Trabajo Laboratorio: Sistemas Multimedia- Audio II- (Codificación señales de audio)

Una vez estudiados en detalle (práctica anterior) los fundamentos de los elementos utilizados para la implementación de los códec de audio, en esta práctica vamos a hacer uso de dichos conceptos editando aplicaciones de audio que nos permitan manejarlos.

Concretamente vamos a realizar dos tipos de aplicaciones:

1. Código fuente Python.
2. Grabador y editor de audio, Audacity.

Para ello tendremos que instalarnos el entorno de desarrollo Python (e incorporar las librerías: scipy, matplotlib, sounddevice, y soundfile), y la aplicación Audacity.

Una vez completadas estas instalaciones, realizaremos las siguientes tareas:

1. (Python) - Generación de una armonía resultado de la suma de tres cosenos (frec.: 261, 329, y 392 Hz), guardando el resultado en un fichero “wav”.(duración 5 seg.)
2. (Python) – Grabación de un mensaje de prueba de voz en un fichero “wav”. (duración 5 seg.)
3. (Python) – Reproducción del fichero anterior de audio.
4. (Python) – Creación de un fichero “wav” con el contenido invertido del mensaje resultado de la tarea 2.
5. (Python) – Representación (amplitud y tiempo) de la señal grabada en la tarea 2.
6. (Audacity) – Reproducción de los ficheros grabados en las tareas 1 y 2, identificando la codificación PCM y frecuencia de muestreo empleadas en su grabación. Compara la señal obtenida al abrir el fichero de la grabación de la tarea 2, con la señal obtenida en la tarea 5.
7. (Audacity) – Creación de los contenidos invertidos de los ficheros anteriores (tareas 1 y 2).
8. (Audacity) – Graba un nuevo audio y aplícale un par de efectos proporcionados por audacity.

Vídeo: graba un video de 2 o 3 minutos de duración en el que muestras el análisis de estas tareas. Nombra tu vídeo *V_Mod_03_B_apellido.mv4.* (o extensión de compresión semejante)

Tarea 1: (Python) - Generación de una armonía resultado de la suma de tres cosenos (frec.: 261, 329, y 392 Hz), guardando el resultado en un fichero “wav”. (duración 5 seg.)

Screenshot 1:

```
import numpy as np
import sounddevice as sd
from matplotlib import pyplot as plt
from scipy.io.wavfile import write

SAMPLE_RATE = 44100
DURATION = 5

# Función para generar una onda cosenoidal
def generate_cosine_wave(freq, sample_rate, duration):
    t = np.linspace(0, duration, sample_rate * duration, endpoint=False)
    y = np.cos(2 * np.pi * freq * t)  # Error corregido en 'np.pi' en lugar de 'np.pl'
    return y

# Generación de las ondas individuales
y261 = generate_cosine_wave(261, SAMPLE_RATE, DURATION)
y329 = generate_cosine_wave(329, SAMPLE_RATE, DURATION)
y392 = generate_cosine_wave(392, SAMPLE_RATE, DURATION)

# Suma de las ondas
y_total = y261 + y329 + y392

# Normalización y conversión a enteros de 16 bits para WAV
normalized_y = np.int16((y_total / np.max(np.abs(y_total))) * 32767)  # Error corregido en 'np.int16'

# Guardar el archivo de audio
write("y_total.wav", SAMPLE_RATE, normalized_y)
```

Breve descripción:

En esta tarea se genera una señal de audio que es la suma de tres ondas cosenoidales de diferentes frecuencias (261, 329 y 392 Hz), creando una armonía que se guarda en un archivo WAV de 5 segundos de duración.

Tarea 2: (Python) – Grabación de un mensaje de prueba de voz en un fichero “wav”. (duración 5 seg.)

Screenshot 2:

```
# pip install sounddevice

import sounddevice as sd
from scipy.io.wavfile import write
import os

# Parámetros de grabación
fs = 44100  # Frecuencia de muestreo
seconds = 5  # Duración de la grabación

print("Habla durante 5 segundos")
myrecording = sd.rec(int(seconds * fs), samplerate=fs, channels=2)
sd.wait()  # Espera a que la grabación termine
print("Grabación terminada")

# Guardar la grabación en un archivo WAV
write("grabacion_24.wav", fs, myrecording)

# Abrir el archivo en Ubuntu (o cualquier Linux que soporte xdg-open)
os.system("xdg-open grabacion_24.wav")

ej2.py (END)
```

Breve descripción:

Se graba un mensaje de prueba en formato WAV, con una duración de 5 segundos, utilizando un micrófono conectado al sistema. Esto permite captar y guardar una señal de audio en un archivo.

