Audio Processing

February 20, 2025

1 Assignment 03 Audio Processing

The objective of this project is to process a selected audio file from the data folder by removing sections of silence longer than 0.5 seconds and then applying an echo effect to the remaining audio. This involves:

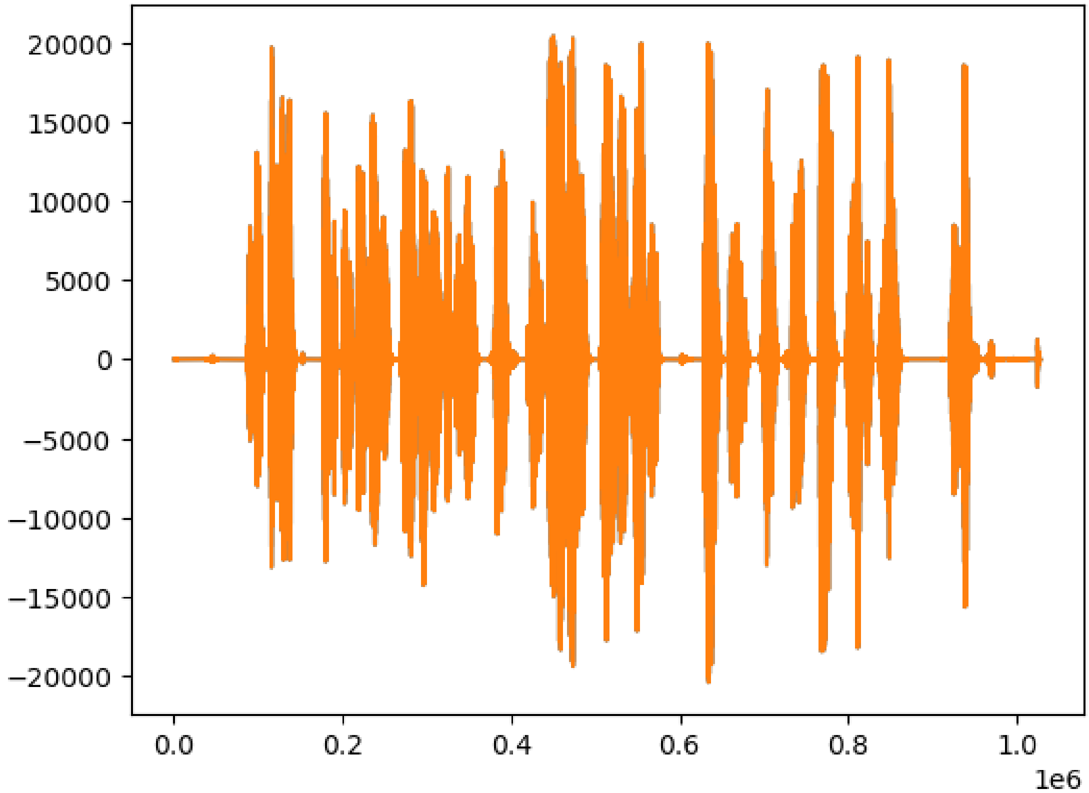
- Audio File Selection: Choosing a specific audio file from the designated data folder for analysis and manipulation.
- Silence Removal: Identifying and eliminating segments of the audio that contain silence lasting longer than 0.5 seconds to enhance the overall listening experience.
- Echo Effect Application: Adding an echo effect to the processed audio to create a richer and more engaging sound.

[1]:

```
# your code
import matplotlib.pyplot as mpl
from scipy.io.wavfile import read
import numpy as np

# Read the sound file and get the sampling rate and audio data
a = read("data/my_second_example.wav")
samplingRate = a[0]
audioData = a[1]

# Plot the audio data
mpl.plot(audioData);
```



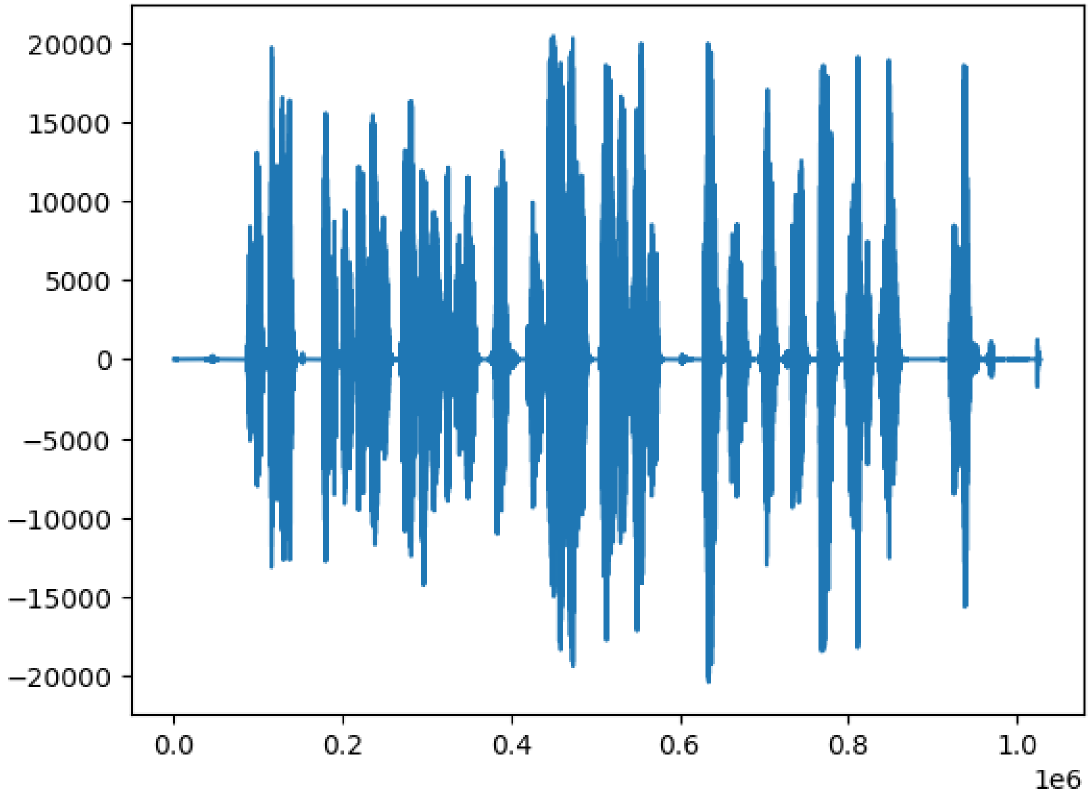
[2]:
```
# Extract the left and right audio channels from the audio data
left_channel = audioData[:, 0]
right_channel = audioData[:, 1]

# Check if the channels are the same
are_same = np.array_equal(left_channel, right_channel)

# Print a message based on whether the channels are the same or different
if are_same:
    print("The left and right channels are the same.")
else:
    print("The left and right channels are different.")
```

```
The left and right channels are the same.
```

