Física Experimental II

Analizador de Espectro de Audio en Tiempo Real con Python

Roqueta, Matías Daniel.

Centro Atómico Bariloche y Instituto Balseiro, Comisión Nacional de Energía Atómica

Resumen

Este informe presenta la herramienta de visualización de audio scopeRC, desarrollada en Python haciendo uso de las librerías pyaudio para la adquisición de sonido, y pyqtgraph para la visualización en tiempo real.

Se recomienda usar scopeRC.py desde terminal con cualquier plataforma, provisto que se tengan instalados Python y las librerías mencionada. Alternativamente, existe una versión *standalone* con extensión **.exe** para ejecutar en Windows.

Introducción

El objetivo de este proyecto consiste en utilizar las librerías provistas por Python para crear un programa que permita visualizar el espectro de audio adquirido por el micrófono de una computadora y medir el valor de las frecuencias dominantes.

Del bucle de ejecución se pueden distinguir tres etapas individuales.

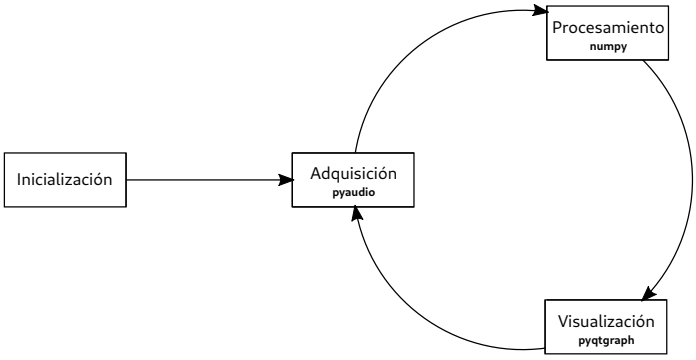

Figura 1: Diagrama de flujo de ejecución del programa.

- Adquisición: Detectar y decodificar el canal de entrada de audio. Se usa la librería **pyaudio**. [1]
- Procesamiento: De los datos adquiridos, realizar su FFT e identificar las frecuencias dominantes. Se usa la librería **numpy**. [2]
- Visualización: Presentar en tiempo real la información. Se usa la librería **pyqtgraph**. [3]

Adquisición

Adquirir los datos de la tarjeta de audio requiere usar la librería **pyaudio** para inicializar una interfaz de audio, llamada *stream*. Esta interfaz se ve definida por su formato, la tasa de muestreo, y el tamaño del buffer.

El formato de audio corresponde a la codificación discreta de los valores continuos de sonido medidos por el micrófono.

En el programa se usa el formato **pyaudio.paInt16**, que almacena cada muestra como una cadena de caracteres representando número entero de 16 bits en formato hexadecimal.

La tasa de muestreo es la inversa del intervalo de muestreo, es un concepto fundamental de procesamiento digital de señales y permite asociar frecuencias discretas con sus correspondientes frecuencias continuas. En el programa se usa una tasa de muestreo de 44,1 kHz.

El buffer es la cantidad de valores almacenados en un momento en memoria. En este programa corresponde a los valores de amplitud en el dominio temporal, que se actualiza en cada ciclo de ejecución.

El programa usa un buffer de 1024*4 bits. Notar que un buffer muy chico implica insuficientes datos para extraer información útil y un buffer muy grande implica demasiados datos para asegurar procesamiento en tiempo real.

Estos parámetros se usan para inicializar un objeto stream, en la etapa de adquisición de cada ciclo de ejecución se leen los valores almacenados en el buffer y se decodifican con la librería **struct**. [4]

Procesamiento

La etapa de procesamiento consiste en dar sentido en el dominio temporal a los datos obtenidos en la etapa de adquisición y usar el algoritmo Fast Fourier Transform provisto por la librería **numpy.fft** para extraer los componentes de frecuencia de estos datos. [5]

Para esto se inicializan los ejes de tiempo y frecuencia

a los cuales corresponderán los datos numéricos

$$t[n] = [0,\, t_s,\, \cdots,\, \mathrm{N}\, t_s] \\ f[k] = \left[0,\, f_s,\, \cdots,\, \tfrac{\mathrm{N}}{2} f_s\right] \tag{1}$$

donde f_s es la tasa de muestreo, $t_s = f_s^{-1}$ es el intervalo de muestreo, y N es el tamaño del buffer.[1]

