Tarea 4: Procesamiento digital de señales

Alan Andrés Mérida Morales, 202100023[1, *]

[1] *Facultad de Ingeniería, Escuela de Ingeniería Mecánica Eléctrica, Universidad de San Carlos, Ciudad Universitaria, Zona 12, Guatemala.*

I. OBJETIVOS

- Desarrollar un programa para la captura, almacenamiento, reproducción y análisis de señales de audio.
- Implementar funcionalidades para la reproducción y visualización de señales de audio.
- Realizar análisis espectral de las señales de audio grabadas.
- Comparar la implementación del programa en dos entornos de programación diferentes: Octave y Python.
-

II. INTRODUCCIÓN

El procesamiento digital de señales (PDS) es una disciplina clave en la ingeniería, enfocada en la manipulación de señales digitales para aplicaciones como audio, imágenes y comunicaciones. Este reporte presenta el desarrollo de un sistema en Python para grabar, reproducir y analizar señales de audio, destacando conceptos fundamentales del PDS, como la digitalización de señales, la representación temporal y el análisis espectral. El proyecto ofrece una visión práctica de cómo se aplican estas técnicas para transformar y entender las señales en diferentes dominios.

III. MARCO TEÓRICO

El procesamiento digital de señales (PDS) es una técnica fundamental en la ingeniería y las ciencias aplicadas que se encarga de la manipulación de señales en formato digital. Una señal digital es una representación discreta en el tiempo y en la amplitud de una señal continua, como el audio, la imagen o los datos. El PDS permite realizar operaciones como filtrado, compresión, y análisis espectral de estas señales para extraer información útil y mejorar su calidad.

* 3690273450101@ingenieria.usac.edu.gt

A. Digitalización de Señales

La digitalización de una señal implica su conversión de una forma analógica a digital a través del muestreo y la cuantificación. El muestreo toma muestras discretas de la señal continua a intervalos regulares, mientras que la cuantificación convierte estas muestras en valores digitales. La frecuencia de muestreo determina la resolución temporal de la señal digitalizada.

B. Análisis en el Dominio Temporal y Frecuencial

El análisis en el dominio temporal examina cómo varía la señal a lo largo del tiempo, mientras que el análisis en el dominio frecuencial examina cómo se distribuye la energía de la señal a través de diferentes frecuencias. La Transformada Rápida de Fourier (FFT) es una herramienta clave para convertir una señal del dominio temporal al dominio frecuencial, permitiendo la evaluación de componentes de frecuencia y su intensidad.

C. Ventanas en el Análisis Espectral

Para mejorar la precisión en el análisis espectral y minimizar los efectos de discontinuidades en la señal, se utilizan ventanas como la ventana de Hann. Estas ventanas suavizan los bordes de la señal y reducen el aliasing y las fugas espectrales en el cálculo de la FFT.

D. Aplicaciones del PDS

El PDS tiene aplicaciones en diversas áreas, como la ingeniería de audio, donde se utiliza para la mejora de la calidad del sonido, la reducción de ruido y la compresión de datos. También se aplica en el procesamiento de imágenes, la telemetría y las comunicaciones, donde ayuda a optimizar la transmisión y la recepción de señales.

En este proyecto, se implementaron técnicas de PDS para grabar, reproducir, y analizar señales de audio, proporcionando una comprensión práctica de estos conceptos fundamentales.

IV. RESULTADOS

A. Códigos realizados

1. Octave

```
if(exist('OCTAVE_VERSION', 'builtin')==0)
% estamos en octave
pkg load signal;
end

%menu principal
opcion = 0;
while opcion ~= 5
  % opcion = input('seleccione una opcion:\n 1. grabar audio\n 2. reproducir audio\n 3. graficar
  audio\n 4. salir\n ');
  %menu de opciones
disp('1. Grabar')
disp('2. Reproducir')
disp('3. Graficar')
disp('4. Graficar densidad')
disp('5. Salir')
opcion = input('Ingrese su eleccion: ');

switch opcion
  case 1
    %grabacion de audio
    try
      duracion = input('Ingrese la duracion de la grabacion en segundos: ');
      disp('Comenzando la grabacion...')
      recObj = audiorecorder; % 44100 Hz, 16 bits, 1 canal (mono)
      recordblocking(recObj, duracion);
      disp('Grabacion finalizada');
      data = getaudiodata(recObj);
      audiowrite('Alan ocatave.wav', data, recObj.SampleRate);
      disp('Archivo de audio grabado correctamente');
    catch
      disp('Errorr al grabar audio');
    end
  case 2
    %reproduccion de audio
    try
      disp('Reproduciendo...');
      [data, fs] = audioread('Alan ocatave.wav');
      sound(data, fs);
    catch
      disp('Error al reproducir el audio');
    end
  case 3
    %grafico de audio
    try
      [data, fs]=audioread('Alan_ocatave.wav');
      tiempo = linspace(0, length(data)/fs, length(data));
      plot(tiempo, data);
```