[3]:
```
#plot only the left channel
mpl.plot(left_channel);
```



[4]:
```
#Listen to audio from within the notebook using: IPython.display.Audio
from IPython import display
display.Audio(left_channel, rate = samplingRate)
```

[4]: <IPython.lib.display.Audio object>

[5]:
```
# Your code here
import random

# create noise (we have done it in the last lecture)
N_max = len(audioData) # Number of samples
noise = np.random.rand(N_max) * left_channel.max()

# write audioWithNoise (sum audio with noise!)
audioWithNoise = noise + left_channel
```

[6]:
```
from scipy.io.wavfile import write

# Store the noisy version as a wav file
scaled = np.int16(audioWithNoise/np.max(np.abs(audioWithNoise)) * 32767)
write('data/audio_with_noise.wav', 44100, scaled)
```

[7]:
```python
# Create an audio widget to play the scaled audio at the specified sampling rate
display.Audio(scaled, rate = samplingRate)
```

[7]: <IPython.lib.display.Audio object>

[8]:
```python
# Trim the quiet parts by setting values below a threshold to 0
mapped = list(map(lambda data : data if data > 750 or data < -750 else 0,␣
 ↪list(left_channel)))
arr = np.array(mapped)

# Iterate through the audio data to identify regions of sound
regions = []
consec = 0
for i, e in enumerate(arr):
    if e == 0 and consec == 0:
        start = i
        consec = 1
    elif e != 0 and consec == 1:
        regions.append((start, i-1))
        consec = 0

# Remove any pause longer than a half second
threshold_samples = int(0.5 * samplingRate)

# Filter out regions shorter than the threshold
regions = [x for x in regions if x[1]-x[0] > threshold_samples]
```

[9]:
```python
# Loop through the identified regions in reverse order to delete quiet sections
result = left_channel
for x in regions[::-1]:
    result = np.delete(result, np.s_[x[0]:x[1]])

# Scale the audio recording for output
scaled = np.int16(result/ np.max(np.abs(result)) * 32767)

# Specify the output file path for the new audio
output_file = 'data/new_audio_output.wav'
write(output_file, samplingRate, scaled)

# Create an audio widget to play the trimmed audio at the specified sampling␣
 ↪rate
display.Audio(result, rate = samplingRate)
```

[9]: <IPython.lib.display.Audio object>

[10]:
```python
# Create an echo effect
base = np.concatenate([result, np.zeros(int(0.2 * samplingRate))])
```

```
# Create the echo by adding a delayed version of the original audio
echo = np.concatenate([np.zeros(int(0.2 * samplingRate)), result])

# Combine the original audio with the echoed version
echoed = echo + base

# Specify the output file path for the new audio
output_file = 'data/new_audio_output.wav'
write(output_file, samplingRate, scaled)

# Display the echoed audio for playback
display.Audio(echoed, rate = samplingRate)
```

[10]: <IPython.lib.display.Audio object>

[11]:
```
!pip3 install SpeechRecognition pydub
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: SpeechRecognition in
/home/izjiang/.local/lib/python3.11/site-packages (3.14.1)
Requirement already satisfied: pydub in
/home/izjiang/.local/lib/python3.11/site-packages (0.25.1)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.11/site-packages (from SpeechRecognition) (4.11.0)
```

[12]:
```
import speech_recognition as sr

# Create a recognizer instance
r = sr.Recognizer()
```

[13]:
```
# Load the audio file for processing
with sr.AudioFile("data/my_second_example.wav") as source:
    # Listen for the data (load audio to memory)
    audio_data = r.record(source)

    # Recognize (convert from speech to text)
    text = r.recognize_google(audio_data)
    print(text)
```

```
hello everybody this is an example of an audio recording for DSC 961234567 stop
```

Summary of Findings

The project successfully processed an audio file by implementing the following key actions:

1. Silence Removal: Sections of the audio containing silence longer than 0.5 seconds were effectively identified and removed. This was achieved by setting values below a specified threshold

to zero and iterating through the audio data to find regions of sound. The resulting audio was trimmed to enhance clarity and maintain listener engagement.

2. Echo Effect Application: An echo effect was added to the processed audio. This involved creating a delayed version of the original audio and combining it with the trimmed audio. The echo effect enriched the sound, providing a more immersive listening experience.
3. Output Generation: The processed audio, both trimmed and echoed, was scaled to ensure it remained within a suitable range for playback. The final audio was then saved to a new WAV file, allowing for easy access and playback.
4. Playback Capability: An audio widget was created to facilitate the playback of both the trimmed and echoed audio, enabling immediate review of the modifications made.