Procesamiento de Imágenes, Audio y Vídeo

Práctica 4:
Procesamiento de audio

Manuel Alejandro Torrealba Torrealba, Alejandro Alemán Alemán

Ejercicio 1: Construir un identificador de notas musicales. Es decir; en su versión más sencilla (y suficiente) la entrada es un sonido con una sola nota musical y debe identificar cuál es. Por simplicidad elija un único instrumento para la identificación.

(Aportes Adicionales)
a) El caso más sencillo es el del piano, pero se valorará que se haga con otros instrumentos como la guitarra, la trompeta…
b) Se valorará que se identifiquen octavas de notas
d) Aportes adicionales: Uso de otro método

1. Tabla de nombres de notas musicales

Para poder traducir una frecuencia detectada en el audio a su nota musical correspondiente, se define una tabla con los nombres de las **12 notas del sistema temperado occidental**, expresadas en notación española:

```
NOTE_NAMES = ["Do", "Do#", "Re", "Re#", "Mi", "Fa",
              "Fa#", "Sol", "Sol#", "La", "La#", "Si"]
```

En la música occidental, una octava se divide en **12 semitonos**, que se repiten de forma cíclica: Do → Do# → Re → … → Si → (y vuelve a Do).
Cuando se conozca el número MIDI de una nota, bastará aplicar un módulo 12 para determinar a cuál de estos nombres pertenece.

2. Detección de la frecuencia dominante mediante FFT

El objetivo principal es obtener la **frecuencia fundamental** de un fragmento de audio que contiene una nota musical. Para ello se utiliza la **Transformada Rápida de Fourier (FFT)**, que permite analizar el contenido frecuencial de la señal.

a) Preprocesamiento del audio

Antes de aplicar la FFT, se realizan varias operaciones que mejoran la precisión del análisis:

Conversión a mono:

Si el archivo de audio posee dos canales (estéreo), se promedian ambos para obtener un solo canal:

```
if audio_data.ndim == 2:
    audio_data = audio_data.mean(axis=1)
```

Conversión a coma flotante y eliminación de la componente DC:

Se resta la media de la señal para eliminar desplazamientos verticales que podrían interferir en el espectro.

```
audio = audio_data.astype(np.float32)
audio = audio - np.mean(audio)
```

Aplicación de una ventana de Hamming:
Para reducir las fugas espectrales (*spectral leakage*) al calcular la FFT, se utiliza una ventana de Hamming, que suaviza los extremos de la señal.

```
window = np.hamming(len(audio))
audio_win = audio * window
```

b) Cálculo de la FFT y búsqueda de la frecuencia dominante

Realización de la FFT real:
Se emplea `rfft`, que devuelve únicamente la mitad positiva del espectro (suficiente para señales reales).

```
spectrum = np.fft.rfft(audio_win)
freqs = np.fft.rfftfreq(N, d=1.0 / sample_rate)
magnitude = np.abs(spectrum)
```

Restricción del rango de frecuencias válidas:
Para evitar detectar ruido en frecuencias muy bajas o armónicos superiores, se delimita el rango de búsqueda:

```
valid = (freqs >= min_freq) & (freqs <= max_freq)
```

Detección del pico máximo:
La frecuencia dominante se identifica como el índice de la mayor magnitud dentro del rango válido:

```
idx_peak = np.argmax(mag_valid)
freq_dominante = freqs_valid[idx_peak]
```

Esta frecuencia es, en la mayoría de los casos, la **frecuencia fundamental** de la nota musical.

3. Conversión de frecuencia a nota musical y octava

Una vez calculada la frecuencia fundamental, es necesario traducirla a su **representación musical**: nombre de nota y octava.

