Tecnológico de Monterrey

TE3003B.581

Integración de robótica y sistemas inteligentes (Gpo 581)

Tarea 1. HMI for signal processing

Ana Lucía Ahedo Reyes A01661890 A01661890@tec.mx

Fecha de entrega: 10 de abril del 2025

Profesor: Jesús Manuel Vázquez Nicolás

Manual de Usuario

Este procesador de audio es una aplicación desarrollada en Python que usa la biblioteca PyQt5, la cual permite realizar operaciones básicas de procesamiento digital de señales de audio. Entre sus funciones principales se encuentran la carga y visualización de archivos de audio, la aplicación de filtros (pasa-bajas, pasa-altas y pasa-banda), la generación del análisis en frecuencia mediante la Transformada de Fourier (FFT) y la exportación del audio procesado a un nuevo archivo.

Requisitos del Sistema

Para utilizar la aplicación, es necesario tener instalada una versión de Python 3.7 o superior, además de las siguientes bibliotecas:

- numpy – para operaciones numéricas.
- scipy – para el diseño y aplicación de filtros.
- librosa – para la carga de archivos de audio.
- soundfile – para guardar archivos WAV.
- matplotlib – para la visualización gráfica.
- PyQt5 – para construir la interfaz gráfica.

En caso de no contar con estas dependencias, las puedes instalar desde la terminal ejecutando:

```Shell
pip install numpy scipy librosa soundfile matplotlib PyQt5
```

Ejecución de la Aplicación

Para comenzar a utilizar la herramienta, se debe de guardar el código en un archivo, en este caso llamado “HMI.py”. Luego, abrimos una terminal, y dentro de la carpeta donde se encuentra el archivo y ejecuta el siguiente comando:

```Python
python HMI.py
```

Esto abrirá una ventana con la interfaz principal de la aplicación.

Descripción de la Interfaz

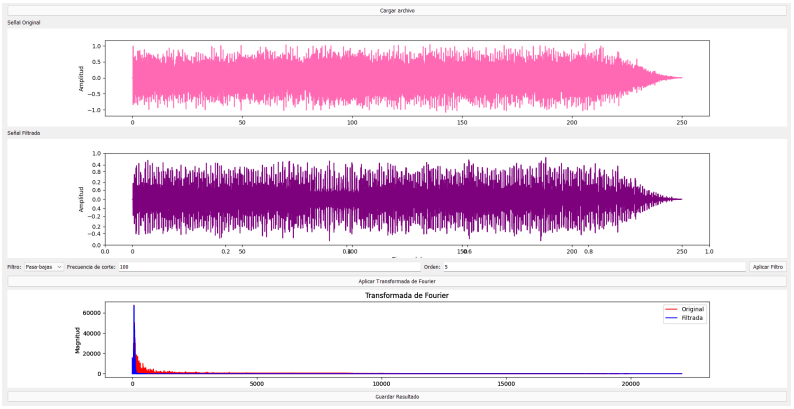
En la parte superior de la interfaz se encuentra el botón para cargar un archivo de audio. A continuación, se muestran dos gráficos: uno para la señal original y otro para la señal filtrada. Debajo de ellos se encuentran los controles para aplicar filtros, especificar los parámetros y activar el procesamiento. Finalmente, se ubican el botón para aplicar la Transformada de Fourier y el botón para guardar el audio resultante.

Carga de un Archivo de Audio

Al hacer clic en el botón "Cargar archivo", se abrirá una ventana para seleccionar un archivo de audio. La aplicación soporta formatos comunes como “.wav”, “.mp3” y “.aac”. Una vez cargado, se mostrará la forma de onda del audio original en un gráfico de color rosa, que representa la amplitud de la señal a lo largo del tiempo. En este momento, aún no se ha aplicado ningún procesamiento sobre la señal.