Figura 1: Código utilizado para realizar el procesamiento digital de señales en Octave

```
    plot(tiempo, data);
    xlabel('Tiempo (s)');
    ylabel('Amplitud');
    title('Audio');
  catch
    disp('Error al graficar el audio.');
  end
case 4
  %graficando espectro de frecuencia
  try
    disp('Graficando espectro de frecuencia...');
    [audio, Fs] = audioread('Alan ocatave.wav'); %lee la señal desde al archivo .wav
    N = length(audio); %numero de muestras de la señal
    f = linspace(0, Fs/2, N/2+1); %vector de frecuencias
    ventana = hann(N); %ventana de hann para reducir el efecto de las discontinuidades al calcular
    la FFT
    Sxx = pwelch(audio, ventana, 0, N, Fs); %densidad espectral de potencia
    plot(f, 10*log10(Sxx(1:N/2+1))); %grafica el espectro de frecuencia en dB
    xlabel('Frecuencia (Hz)');
    ylabel('Espectro de frecuencia de la señal grabada');
  catch
    disp('Error al graficar el audio');
  end
case 5
  % salir
  disp('Saliendo del programa...');
otherwise
  disp('Opcion no valida');
end
end
```

Figura 2: Código utilizado para realizar el procesamiento digital de señales en Octave

2. Python

grabador.py

```
import numpy as np
import matplotlib.pyplot as plt
import sounddevice as sd
import wavio
from scipy.io import wavfile
from scipy.signal import get_window, welch

# Función principal
def main():
    opcion = 0
    while opcion != 5:
        # Menú de opciones
        print('1. Grabar')
        print('2. Reproducir')
        print('3. Graficar')
        print('4. Graficar densidad')
        print('5. Salir')
        opcion = int(input('Ingrese su elección: '))

        if opcion == 1:
            try:
                # Grabación de audio
                duracion = int(input('Ingrese la duración de la grabación en segundos: '))
                print('Comenzando la grabación...')
                fs = 44100  # Frecuencia de muestreo
                data = sd.rec(int(duracion * fs), samplerate=fs, channels=1, dtype='float64')
                sd.wait()
                print('Grabación finalizada')
                wavio.write('Alan_python.wav', data, fs, sampwidth=2)
                print('Archivo de audio grabado correctamente')
            except Exception as e:
                print(f'Error al grabar audio: {e}')
```

Figura 3: Código utilizado para realizar el procesamiento digital de señales en python

```
        elif opcion == 2:
            try:
                # Reproducción de audio
                print('Reproduciendo...')
                fs, data = wavfile.read('Alan_python.wav')
                sd.play(data, fs)
                sd.wait()

            except Exception as e:
                print(f'Error al reproducir el audio: {e}')

        elif opcion == 3:
            try:
                # Gráfico de audio
                fs, data = wavfile.read('Alan_python.wav')
```

localhost:53091/56b00627-1ff1-4d42-92d6-c2033e34a93b/

17/8/24, 10:02 a.m. grabador.py

```
                tiempo = np.linspace(0, len(data) / fs, len(data))
                plt.plot(tiempo, data)
                plt.xlabel('Tiempo (s)')
                plt.ylabel('Amplitud')
                plt.title('Audio')
                plt.show()
            except Exception as e:
                print(f'Error al graficar el audio: {e}')

        elif opcion == 4:
            try:
                # Graficando espectro de frecuencia
                print('Graficando espectro de frecuencia...')
                fs, audio = wavfile.read('Alan_python.wav')
                N = len(audio)
```

Figura 4: Código utilizado para realizar el procesamiento digital de señales en python

```
                f = np.linspace(0, fs/2, N//2+1)
                ventana = get_window('hann', N)
                f, Sxx = welch(audio, fs, window=ventana, nperseg=N)
                plt.plot(f, 10 * np.log10(Sxx))
                plt.xlabel('Frecuencia (Hz)')
                plt.ylabel('Espectro de frecuencia de la señal grabada (dB)')
                plt.show()
            except Exception as e:
                print(f'Error al graficar la densidad espectral: {e}')

        elif opcion == 5:
            print('Saliendo del programa...')
        else:
            print('Opción no válida')

if __name__ == "__main__":
    main()
```

Figura 5: Código utilizado para realizar el procesamiento digital de señales en python

B. Explicación de los códigos

1. *Código en Octave*

Configuración y Menú Principal: El código empieza verificando si está en Octave e importa el paquete signal necesario para el análisis de señales y presenta un menú al usuario con opciones para grabar, reproducir, graficar y analizar la señal de audio.