La música moderna utiliza la afinación estándar **La4 = 440 Hz**, asociada al número **MIDI 69**.
A partir de esta referencia, cualquier frecuencia puede convertirse a número MIDI mediante la fórmula:

$$\text{MIDI} = 69 + 12 \log_2 \left(\frac{f}{440} \right)$$

En el código:

```
midi = int(round(69 + 12 * np.log2(f / 440.0)))
```

Una vez obtenido el número MIDI:
El **nombre de la nota** se calcula como:

```
nombre_nota = NOTE_NAMES[midi % 12]
```

La **octava** se obtiene mediante:

```
octava = midi // 12 - 1
```

Esto sigue la definición estándar, donde:

- Do4 está en la octava 4
- La4 = 440 Hz está en la octava 4
- La3 = 220 Hz, La2 = 110 Hz, etc.

Por qué el método funciona para cualquier instrumento

Este método es válido para cualquier instrumento porque todos, independientemente de su timbre o forma de onda, producen una nota que tiene siempre una **frecuencia fundamental** claramente definida. Aunque cada instrumento pueda sonar diferente debido a sus armónicos y características propias, la frecuencia fundamental —que es la que determina la altura de la nota— es la misma para todos cuando tocan la misma nota. Como el algoritmo detecta precisamente esa frecuencia fundamental mediante la FFT, la identificación funciona igual para un piano, una guitarra, una trompeta o la voz humana. Por tanto, el sistema no depende del instrumento, sino únicamente de la frecuencia, lo que garantiza su generalidad.

Aporte adicional: detección de frecuencia mediante autocorrelación

Además del método basado en la Transformada de Fourier, se ha implementado un segundo procedimiento para la estimación de la frecuencia fundamental utilizando **autocorrelación**. Este método es ampliamente empleado en análisis de audio porque se basa en la naturaleza periódica de las señales musicales y, en muchos casos, resulta más robusto frente a armónicos dominantes o ligeras variaciones tímbricas.

Fundamento teórico

La idea principal consiste en calcular la **autocorrelación** de la señal consigo misma. Una onda periódica se repite cada cierto intervalo llamado *periodo*. Por tanto, al desplazar la señal y compararla consigo misma, la autocorrelación presentará máximos cuando el desplazamiento coincida con un múltiplo entero del periodo. El primer máximo significativo (excluyendo el desplazamiento cero) permite estimar directamente la duración del periodo y, por tanto, la frecuencia fundamental:

$$f = \frac{1}{T} = \frac{\text{sample\_rate}}{\text{lag}}$$

Este método tiene la ventaja de que **no depende de la amplitud de los armónicos**, ya que la periodicidad de la forma de onda permite identificar la frecuencia fundamental incluso cuando los armónicos superiores tienen mayor energía que el fundamental, algo que puede ocurrir en instrumentos como la trompeta, el clarinete o la guitarra.

Ventajas del método

- **Mayor robustez frente a armónicos.**
  A diferencia de la FFT, que puede confundir un armónico dominante con el fundamental, la autocorrelación detecta directamente la periodicidad global de la señal, identificando la frecuencia correcta incluso cuando los primeros armónicos poseen mayor energía.
- **Menor sensibilidad al ruido armónico.**
  Si bien un ruido fuerte puede afectar al cálculo, la autocorrelación tolera bastante bien pequeñas perturbaciones.
- **Independencia del instrumento.**
  De nuevo, el método se basa en la periodicidad de la nota, por lo que funciona con guitarra, trompeta, voz, violín o cualquier instrumento monofónico.

Limitaciones

- Es sensible al ruido impulsivo o a señales poco periódicas.
- No funciona bien si la nota contiene vibrato intenso o glissando, ya que la periodicidad deja de ser estable.
- Es más costoso computacionalmente que la FFT.

Este método complementa al detector basado en FFT y permite comparar dos aproximaciones clásicas al problema. Aunque ninguna de las dos técnicas es perfecta en grabaciones reales de instrumentos (especialmente con vibrato o técnicas extendidas), la autocorrelación ofrece una alternativa sólida y educativa para mostrar cómo se puede detectar el periodo fundamental en señales musicales.

c) Identificación de acordes

Además de identificar una única nota musical, se ha implementado un sistema experimental capaz de detectar acordes (es decir, conjuntos de notas sonando simultáneamente). A diferencia del caso monofónico, donde la frecuencia fundamental

es dominante y fácilmente localizable, los acordes presentan múltiples frecuencias relevantes, correspondientes a cada una de las notas que los forman. Por ello, su análisis requiere una estrategia distinta basada en la identificación de varios picos espectrales.

