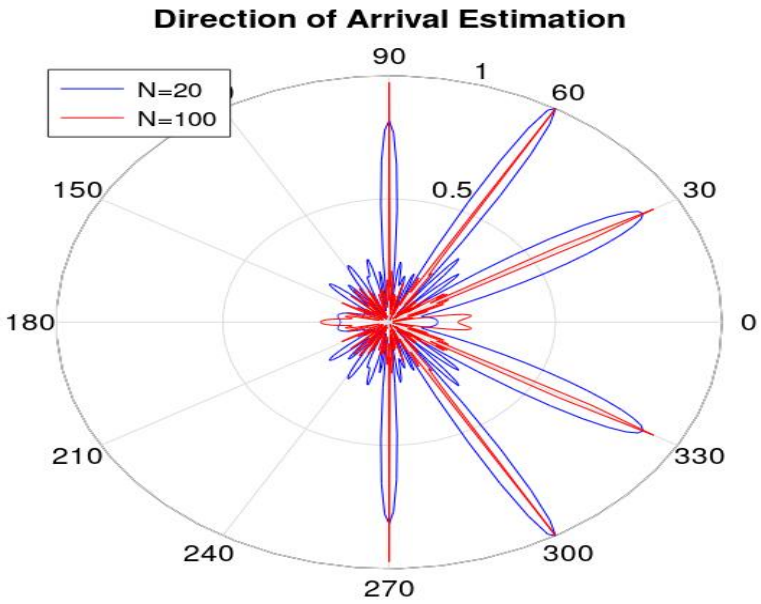


Figure 7 Sound Angle Measuring

What is Hilbert Transformation?

The Hilbert Transform is a mathematical technique used to extract the instantaneous phase and frequency information from a signal. It is named after David Hilbert, a German mathematician who first proposed the concept in the late 19th century.

The Hilbert transform is used to obtain an analytical signal from a real-valued signal. An analytical signal is a complex-valued signal that contains both amplitude and phase information of the original signal. The analytical signal is obtained by applying the Hilbert transform to the original signal.

The Hilbert transform of a signal x(t) can be defined as the convolution of x(t) with an imaginary unit times the negative of the sign function and is mathematically represented as:

$$H(x(t)) = x(t) * (-i * sgn(t))$$

Where i is the imaginary unit and sgn(t) is the sign function.

The most common application of the Hilbert Transform is in signal processing, particularly in communication systems and control systems. It is also widely used in signal analysis, such as in the study of speech and audio signals, as well as in the analysis of biomedical signals.

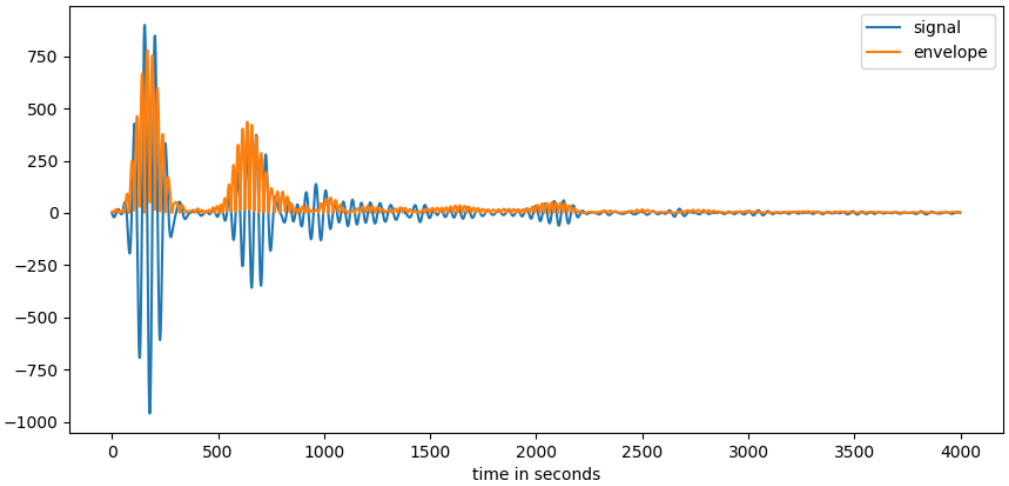


Figure 11 Hilbert Transformation