Periféricos y Dispositivos de Interfaz Humana
Práctica 5

UNIVERSIDAD DE GRANADA

Realizado por :
Sergio España Maldonado
Jose Luis Molina Aguilar

1. Creación de los ficheros wav.

Para la creación de estos archivos utilizamos el sintetizador de voz “espeak” de la siguiente manera, usando el comando:

- espeak “texto que deseamos reproducir entre las comillas” -w nombre_del_archivo.wav

2. Leer dos ficheros de sonido (WAV o MP3) de unos pocos segundos de duración cada uno. En el primero debe escucharse el nombre de la persona que realiza la práctica. En el segundo debe escucharse el apellido.

Aquí usamos la la primera parte de la función print_audio_signal(file_name, title) con la que al principio realiza wave.open() y abre el archivo deseado

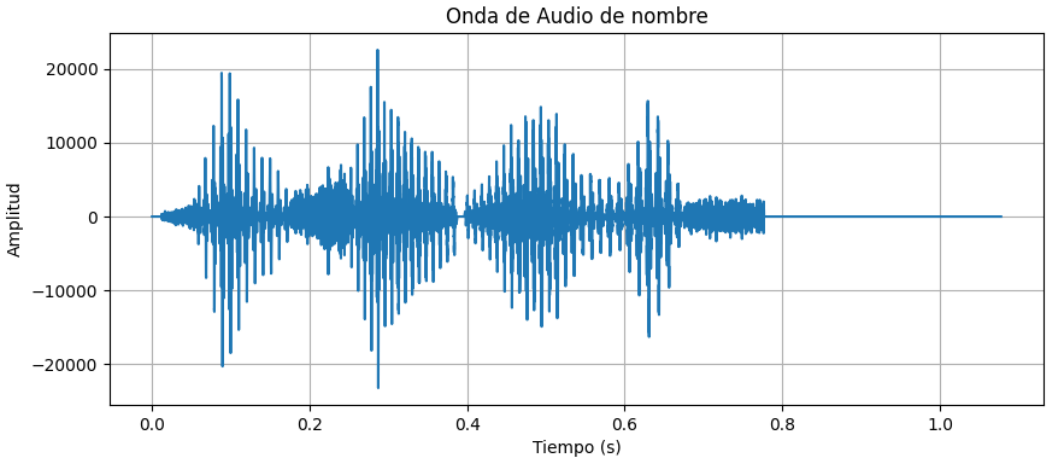
```
def print_audio_signal(file_name, title):
    wav_file = wave.open(file_name, 'rb')
    sample_width = wav_file.getsampwidth()
    sample_rate = wav_file.getframerate()
    num_frames = wav_file.getnframes()
    audio_data = wav_file.readframes(num_frames)
    wav_file.close()
```

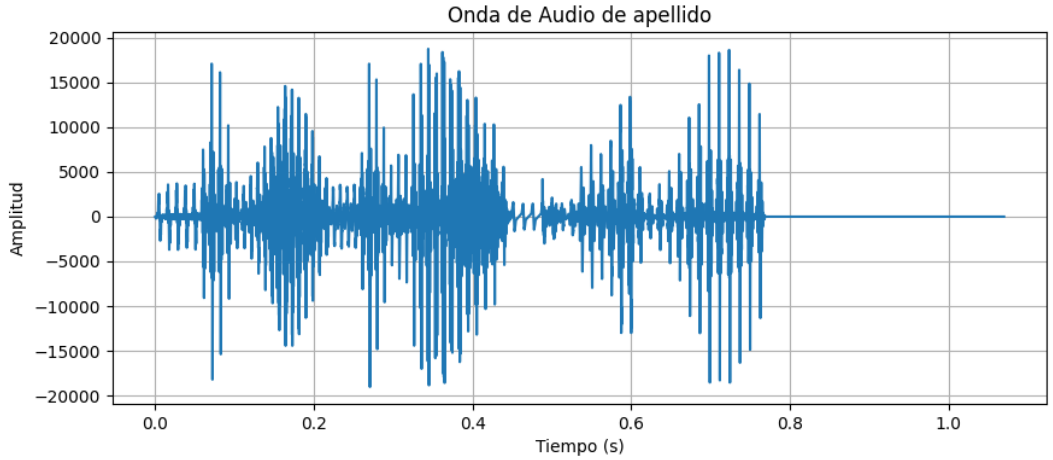
3. Dibujar la forma de onda de ambos sonidos.