Grabar Audio (Opción 1): Se solicita la duración de la grabación y se inicia la grabación con audiorecorder, la grabación se guarda en un archivo WAV utilizando audiowrite.

Reproducir Audio (Opción 2): Se lee el archivo WAV usando audioread y se reproduce el audio con la función sound.

Graficar Audio (Opción 3): Se lee el archivo WAV y se grafica la señal de audio en función del tiempo.

Graficar Densidad Espectral de Potencia (Opción 4): Se calcula la densidad espectral de potencia utilizando la función pwelch y se grafica el espectro de frecuencia.

Salir (Opción 5): Finaliza el programa.

2. *Código en Python*

Configuración y Menú Principal:

El código importa bibliotecas necesarias (numpy, matplotlib, sounddevice, wavio, scipy) y presenta un menú al usuario con opciones similares al de Octave.

Grabar Audio (Opción 1):

Solicita la duración de la grabación y usa sounddevice.rec para capturar la señal de audio. Guarda el audio en un archivo WAV utilizando wavio.write.

Reproducir Audio (Opción 2):

Lee el archivo WAV con scipy.io.wavfile.read y reproduce el audio usando sounddevice.play.

Graficar Audio (Opción 3):

Lee el archivo WAV y grafica la señal en función del tiempo usando matplotlib.

Graficar Densidad Espectral de Potencia (Opción 4):

Lee el archivo WAV, calcula la densidad espectral de potencia utilizando scipy.signal.welch y grafica el espectro de frecuencia.

Salir (Opción 5):

Finaliza el programa.

C. Programa ejecutado

1. *Octave*

```
>> tarea

1. Grabar
2. Reproducir
3. Graficar
4. Graficar densidad
5. Salir
Ingrese su eleccion:
```

Figura 6: Menú desplegado en octave

```
1. Grabar
2. Reproducir
3. Graficar
4. Graficar densidad
5. Salir
Ingrese su eleccion: 1
Ingrese la duracion de la grabacion en segundos: 10
Comenzando la grabacion...
Grabacion finalizada
```

Figura 7: Ejecución de la opcion 1 en octave

Figura 8: Archivo .wav creado

```
1. Grabar
2. Reproducir
3. Graficar
4. Graficar densidad
5. Salir
Ingrese su eleccion: 2
Reproduciendo...
```