Aplicación de Filtros

La sección de filtrado permite modificar el contenido frecuencial del audio cargado. Podés elegir entre tres tipos de filtros:

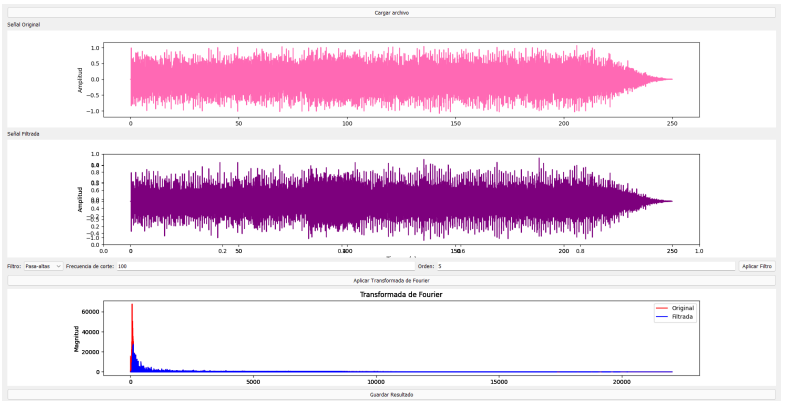


1. Pasa-bajas: atenúa las frecuencias por encima de la frecuencia de corte.
2. Pasa-altas: atenúa las frecuencias por debajo de la frecuencia de corte.
3. Pasa-banda: permite el paso de un rango de frecuencias y atenúa el resto.

Para aplicar un filtro, primero seleccioná el tipo de filtro desde el menú desplegable. Luego, ingresá la frecuencia de corte en Hz en el campo correspondiente (por ejemplo, 1000) y el orden del filtro (por ejemplo, 5), que define la inclinación de la curva de atenuación (un orden más alto significa una caída más abrupta). Finalmente, hacé clic en "Aplicar Filtro". El resultado se mostrará en el segundo gráfico, donde la señal filtrada aparecerá en color morado.

Análisis de Frecuencia (Transformada de Fourier)

Una de las funciones más útiles de esta aplicación es la posibilidad de analizar el contenido en frecuencia del audio utilizando la Transformada Rápida de Fourier (FFT). Al hacer clic en el botón "Aplicar Transformada de Fourier", se genera un gráfico que muestra el espectro de frecuencias. La curva roja representa la FFT del audio original, mientras que la curva azul representa la FFT del audio filtrado. Este análisis es muy útil para visualizar cómo el filtro afecta el contenido armónico o ruidoso del archivo.



Guardado del Archivo Procesado

Una vez aplicado el filtro, es posible guardar el audio resultante. Para ello, debes de hacer clic en "Guardar Resultado". Se abrirá una ventana de diálogo donde puedes elegir el nombre y la ubicación del nuevo archivo. El audio se guarda en formato “.wav” con la misma frecuencia de muestreo del archivo original.

Colores en los Gráficos

Para una mejor interpretación visual, la aplicación utiliza una codificación de colores:

- Rosa (hotpink): señal original (forma de onda).
- Morado (purple): señal filtrada (forma de onda).
- Rojo (red): FFT del audio original.
- Azul (blue): FFT del audio filtrado.

Posibles Problemas y Soluciones

a. No se muestra la señal filtrada: Verifica que hayas cargado un archivo y hayas hecho clic en "Aplicar Filtro".
b. El filtro no tiene efecto visible: Prueba con diferentes valores de frecuencia de corte o un orden mayor.

c. El botón de guardar no hace nada: Asegurate de haber aplicado un filtro antes de intentar guardar.
d. La aplicación no se ejecuta o lanza errores: Confirma si tienes instaladas todas las librerías necesarias.

Explicación del Código

En la siguiente sección se explicará la funcionalidad del código implementado para la interfaz gráfica y el análisis de audio:
Lo primero que se hace es importar todas las librerías que se usarán las cuales son: “numpy”, “scipy”, “librosa” y “soundfile”, todas estas sirven para la parte de procesamiento del audio. Posteriormente, se utilizó “PyQt5.QtWidgets” para la creación de la interfaz gráfica. Finalmente, se importó “matplotlib” para graficar las señales dentro de la aplicación.

```python
import sys
import numpy as np
import librosa
import soundfile as sf
from scipy.signal import butter, lfilter
from scipy.fft import fft, fftfreq
from PyQt5.QtWidgets import (QApplication, QWidget, QVBoxLayout,
QPushButton, QLabel,QFileDialog, QComboBox, QHBoxLayout, QLineEdit)
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure
```