Si llamamos al vector de valores adquiridos $x[n]$, el vector de sus componentes de frecuencia $X[k]$ son obtenidas por el algoritmo FFT, que realiza una transformada discreta de Fourier (optimizada), definida por

$$X[k] = \sum_{n=0}^{\mathrm{N}-1} x[n]\, e^{-j\frac{nk}{\mathrm{N}}} \tag{2}$$

Obteniendo así los valores $x[n]$ asociados a los tiempos $t[n]$ y los valores $X[k]$ asociadas a las frecuencias $f[k]$. Notar que las unidades de $x[n]$ y $X[k]$ son arbitrarias. Encontrar las frecuencias dominantes del sonido medido implica encontrar los máximos de $X[k]$, y asociarlos con su $f[k]$ correspondiente.

Los picos se identifican con la función **fpeaks** de la librería **scipy.signal**, tomando únicamente los que excedan determinado valor mínimo llamado *threshold*. En cada ciclo se almacenan las frecuencias de los picos que superan el threshold, ordenadas por amplitud. [6]

Visualización

El mayor desafío del programa es la visualización en tiempo real. La librería más usada para graficar en Python, **matplotlib**, no está optimizada para graficar en tiempo real. Por eso el programa usa **pyqtgraph** que está diseñada específicamente con ese propósito.

El uso de **pyqtgraph** es complejo y entrar en detalle en el mismo excede el propósito de la práctica, por lo que se considera como una caja negra que admite las trazas definidas por los pares ordenados $t[n]$, $x[n]$ para mostrar un osciloscopio y $f[k]$, $X[k]$ para mostrar un analizador de espectro. Admite además la tabla de picos para rotularlos en el gráfico correspondiente al analizador de espectro y usar la leyenda del mismo como frecuencímetro.

Es útil mencionar que **pyqtgraph** genera una ventana emergente con un menú contextual que permite modificar los ejes del gráfico, exportar el gráfico en formato mapa de bits o vectorial, o directamente exportar los valores numéricos presentados.

Las etapas de adquisición, procesamiento, y visualización de datos están comprendidas en una función llamada *update*. Se asocia la función a un temporizador de 25 ms provisto por la librería **QTimer**, lo que significa que el ciclo se ejecuta 40 veces por segundo.

El apéndice 1 contiene los códigos fuente correspondientes a la adquisición y procesamiento de datos así como el ciclo *update*.

Resultados

Para evaluar el rendimiento se probó el programa con una onda pura de 440 Hz, permitiendo exportar las siguientes gráficas.

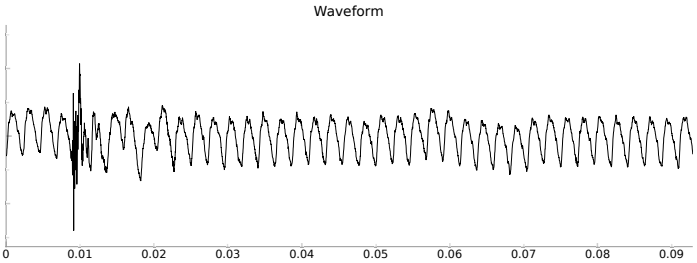

Figura 2: Forma de onda de una onda senoidal pura de 440 Hz exportada por el programa.

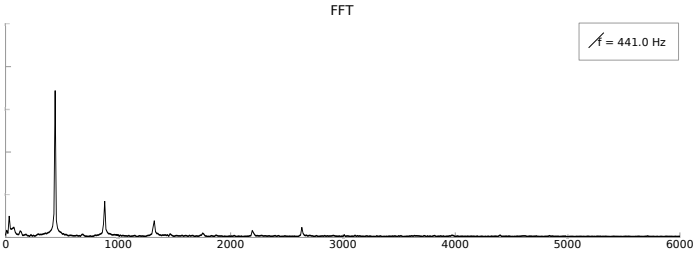

Figura 3: Espectro en frecuencias de una onda senoidal pura de 440 Hz exportada por el programa.

También se evaluó el rendimiento ante ondas compuestas, estudiando el espectro de una onda triangular de 440 Hz.

