Digital Audio Signal Processing: Bandpass Filtering, Echo, and Robotic Voice Conversion

Fatemeh Amirabadi Zadeh

1 Introduction

This project explores digital audio signal processing techniques, including bandpass filtering, echo effects, and robotic voice conversion. The implementation is done using Python.

2 Recording Audio

We recorded an audio file where we count from one to ten using the following Python script:

Listing 1: Audio Recording Code

```
import sounddevice as sd
import numpy as np
import scipy.io.wavfile as wavfile

def record_audio(duration=10, fs=44100, channels=2, filename="recorded
    print(f"Recording for {duration} seconds...")
    recording = sd.rec(int(duration * fs), samplerate=fs, channels=cha
    sd.wait()
    wavfile.write(filename, fs, recording)
    print(f"Audio saved to: {filename}")

ecord_audio()
```

3 Signal Analysis and Filtering

3.1 Time-Domain Analysis

The recorded audio is loaded and plotted in the time domain. This plot shows the amplitude variation over time.

Observation: The time-domain plot displays the variations in amplitude over time, representing the loudness of the recorded speech.

3.2 Frequency-Domain Analysis

The frequency content of the audio signal is analyzed using the Fourier Transform. The two-channel system allows stereo sound representation.

Observation: The frequency-domain plots show the magnitude of different frequency components in the audio signal. The use of two channels enables stereo sound, which enhances spatial perception.

4 Bandpass Filtering

We apply a bandpass filter to retain frequencies between 500 Hz and 2000 Hz:

Listing 2: Bandpass Filtering Code

```
from scipy.signal import butter, lfilter

def apply_bandpass_filter(data, fs, lowcut=500, highcut=2000):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(4, [low, high], btype='band')
    return lfilter(b, a, data)
```

Observations: - The bandpass filter attenuates frequencies outside the range of 500 Hz to 2000 Hz. - Low-frequency components (e.g., background noise) and high-frequency components (e.g., sharp sounds) are suppressed.

5 the original signal

Image Placeholder:

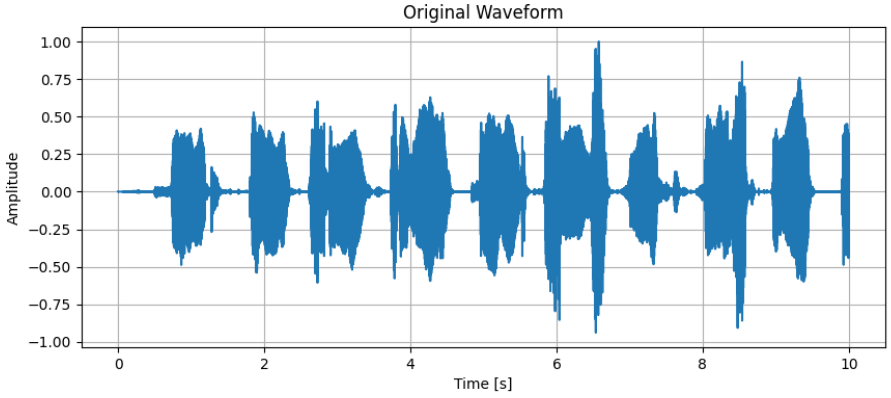

Figure 4: Echo Effect on the Signal

6 Applying Echo Effect

The following code introduces an echo effect:

Listing 3: Echo Effect Code

```
def apply_echo(data, fs, delay=0.2, attenuation=0.5):
    delay_samples = int(delay * fs)
    echoed_data = np.zeros_like(data)
    echoed_data[:delay_samples] = data[:delay_samples]
    echoed_data[delay_samples:] = data[delay_samples:] + attenuation *
    return echoed_data
```

Observations: The echo effect introduces delayed versions of the signal, creating a reverberation effect.

Image Placeholder:

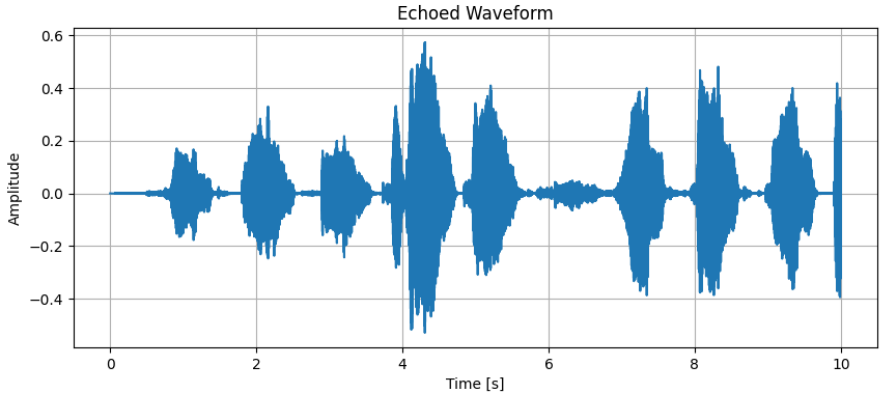

Figure 4: Echo Effect on the Signal

7 Robotic Voice Conversion

The robotic voice effect is implemented using amplitude modulation:

Listing 4: Robotic Voice Code

```
def apply_amplitude_modulation(data, fs, mod_freq=200):
    t = np.arange(len(data)) / fs
    carrier = 0.5 * np.sin(2 * np.pi * mod_freq * t)
    return data * carrier
```

Observations: Amplitude modulation alters the spectral content of the signal, introducing robotic-like distortions.

Image Placeholder:

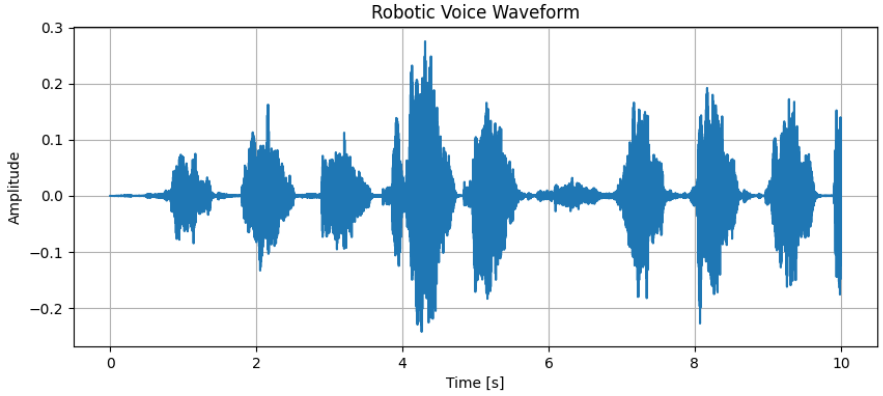

Figure 5: Robotic Voice Effect on the Signal

8 Conclusion

This project demonstrated digital audio processing techniques, including filtering, echo effects, and robotic voice conversion. The bandpass filter removed unwanted frequencies, the echo effect added reverberation, and amplitude modulation created a robotic sound.