Se define una clase que hereda de “QWidge”t, esto nos permitirá realizar la interfaz gráfica. Dentro del constructor *__init__()* se inicializan las variables principales y se llama a *init_ui()* para construir la interfaz de usuario.

```python
class AudioProcessor(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Procesador de Audio")
        self.setGeometry(100, 100, 1000, 600)

        self.audio = None
        self.sr = None
```

```
        self.filtered_audio = None
        self.init_ui()
```

En el siguiente método *init_ui()* se define la interfaz gráfica. Se organiza la ventana usando un layout vertical, añadiendo botones para cargar archivos, aplicar filtros, realizar la transformada de Fourier y guardar resultados. También se incluyen las gráficas para visualizar la señal original (en rosa), la filtrada (en morado) y su espectro de frecuencias. Además, se incorporan controles para elegir el tipo de filtro, frecuencia de corte y orden.

```python
def init_ui(self):
        layout = QVBoxLayout()

        # Botón Cargar archivo
        self.load_btn = QPushButton("Cargar archivo")
        self.load_btn.clicked.connect(self.load_audio)
        layout.addWidget(self.load_btn)

        # Visualización
        self.plot_original = FigureCanvas(Figure(figsize=(5, 2.5)))
        self.plot_filtered = FigureCanvas(Figure(figsize=(5.5, 3)))
        layout.addWidget(QLabel("Señal Original"))
        layout.addWidget(self.plot_original)
        layout.addWidget(QLabel("Señal Filtrada"))
        layout.addWidget(self.plot_filtered)

        # Controles de Filtro
        filter_layout = QHBoxLayout()
        self.filter_combo = QComboBox()
        self.filter_combo.addItems(["Pasa-bajas", "Pasa-altas",
"Pasa-banda"])
        self.cutoff_input = QLineEdit("1000")  # Frecuencia de corte
        self.order_input = QLineEdit("5")      # Orden del filtro
        self.apply_filter_btn = QPushButton("Aplicar Filtro")
        self.apply_filter_btn.clicked.connect(self.apply_filter)

        filter_layout.addWidget(QLabel("Filtro:"))
        filter_layout.addWidget(self.filter_combo)
        filter_layout.addWidget(QLabel("Frecuencia de corte:"))
        filter_layout.addWidget(self.cutoff_input)
        filter_layout.addWidget(QLabel("Orden:"))
        filter_layout.addWidget(self.order_input)
        filter_layout.addWidget(self.apply_filter_btn)
        layout.addLayout(filter_layout)
```

```
        # Transformada
        self.fft_btn = QPushButton("Aplicar Transformada de Fourier")
        self.fft_btn.clicked.connect(self.plot_fft)
        layout.addWidget(self.fft_btn)

        self.fft_canvas = FigureCanvas(Figure(figsize=(5, 2.5)))
        layout.addWidget(self.fft_canvas)

        # Guardar resultado
        self.save_btn = QPushButton("Guardar Resultado")
        self.save_btn.clicked.connect(self.save_audio)
        layout.addWidget(self.save_btn)

        self.setLayout(layout)
```

En la siguiente función se le permite al usuario cargar un archivo de audio desde su computadora utilizando un cuadro de diálogo. Se usa “librosa” para leer el archivo y obtener tanto la señal como la frecuencia de muestreo. Una vez cargado, se grafica la señal original en color rosa usando *plot_signal*, y se limpia el gráfico de la señal filtrada.

```
Python
def load_audio(self):
        filename, _ = QFileDialog.getOpenFileName(
            self, "Cargar archivo", "", "Archivos de audio (*.wav *.mp3
*.aac)"
        )
        if filename:
            self.audio, self.sr = librosa.load(filename, sr=None)
            self.plot_signal(self.plot_original.figure, self.audio, self.sr,
color='hotpink')  # Rosa
            self.filtered_audio = None
            self.plot_signal(self.plot_filtered.figure, [], self.sr,
color='purple')  # Morado
```