Tarea 3: (Python) – Reproducción del fichero anterior de audio.

Screenshot 3:

```
import sounddevice as sd
import soundfile as sf

filename = "grabacion_24.wav"

# Cargar el archivo de audio
data, fs = sf.read(filename)

# Reproducir el archivo de audio
sd.play(data, fs)
sd.wait()  # Espera a que termine la reproducción

(END)
```

Breve descripción:

Se reproduce el archivo WAV creado en la tarea anterior para verificar la grabación y su calidad de sonido.

Tarea 4: (Python) – Creación de un fichero "wav" con el contenido invertido del mensaje resultado de la tarea 2.

Screenshot 4:

```
import sounddevice as sd
import soundfile as sf
from scipy.io.wavfile import write

filename = "grabacion_24.wav"

# Leer el archivo de audio
data, fs = sf.read(filename)
print(type(data))

# Invertir el contenido del archivo de audio
reverse_data = data[::-1]

# Reproducir el archivo invertido
sd.play(reverse_data, fs)
sd.wait()  # Esperar a que termine la reproducción

# Guardar el archivo invertido en un nuevo archivo WAV
write('REV_grabacion_24.wav', fs, reverse_data)
ej4.py (END)
```

Breve descripción:

Esta tarea consiste en cargar la señal grabada en la tarea 2, invertir el contenido (es decir, reproducirlo de forma invertida) y guardarlo en un nuevo archivo WAV.

Tarea 5: (Python) – Representación (amplitud y tiempo) de la señal grabada en la tarea 2.

Screenshot 5:

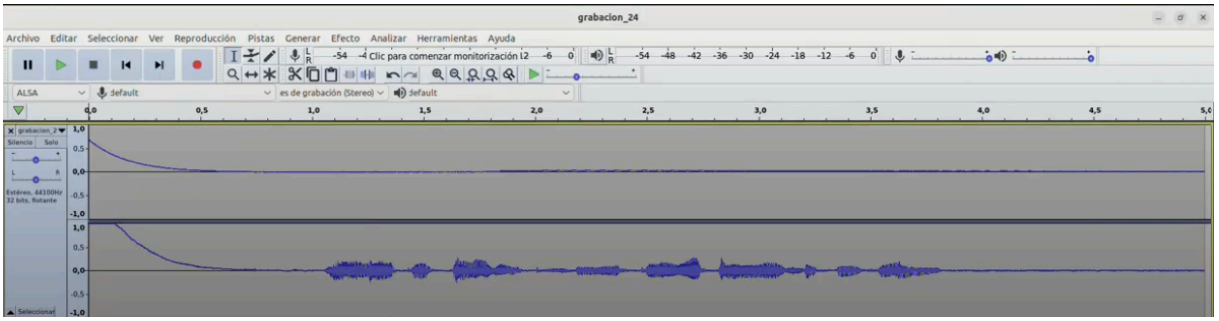
```
import matplotlib.pyplot as plt
import numpy as np
import soundfile as sf

filename = "grabacion_24.wav"

# Leer el archivo de audio
data, fs = sf.read(filename)

# Si el audio es estéreo, selecciona el primer canal
data0 = data[:, 0] if data.ndim > 1 else data

# Crear el vector de tiempo
t = np.linspace(0, len(data0) / fs, num=len(data0))

# Crear la gráfica
plt.figure()
plt.title("Sound Wave")
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")

# Graficar la señal
plt.plot(t, data0)
plt.show()

(END)
```

Breve descripción:

Se realiza una representación gráfica de la señal grabada en la tarea 2, mostrando la amplitud en función del tiempo para analizar visualmente la forma de la onda de audio.

Tarea 6: (Audacity) – Reproducción de los ficheros grabados en las tareas 1 y 2, identificando la codificación PCM y frecuencia de muestreo empleadas en su grabación. Compara la señal obtenida al abrir el fichero de la grabación de la tarea 2, con la señal obtenida en la tarea 5.