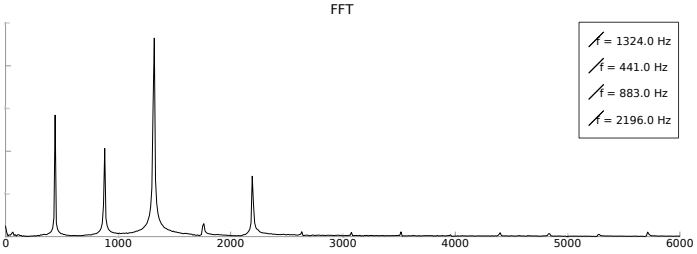

Figura 4: Espectro en frecuencias de una onda triangular de 440 Hz exportada por el programa.

[1] El motivo por el cual se toma únicamente el semieje positivo de frecuencia en el análisis de audio es porque la transformada de Fourier de una señal real es par en módulo e impar en fase, cualquier información en el semieje negativo es totalmente redundante.

En general, evaluando las mediciones del frecuencímetro ante ondas senoidales puras de diferentes frecuencias se determina que el instrumento tiene un margen de error de ±3 Hz en el rango de los [300, 800] Hz.

También se evaluó el comportamiento en tiempo real del instrumento, exponiéndolo a música y voz, el programa es capaz de seguir los sonidos que detecta el micrófono en tiempo real y sin atraso.

Discusión

El programa es una buena alternativa de bajo costo a herramientas profesionales para experimentos que requieran medir la frecuencia de algún sonido puro, y es perfectamente capaz de medir armónicos.

Existen limitaciones inherentes al instrumental usado, por ejemplo, los experimentos se realizaron usando el micrófono interno de la computadora el cual tiene sus limitaciones, por ejemplo, se ve en la figura 3 que aparecen armónicos a pesar de estar midiendo una onda senoidal pura.

Esto probablemente se deba a que el micrófono usado no es de alta gama, y puede tener resonancias parásitas afectando la medición, pero en caso de necesitar una medición precisa de armónicos se podría obtener una medición mas pura usando un micrófono externo.

El osciloscopio provisto por el instrumento es bastante rudimentario, pero no pretende ser más que una herramienta de visualización, el objetivo del programa es proveer un frecuencímetro y visualizador de espectro en tiempo real.

Referencias

[1] H. Pham, "Pyaudio documentation," 2006. PyAudio 0.2.11.

[2] "Numpy v1.19 manual," 2008.

[3] L. Campagnola, "Pyqtgraph documentation," 2020.

[4] "The python standard library." Python 3.8.4.

[5] B. V. V. Simon Haykin, *Señales y Sistemas*. Limusa Wiley, 2001.

[6] "Signal processing (scipy.signal)," July 2020.

Apéndice 1

Listing 1: Inicialización de Canal de Entrada de Audio

```
import pyaudio as aud
import struct

BUFF = 1024*4
FORMAT = aud.paInt16
RATE = 44100
bits = 16;
bitFormat = 'h'

audio = aud.PyAudio()
stream = audio.open(
        format = FORMAT,
        channels = 1,
        rate = RATE,
        input = True,
        output = False,
        frames_per_buffer = BUFF)
```

Listing 2: Inicialización de vectores tiempo, frecuencia, y Funciones Auxiliares de Picos

```
import numpy as np
from scipy.signal import find_peaks as fpeaks

t = np.arange(0, ts*BUFF, 1/RATE)
F = np.fft.fftfreq(BUFF, 1/RATE)
F = np.split(F,2)[0]
Threshold = 100

def getPeak(data, treshold):
        index = np.argmax(data)
        return index if data[index] > treshold else False

def sort_by(list1, list2):
        zipped_pairs = zip(list2.get('peak_heights'), list1)
        z = [x for _, x in sorted(zipped_pairs, reverse=True)]
        return z
```

Listing 3: Bucle de Ejecución y Temporizador

```
import pyqtgraph as pg

def update():
        #Adquisición
        data = stream.read(BUFF)
        data_int = struct.unpack(str(BUFF)+bitFormat, data)
        #Procesamiento
        data_F = np.fft.fft(data_int)/(2**bits)
        data_F = np.abs(data_F[0:len(data_F)//2])
        peaks, amp = fpeaks(data_F, height=Threshold)
        peaks = sort_by(peaks, amp)[0:4]
        #Visualización
        set_plotdata('waveform', t, data_int)
        set_plotdata("spectrum", F, data_F)
        set_plotdata("peaks", F[peaks], data_F[peaks])
        set_legend(F[peaks])

#Temporizador de actualización
timer = pg.Qt.QtCore.QTimer()
timer.timeout.connect(update)
timer.start(25)
```