Posteriormente, la función *butte_filter* implementa un filtro digital Butterworth. Dependiendo del tipo (pasa-bajas, pasa-altas, pasa-banda), calcula los coeficientes del filtro con “scipy.signal.butter” y lo aplica a la señal con “lfilter”. En esta, también se convierten las frecuencias de corte al formato normalizado (dividiendo por Nyquist) y se devuelve la señal filtrada.

```python
def butter_filter(self, data, cutoff, fs, order=5, ftype='low'):
        nyq = 0.5 * fs
        if ftype == 'band':
            low, high = cutoff
            low /= nyq
            high /= nyq
            b, a = butter(order, [low, high], btype='band')
        else:
            cutoff /= nyq
            b, a = butter(order, cutoff, btype=ftype)
        return lfilter(b, a, data)
```

Después, en el siguiente bloque de código, se toma los parámetros ingresados por el usuario (tipo de filtro, frecuencia de corte y orden), y llama a *butter_filter()* para procesar la señal original. Luego, se grafica la señal filtrada en color morado y se manejan errores básicos en caso de entradas inválidas.

```python
def apply_filter(self):
        if self.audio is None:
            return
        try:
            ftype = self.filter_combo.currentText()
            cutoff = float(self.cutoff_input.text())
            order = int(self.order_input.text())

            if ftype == "Pasa-bajas":
                self.filtered_audio = self.butter_filter(self.audio, cutoff,
self.sr, order, ftype='low')
            elif ftype == "Pasa-altas":
                self.filtered_audio = self.butter_filter(self.audio, cutoff,
self.sr, order, ftype='high')
            elif ftype == "Pasa-banda":
                self.filtered_audio = self.butter_filter(self.audio,
[cutoff, cutoff * 1.5], self.sr, order, ftype='band')

            self.plot_signal(self.plot_filtered.figure, self.filtered_audio,
self.sr, color='purple')  # Morado
        except Exception as e:
            print(f"Error aplicando filtro: {e}")
```

Luego, se aplica la Transformada Rápida de Fourier en la función *plot_fft()*, en esta se calcula y grafica la Transformada de Fourier de la señal original (en rojo) y la señal filtrada (en azul),

si existe. Después, se muestra un gráfico con el eje X como frecuencia en Hz y el eje Y como magnitud de la señal.

```python
def plot_fft(self):
        if self.audio is None:
            return

        ax = self.fft_canvas.figure.subplots()
        ax.clear()

        # FFT de la señal original (rojo)
        N_orig = len(self.audio)
        yf_orig = np.abs(fft(self.audio))[:N_orig // 2]
        xf_orig = fftfreq(N_orig, 1 / self.sr)[:N_orig // 2]
        ax.plot(xf_orig, yf_orig, color='red', label='Original')

        # FFT de la señal filtrada (azul)
        if self.filtered_audio is not None:
            N_filt = len(self.filtered_audio)
            yf_filt = np.abs(fft(self.filtered_audio))[:N_filt // 2]
            xf_filt = fftfreq(N_filt, 1 / self.sr)[:N_filt // 2]
            ax.plot(xf_filt, yf_filt, color='blue', label='Filtrada')

        ax.set_title("Transformada de Fourier")
        ax.set_xlabel("Frecuencia (Hz)")
        ax.set_ylabel("Magnitud")
        ax.legend()
        self.fft_canvas.draw()
```

Casi para finalizar, el método *plot_signal()* se usa para graficar una señal de audio en función del tiempo. Se utiliza “matplotlib” para graficar, y permite personalizar el color (rosa para original, morado para filtrada).

```python
def plot_signal(self, fig, data, sr, color='blue'):
        ax = fig.subplots()
        ax.clear()
        if len(data) > 0:
            t = np.linspace(0, len(data) / sr, len(data))
            ax.plot(t, data, color=color)
            ax.set_xlabel("Tiempo (s)")
            ax.set_ylabel("Amplitud")
        fig.canvas.draw()
```

Finalmente, la función *save_audio()* permite guardar la señal filtrada como un archivo ".wav" usando soundfile. Solo se activa si existe una señal filtrada.

```python
def save_audio(self):
        if self.filtered_audio is None:
            return
        filename, _ = QFileDialog.getSaveFileName(
            self, "Guardar archivo", "", "Archivo WAV (*.wav)"
        )
        if filename:
            sf.write(filename, self.filtered_audio, self.sr)
```