Fundamento del método

Cuando se reproduce un acorde (por ejemplo, Do–Mi–Sol), cada nota genera una **frecuencia fundamental** (Do≈261 Hz, Mi≈329 Hz, Sol≈392 Hz),
más una serie de **armónicos** en múltiplos enteros de dichas frecuencias.

En la Transformada de Fourier de un acorde no aparece un único pico, sino **varios picos prominentes**, uno por cada nota fundamental (y otros muchos debido a sus armónicos). Por tanto, la estrategia consiste en:

1. **Calcular la FFT** del audio.
2. **Localizar los picos de mayor energía** en el espectro.
3. Escoger los picos más relevantes (los n primeros que superen un umbral).
4. **Asociar cada pico a la nota más cercana** en una tabla de notas que incluye todas las octavas.
5. Agrupar y devolver el conjunto de notas detectadas, lo cual corresponde al acorde.

Este procedimiento permite identificar acordes sencillos (tríadas) con bastante fiabilidad siempre que el audio sea relativamente limpio y las notas estén bien separadas en frecuencia.

Es importante destacar que la identificación de acordes es un problema considerablemente más complejo que la detección de notas individuales:

- Los armónicos de cada nota pueden confundirse con otras notas del acorde.
- En acordes densos o con instrumentos muy ricos en armónicos (piano, guitarra, violín), los picos no están siempre claramente separados.
- La amplitud de cada nota no siempre es la misma, por lo que su pico puede no ser de los más fuertes.
  La afinación, la reverberación y el ruido pueden afectar al resultado.

Sin embargo, incluso con estas limitaciones, el sistema implementado permite detectar adecuadamente acordes simples, lo cual constituye un aporte notable y claramente superior a lo requerido en la práctica.

Ejercicio 2: Aplicación interactiva de filtrado de audio en dominio frecuencial

En este ejercicio se desarrolla una aplicación interactiva que permite aplicar distintos tipos de filtros a señales de audio. La herramienta incorpora controles para modificar los umbrales de filtrado, representación gráfica en el dominio temporal y frecuencial, reproducción de la señal original y filtrada, y la posibilidad de grabar audio en tiempo real para procesarlo de forma inmediata.

El objetivo es comprender el funcionamiento de los filtros digitales basados en la Transformada Rápida de Fourier (FFT), así como observar de manera intuitiva cómo afectan a una señal ruidosa.

1. Descripción general de la aplicación

La aplicación se implementa en Python utilizando las siguientes bibliotecas:

- **Tkinter** para la interfaz gráfica de usuario.
- **NumPy** para el procesado numérico.
- **SciPy (wavfile)** para cargar archivos de audio WAV.
- **sounddevice** para grabar y reproducir audio.
- **Matplotlib** para mostrar las gráficas embebidas dentro de la propia aplicación.

El usuario dispone de dos modalidades de entrada:

1. **Cargar un archivo WAV desde el equipo.**
2. **Grabar audio directamente mediante micrófono**, que se normaliza y se incorpora automáticamente como señal de entrada.

2. Implementación del filtrado mediante FFT

El filtrado se implementa en el dominio frecuencial, aplicando la Transformada Rápida de Fourier a la señal:

$$X(f) = \mathcal{F}\{x(t)\}$$

El filtrado se realiza multiplicando el espectro por una máscara que anula o conserva determinadas bandas de frecuencia según el filtro seleccionado.