Figura 9: Ejecución de la opcion 2 en octave

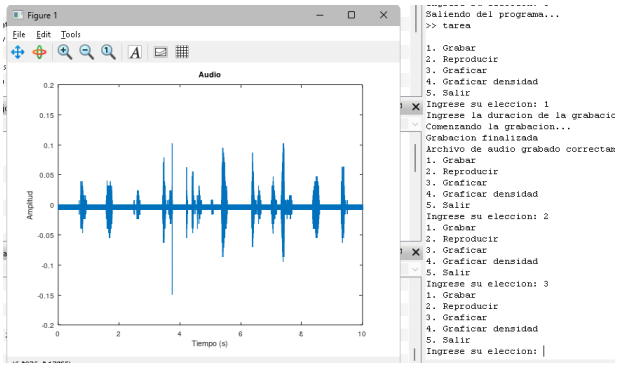

Figura 10: Opcion 3 ejecutado en octave

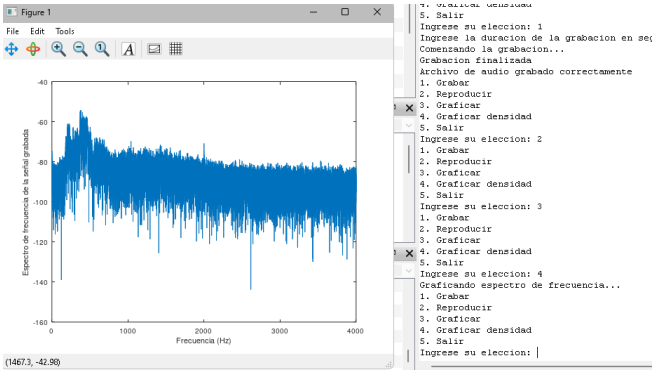

Figura 11: Opcion 4 ejecutado en octave

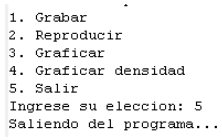

Figura 12: Opcion 5 ejecutado en octave

2. *Python*

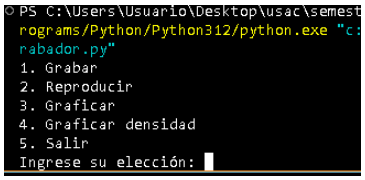

Figura 13: Menú desplegado en python

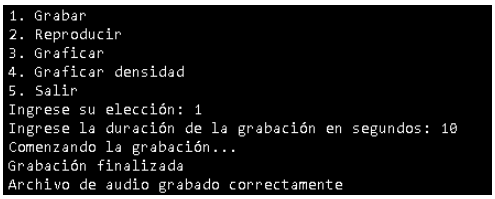

Figura 14: Ejecución de la opcion 1 en python

Figura 15: Archivo .wav creado

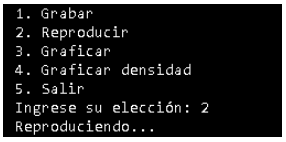

Figura 16: Ejecución de la opcion 2 en python

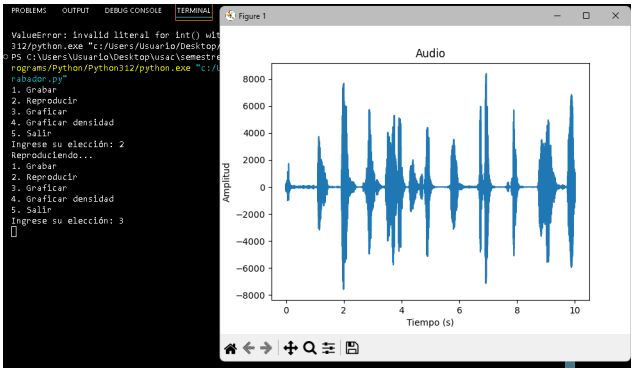

Figura 17: Opcion 3 ejecutado en python

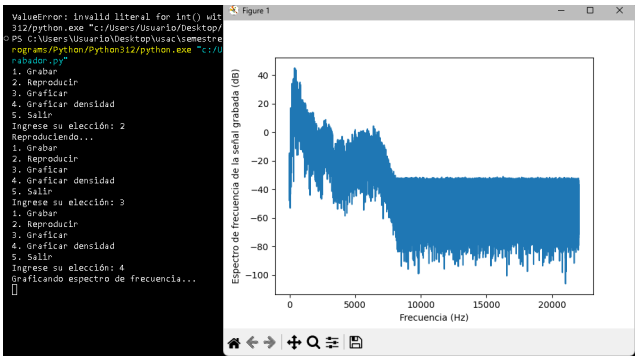

Figura 18: Opcion 4 ejecutado en python

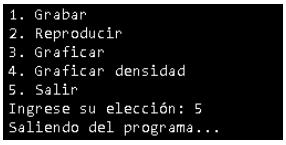

Figura 19: Opcion 5 ejecutado en python

D. Repositorio privado

Repositorio creado en Github

V. CONCLUSIONES

* Se logró capturar señales de audio en tiempo real y almacenarlas en formato digital, estableciendo una base sólida para cualquier aplicación de PDS.

* La visualización de las señales en el dominio temporal permitió observar su comportamiento a lo largo del tiempo, mientras que el análisis espectral mediante la densidad espectral de potencia facilitó el estudio de la distribución de energía a través de las frecuencias.

* La implementación de ventanas de Hann en el cálculo de la densidad espectral de potencia demostró su utilidad para reducir errores y mejorar la precisión en el análisis espectral.

* La conversión del código de Octave a Python destacó no solo la mayor versatilidad y la disponibilidad de bibliotecas avanzadas en Python, sino también la superioridad en términos de velocidad de ejecución, lo que facilita un procesamiento más eficiente de las señales.

* El diseño de un menú interactivo hizo que el programa fuera accesible y fácil de usar, cumpliendo con el objetivo de ofrecer una herramienta práctica para la manipulación y análisis de señales de audio.

---

[1] https://docs.python.org/3/
[2] https://ieeexplore.ieee.org/document/1161901
[3] https://python-sounddevice.readthedocs.io/en/0.4.7/