Audio Processing

Mateo Cardona Arias

Agosto 2025

1. Introducción

El procesamiento de audio es una rama de la ingeniería que se encarga de analizar, modificar y representar señales acústicas en el dominio digital. Este tipo de técnicas permiten eliminar ruido, detectar patrones, identificar características como la frecuencia dominante, e incluso comprimir y transmitir señales de forma más eficiente. Su importancia radica en aplicaciones prácticas como el reconocimiento de voz, la música digital, la compresión de audio y la mejora de la calidad en comunicaciones.

Adicionalmente, se realizó una presentación en video para ilustrar el funcionamiento del proyecto. Dirección del video: `https://youtu.be/T5MXE1Jc6jw`

2. Archivo de Audio

Mediante las siguientes funciones en Python:

```
y, sr = librosa.load(audio_file, sr=None)
duration = librosa.get_duration(y=y, sr=sr)
n_channels = 1 if y.ndim == 1 else y.shape[1]
```

El archivo `audio.wav` fue analizado, y presento las siguientes características que fueron calculadas mediante :

- Frecuencia de muestreo: 44100 Hz.
- Duración: 8.1 segundos.
- Número de canales: 1 (mono).

3. Código Desarrollado

A continuación se muestra el código en Python utilizado para realizar el procesamiento:

```
import librosa
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq

audio_file = "audio.wav"
y, sr = librosa.load(audio_file, sr=None)
duration = librosa.get_duration(y=y, sr=sr)
n_channels = 1 if y.ndim == 1 else y.shape[1]

non_silent_intervals = librosa.effects.split(y, top_db=20)
if len(non_silent_intervals) > 0:
    filtered_audio = np.concatenate([y[start:end] for start, end in non_sil
else:
    filtered_audio = y

desired_sample_rate = 8000
bit_depth = 8
```

```
resampled_audio = librosa.resample(filtered_audio, orig_sr=sr, target_sr=de
normalized_audio = resampled_audio / np.max(np.abs(resampled_audio))
max_amplitude = 2**(bit_depth - 1) - 1
quantized_audio = np.round(normalized_audio * max_amplitude).astype(np.int1

N = len(filtered_audio)
T = 1.0 / sr
yf = fft(filtered_audio)
xf = fftfreq(N, T)[:N//2]
amplitude = 2.0/N * np.abs(yf[0:N//2])

eps = 1e-10
power_watts = (amplitude / np.sqrt(2))**2 / 1.0
power_mw = power_watts * 1000
amplitude_dbm = 10 * np.log10(power_mw + eps)

mask = (xf >= 0) & (xf <= 400)
xf_plot = xf[mask]
amplitude_dbm_plot = amplitude_dbm[mask]

dominant_freq = xf_plot[np.argmax(amplitude_dbm_plot)]
cumulative_power = np.cumsum(amplitude_dbm_plot - np.min(amplitude_dbm_plot
cumulative_power /= cumulative_power[-1]

low_idx = np.argmax(cumulative_power >= 0.05)
high_idx = np.argmax(cumulative_power >= 0.95)
freq_range = (xf_plot[low_idx], xf_plot[high_idx])
```

4. Resultados

Durante el análisis se obtuvieron los siguientes resultados:

- Frecuencia dominante: **201.24 Hz**.
- Rango de frecuencias significativas: **30.69 Hz – 382.43 Hz**.
- Intervalos no silenciosos detectados:

```
[[ 29184  73728]
 [ 75264 132096]
 [132608 144384]
 [145920 155648]
 [157184 169984]
 [177152 189440]
 [191488 217600]
 [219136 258560]
 [261120 267776]
 [269824 288768]
```

```
[294912 318976]
[319488 325632]
[327680 339968]]
```

A continuación, se presentan las gráficas obtenidas del análisis:

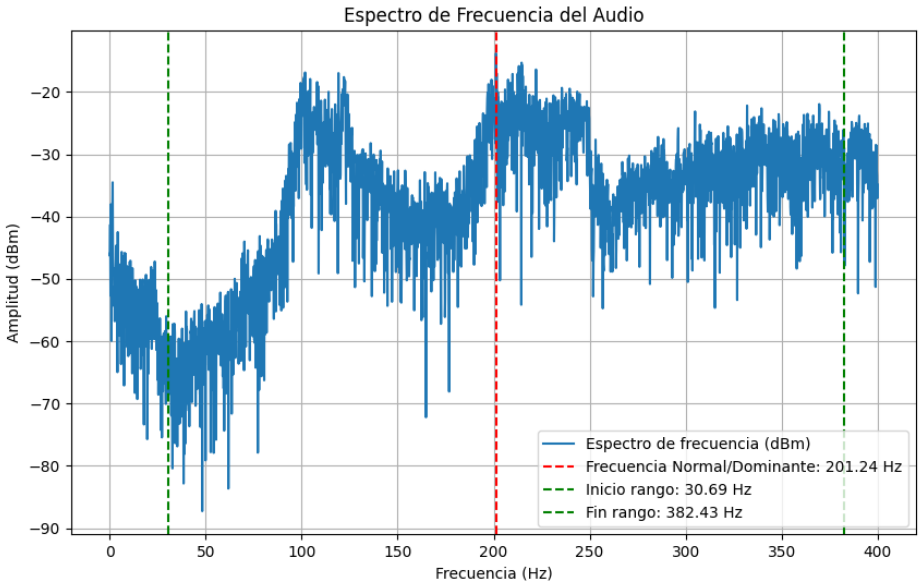

Figura 1: Espectro de frecuencia del audio.

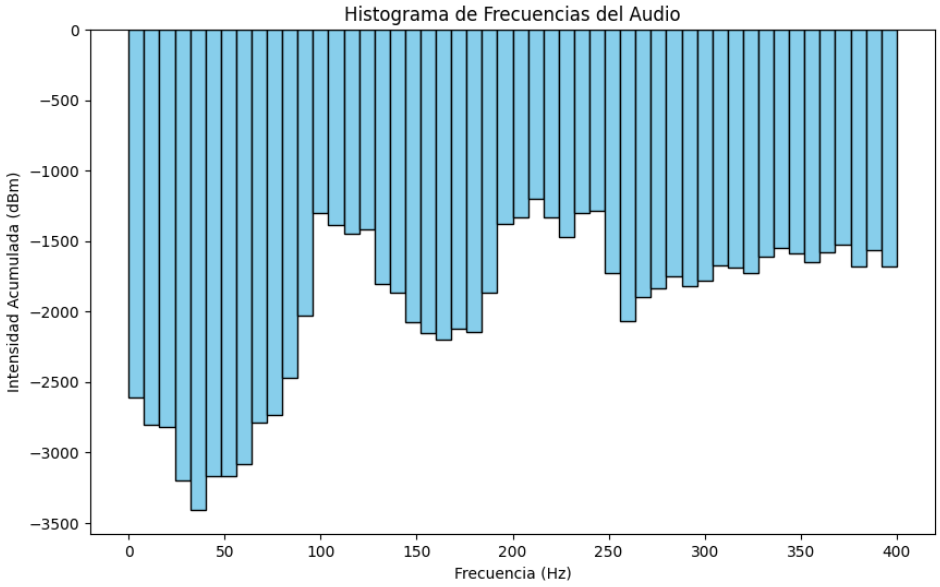

Figura 2: Histograma de frecuencias del audio.