Aquí simplemente usamos la función que nombramos en el apartado anterior en la que acabara de pintarnos las ondas quedando de la siguiente manera:

```
def print_audio_signal(file_name, title):
    wav_file = wave.open(file_name, 'rb')
    sample_width = wav_file.getsampwidth()
    sample_rate = wav_file.getframerate()
    num_frames = wav_file.getnframes()
    audio_data = wav_file.readframes(num_frames)
    wav_file.close()
    audio_np = np.frombuffer(audio_data, dtype=np.int16)
    duration = num_frames / sample_rate
    time = np.linspace(0, duration, num=len(audio_np))
    plt.figure(figsize=(10, 4))
    plt.plot(time, audio_np)
    plt.xlabel('Tiempo (s)')
    plt.ylabel('Amplitud')
    plt.title(f'Onda de Audio de {title}')
    plt.grid(True)
    plt.show()
```

```
################## Ejercicios 1 y 2 ##########################################
# Leemos dos archivos wav, nombre y apellido y mostramos sus ondas de sonido
##############################################################################

print_audio_signal(file_name, 'nombre')
print_audio_signal(file_surname, 'apellido')
```





4. **Obtener la información de las cabeceras de ambos sonidos.**

   Aquí usando la función get_info_from_audio(file_name) que nos imprimirá la información deseada.

   Código:

```
55  def get_info_from_audio(file_name):
56      info = sf.info(file_name)
57      print("-------------------------------------\n")
58      print(info)
59      print("-------------------------------------\n")
60
```

   Resultado:

```
nombre.wav
samplerate: 22050 Hz
channels: 1
duration: 1.078 s
format: WAV (Microsoft) [WAV]
subtype: Signed 16 bit PCM [PCM_16]
-------------------------------------


-------------------------------------

apellidos.wav
samplerate: 22050 Hz
channels: 1
duration: 1.070 s
format: WAV (Microsoft) [WAV]
subtype: Signed 16 bit PCM [PCM_16]
-------------------------------------
```

5. Unir ambos sonidos en uno nuevo.

Aquí usaremos la función que disponemos para combinar los dos audios que es la llamada combine_audio_file(file_name, file_name, output_file) quedando:

Código:

```
def combine_audio_files(file_name1, file_name2, output_file):
    data1, samplerate1 = sf.read(file_name1)
    data2, samplerate2 = sf.read(file_name2)

    if samplerate1 != samplerate2:
        raise ValueError("Las frecuencias de muestreo no coinciden")

    combined_data = np.concatenate((data1, data2))
    sf.write(output_file, combined_data, samplerate1)
```

6. Dibujar la forma de onda de la señal resultante

Aquí realizamos la misma operación que en el paso 3 quedándonos la siguiente imagen:

```
def combine_audio_files(file_name1, file_name2, output_file):
    data1, samplerate1 = sf.read(file_name1)
    data2, samplerate2 = sf.read(file_name2)

    if samplerate1 != samplerate2:
        raise ValueError("Las frecuencias de muestreo no coinciden")

    combined_data = np.concatenate((data1, data2))
    sf.write(output_file, combined_data, samplerate1)
```