Screenshot 6:

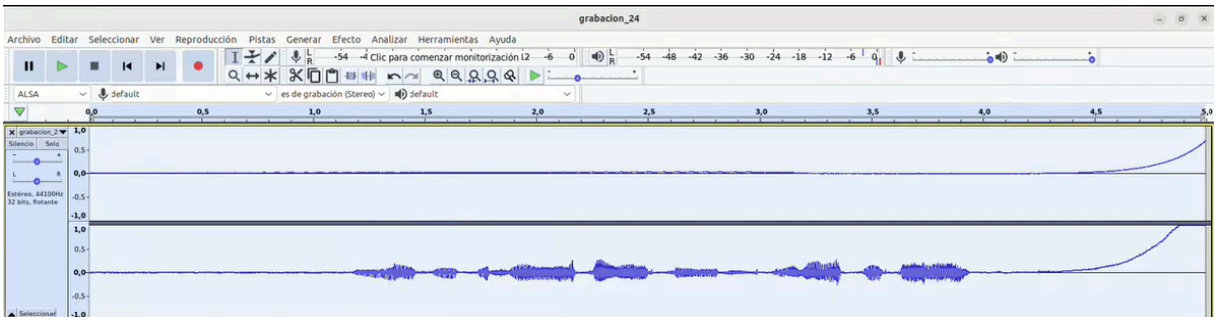


Breve descripción:

En esta tarea, se abren y reproducen en Audacity los archivos generados en las tareas 1 y 2, identificando la codificación PCM y la frecuencia de muestreo, y comparando la forma de onda del archivo original con la gráfica obtenida en Python.

Tarea 7: (Audacity) – Creación de los contenidos invertidos de los ficheros anteriores (tareas 1 y 2).

Screenshot 7:

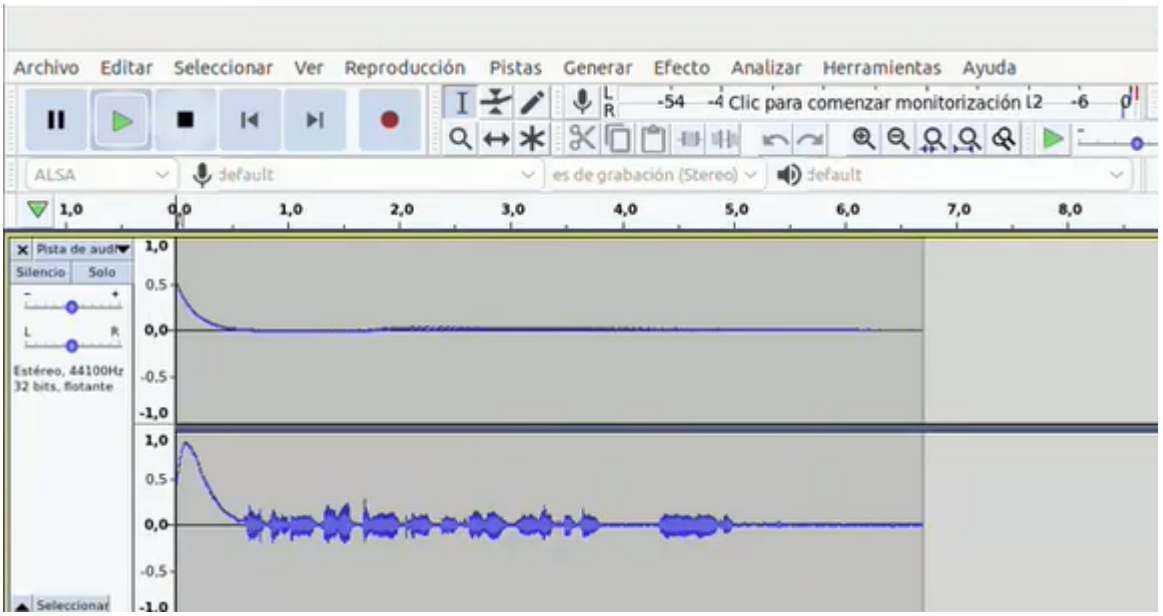


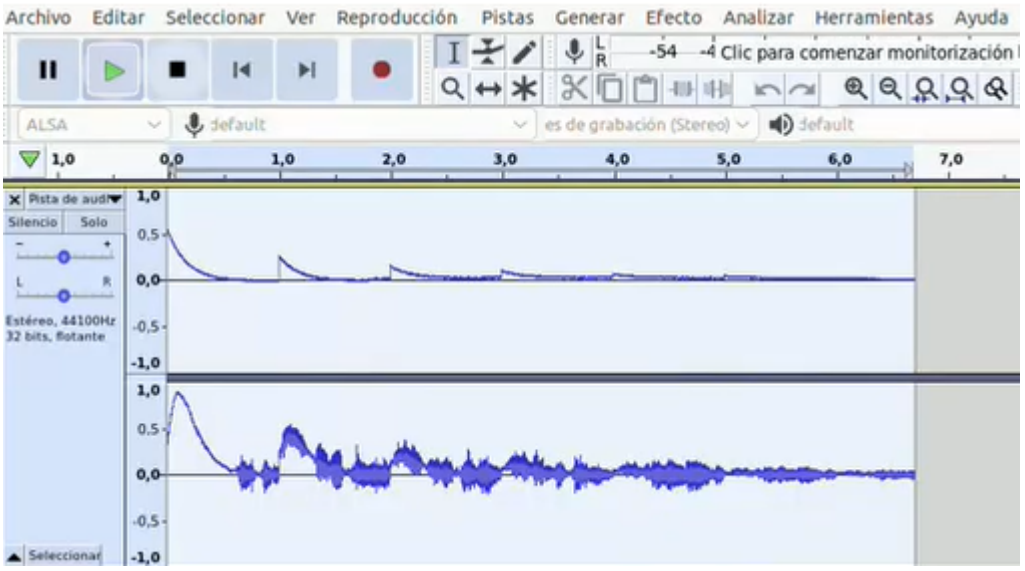
Breve descripción:

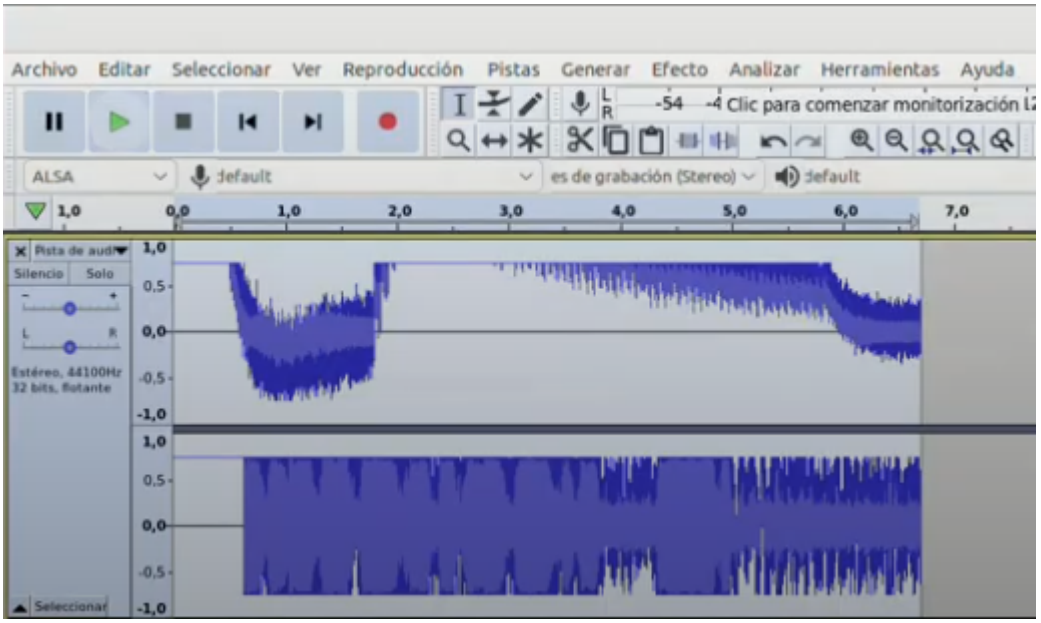
Utilizando las herramientas de edición de Audacity, se invierten los contenidos de los archivos generados en las tareas 1 y 2, verificando el resultado.

Tarea 8: (Audacity) – Graba un nuevo audio y aplícale un par de efectos proporcionados por audacity.

Screenshot 8:







Breve descripción:

Se graba un nuevo archivo de audio en Audacity y se aplican dos efectos, como eco y cambio de tono, para experimentar con las opciones de edición de audio que ofrece la herramienta.