2.1 Tipos de filtros implementados

La aplicación incorpora cuatro tipos de filtros:

• Filtro pasa-bajo

Permite frecuencias inferiores a un umbral f1 y elimina las superiores:

$$H(f) = \begin{cases} 1 & f \leq f_1 \\ 0 & f > f_1 \end{cases}$$

• Filtro pasa-alto

Deja pasar frecuencias por encima del umbral:

$$H(f) = \begin{cases} 0 & f < f_1 \\ 1 & f \geq f_1 \end{cases}$$

• Filtro pasa-banda

Conserva una franja entre dos umbrales f1 y f2:

$$H(f) = \begin{cases} 1 & f_1 \leq f \leq f_2 \\ 0 & \text{en otro caso} \end{cases}$$

• Filtro rechaza-banda (notch)

Elimina una franja central:

$$H(f) = \begin{cases} 0 & f_1 \leq f \leq f_2 \\ 1 & \text{en otro caso} \end{cases}$$

Cada filtro se implementa generando una máscara booleana sobre el espectro, la cual se multiplica directamente por el resultado de la FFT.
La señal filtrada se obtiene mediante la transformada inversa:

$$x_{\text{filtrado}}(t) = \mathcal{F}^{-1}\{X(f) \cdot H(f)\}$$

3. Controles interactivos

La interfaz incorpora varios elementos que permiten un control total sobre el proceso:

Selector de tipo de filtro

Un menú desplegable permite elegir entre los cuatro tipos.
Automáticamente se habilitan los selectores correspondientes:

- **Pasa-bajo / pasa-alto:** solo requiere un umbral.
- **Pasa-banda / rechaza-banda:** requieren dos umbrales.

Sliders de frecuencia

Los umbrales aparecen como deslizadores que abarcan desde 0 hasta la frecuencia de Nyquist del audio cargado o grabado.

Botones de filtrado

Permiten ejecutar el filtrado y visualizar el resultado al instante.

4. Visualización de los resultados

Una vez aplicado un filtro, la aplicación muestra dos gráficas embebidas:

1. Dominio temporal

Compara la señal original y la filtrada:

- Cambios en amplitud
- Desaparición de ruido
- Aparición de oscilaciones más suaves (pasa-bajo) o más rápidas (pasa-alto)

2. Dominio frecuencial

Se representan los módulos del espectro:

- Se observa exactamente qué componentes se atenúan o eliminan.
- El filtro queda visualmente demostrado.

El uso de Matplotlib dentro de Tkinter se resuelve mediante `FigureCanvasTkAgg`, permitiendo gráficos actualizables directamente en la aplicación.

5. Reproducción de audio (Aporte adicional A)

La aplicación permite:

- Reproducir **la señal original**.
- Reproducir **la señal filtrada**, con los parámetros seleccionados.

Esto permite comparar resultados auditivamente, representando una mejora significativa en la comprensión perceptiva del filtrado.

6. Grabación en tiempo real (Aporte adicional B)

Como aporte extra, se incorporó la opción de grabación mediante micrófono:

- El usuario puede grabar voz, ruidos o cualquier sonido.
- Tras finalizar la grabación, la señal se normaliza y queda cargada en la aplicación.
- Cualquier filtro puede aplicarse sobre este audio recién grabado.
- Se diseñó una solución especial para macOS para evitar errores de audio con `InputStream`.

Esta funcionalidad convierte la aplicación en una herramienta interactiva que permite experimentar con filtrado en tiempo real.

7. Observaciones sobre señales ruidosas

Durante las pruebas se utilizaron:

- audios con ruido blanco,
- notas musicales contaminadas con ruido,
- grabaciones propias hechas con micrófono.

Los resultados observados fueron:

- **El filtro pasa-bajo** elimina eficazmente el ruido agudo.
- **El filtro pasa-alto** elimina zumbidos graves o ruidos de fondo.
- **El filtro pasa-banda** permite aislar frecuencias concretas (por ejemplo, formantes de voz).
- **El filtro rechaza-banda** elimina tonos molestos como zumbidos a 50/60 Hz.

Las gráficas permiten demostrar estos comportamientos de manera clara e intuitiva.

8. Conclusiones

La aplicación desarrollada cumple con todos los requisitos del ejercicio:

- Permite aplicar filtrado FFT de manera interactiva.
- Contiene un selector para los cuatro tipos de filtro.
- Permite manipular los umbrales mediante sliders.
- Muestra la señal original y filtrada en dominio temporal y frecuencial.
- Permite reproducir el audio antes y después del filtrado.
- Incluye grabación de audio en tiempo real como aporte adicional.