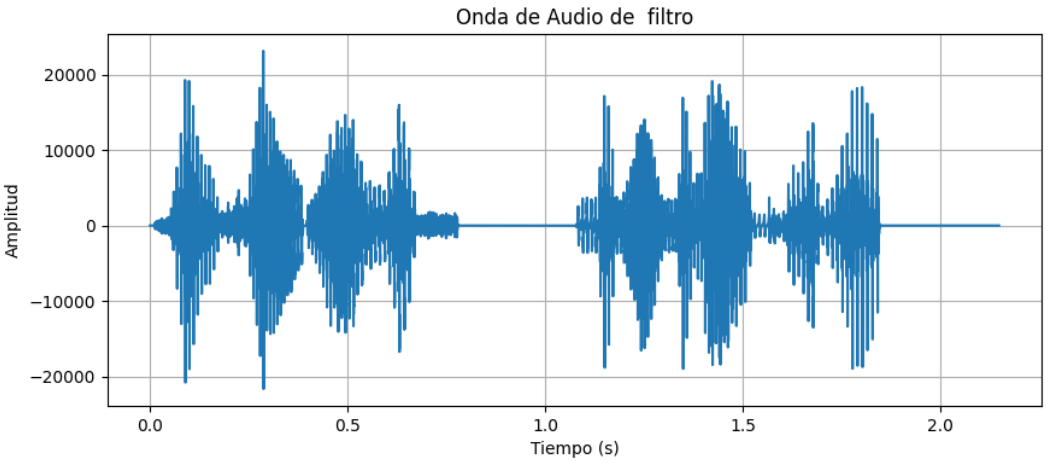
7. Pasarle un filtro de frecuencia para eliminar las frecuencias entre 10000Hz y 20000Hz

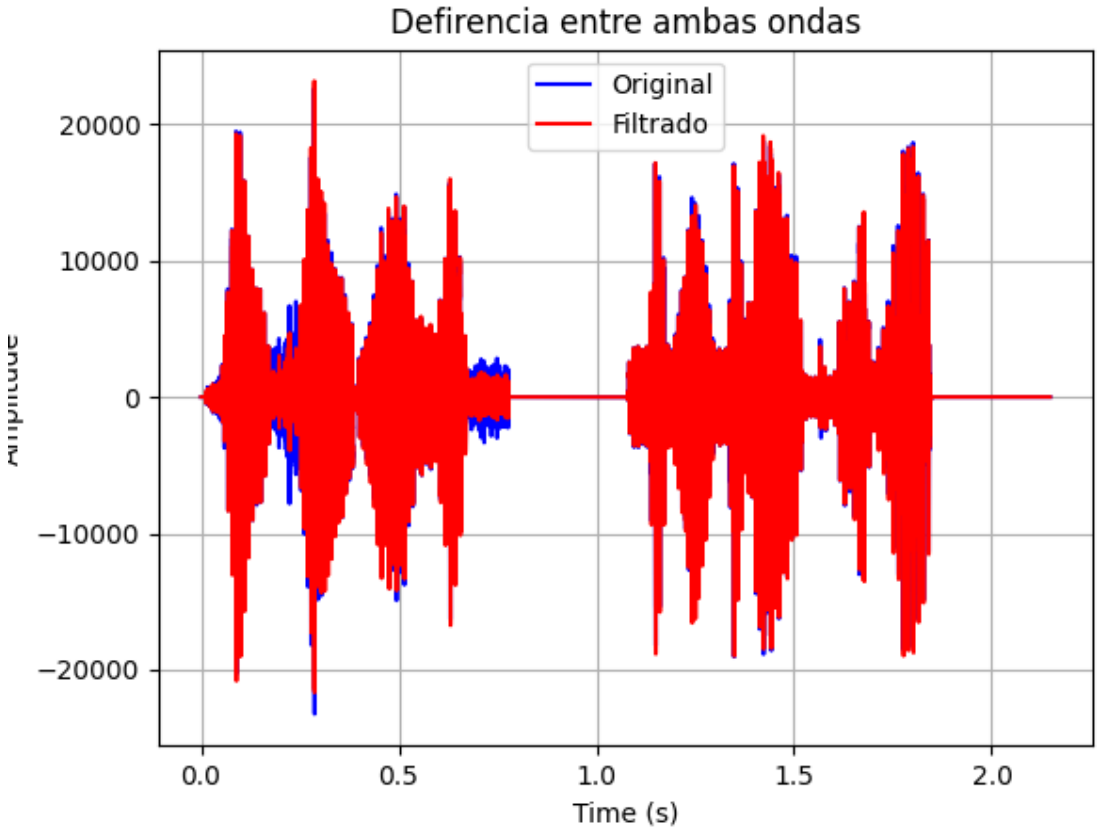
Para este paso usaremos la función aplicar_filtro_frecuencia() y posteriormente la representaremos, también para que se aprecie el cambio creamos la función plot_dual_waveforms() con la que podremos superponer ambas frecuencias y se apreciará la diferencia que hay quedando:

Ambos códigos:

```
def plot_dual_waveforms(file1, file2):
    # Cargar los archivos WAV
    sample_rate1, waveform1 = wavfile.read(file1)
    sample_rate2, waveform2 = wavfile.read(file2)

    # Calcular los ejes de tiempo para ambas formas de onda
    duration1 = len(waveform1) / sample_rate1
    duration2 = len(waveform2) / sample_rate2
    time1 = np.linspace(0., duration1, len(waveform1))
    time2 = np.linspace(0., duration2, len(waveform2))

    # Crear la figura y los ejes
    fig, ax = plt.subplots()

    # Trazar la primera forma de onda en color azul
    ax.plot(time1, waveform1, color='blue', label='Original')

    # Trazar la segunda forma de onda en color rojo
    ax.plot(time2, waveform2, color='red', label='Filtrado')

    # Configurar etiquetas y título del gráfico
    ax.set_xlabel('Time (s)')
    ax.set_ylabel('Amplitude')
    ax.set_title('Defirencia entre ambas ondas')
    plt.grid(True)
    ax.legend()
    plt.show()


def aplicar_filtro_frecuencia(input_file, output_file, freq_min, freq_max):
    # Cargar el archivo WAV
    sample_rate, data = wavfile.read(input_file)
    freq_min /= sample_rate
    freq_max /= sample_rate

    b, a = signal.butter(8, [freq_min, freq_max], btype='bandstop')

    # Aplicar el filtro a los datos
    filtered_data = signal.lfilter(b, a, data)

    # Guardar los datos filtrados en un nuevo archivo WAV
    wavfile.write(output_file, sample_rate, np.int16(filtered_data))
```

Resultado:


Onda de Audio de filtro
Amplitud
Tiempo (s)
20000
10000
0
−10000
−20000
0.0
0.5
1.0
1.5
2.0


Defirencia entre ambas ondas
Original
Filtrado
Amplitude
Time (s)
20000
10000
0
−10000
−20000
0.0
0.5
1.0
1.5
2.0

8. **Cargar un nuevo archivo de sonido, aplicarle eco y a continuación darle la vuelta al sonido. Almacenar la señal obtenida como un fichero WAV denominado “alreves.wav”.**

En este último paso usaremos la función :
process_audio_with_echo_and_reverse():

Código:

```
def add_echo(input_audio, echo_delay=0.5, echo_gain=0.6):
    sample_rate, audio = wavfile.read(input_audio)

    echo_samples = int(echo_delay * sample_rate)

    echo_signal = np.zeros_like(audio)
    echo_signal[echo_samples:] = audio[:-echo_samples] + echo_gain * audio[echo_samples:]

    return echo_signal, sample_rate

def reverse_audio(input_audio):
    sample_rate, audio = wavfile.read(input_audio)
    reversed_signal = np.flip(audio)

    return reversed_signal, sample_rate

def process_audio_with_echo_and_reverse(input_file, output_file, echo_delay=0.5, echo_gain=0.6):
    # Aplicar eco
    echo_signal, sample_rate = add_echo(input_file, echo_delay, echo_gain)
    sf.write('eco.wav', echo_signal, sample_rate)
    # Invertir el sonido
    reversed_signal, _ = reverse_audio('eco.wav')

    # Guardar la señal invertida como un archivo WAV
    sf.write(output_file, reversed_signal, sample_rate)
    print("Procesamiento completado. Archivo guardado como:", output_file)
```