Università degli Studi di Perugia

Dipartimento di Matematica e Informatica

Report di Laboratorio
Sistemi Elettronici e Sensori per l'Informatica

Studenti

Dedi Vittorio
Palladino Danilo

Anno Accademico 2024-2025

Sommario

Introduzione

Nel corso di questa relazione, esamineremo in dettaglio il programma che abbiamo sviluppato per elaborare i file audio forniti. Ci concentreremo sulle funzionalità, le scelte fatte e sul funzionamento di ciascuna parte del programma. Nel primo ed unico capitolo ci addentreremo nella spiegazione del codice. Lo analizzeremo passo dopo passo spiegando come sono state adottate determinate soluzioni. Inizieremo con il processo di lettura dei file audio. Proseguiremo con la rappresentazione grafica dei segnali per analizzarne la struttura. Esamineremo poi l'implementazione della trasformata di Fourier(FFT) per scomporre il segnale nel dominio delle frequenze. Analizzeremo l'identificazione dei picchi e la conversione delle frequenze in note musicali. Infine, discuteremo il filtro passa-basso e i risultati ottenuti, concludendo con il salvataggio dei files.

Capitolo 1: Analisi del programma

Per iniziare abbiamo deciso di presentare il codice introdotto in precedenza. Una spiegazione dettagliata verrà fornita al termine del capitolo, mentre qui di seguito è riportato il codice.

```
import soundfile as sf
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import find_peaks

filename1 = "diapason.wav"
filename2 = "distorta.wav"
filename3 = "pulita_semplice.wav"

data1, samplerate1 = sf.read(filename1)
data2, samplerate2 = sf.read(filename2)
data3, samplerate3 = sf.read(filename3)

def process_audio(data, samplerate):
   if data.ndim == 2:
       channel_1 = data[:, 0]
       channel_2 = data[:, 1]
       time = np.arange(len(channel_1)) / samplerate

       plt.figure(figsize=(10, 6))
       plt.subplot(2, 1, 1)
       plt.plot(time, channel_1, label="Canale 1")
       plt.title('Waveform - Canale 1')
       plt.xlabel('Tempo (s)')
       plt.ylabel('Ampiezza')
       plt.grid(True)

       plt.subplot(2, 1, 2)
       plt.plot(time, channel_2, label="Canale 2", color='r')
       plt.title('Waveform - Canale 2')
       plt.xlabel('Tempo (s)')
       plt.ylabel('Ampiezza')
       plt.grid(True)
       plt.tight_layout()
       plt.show()

   N = len(channel_1)
```

```
T = 1.0 / samplerate
yf = fft(channel_1)
xf = fftfreq(N, T)[:N // 2]

power = np.abs(yf[:N // 2]) ** 2

plt.figure(figsize=(10, 6))
plt.plot(xf, power, label="Potenza")
plt.title('Potenza della FFT')
plt.xlabel('Frequenza (Hz)')
plt.ylabel('Potenza')
plt.grid(True)
plt.show()

peaks, _ = find_peaks(power, height=0.1, distance=100)
peak_frequencies = xf[peaks]
peak_powers = power[peaks]

plt.figure(figsize=(10, 6))
plt.plot(xf, power, label="Potenza")
plt.plot(peak_frequencies, peak_powers, 'x', label="Picchi")
plt.title('Picchi nella Potenza della FFT')
plt.xlabel('Frequenza (Hz)')
plt.ylabel('Potenza')
plt.legend()
plt.grid(True)
plt.show()

peak_widths = np.diff(peak_frequencies)

def frequency_to_note(freq):
    A4_freq = 440.0
    semitone_ratio = 2 ** (1 / 12)
    semitone_distance = round(12 * np.log2(freq / A4_freq))
    notes = ['C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B']
    note = notes[semitone_distance % 12]
    octave = 4 + (semitone_distance // 12)
    return f"{note}{octave}"

peak_notes = [frequency_to_note(f) for f in peak_frequencies]

print("Frequenze dei picchi e le note corrispondenti:")
for freq, note in zip(peak_frequencies, peak_notes):
    print(f"Frequenza: {freq:.2f} Hz - Nota: {note}")

mask = np.ones(len(yf), dtype=complex)
mask[peaks[0]] = 0

filtered_yf = yf * mask

filtered_data = np.real(np.fft.ifft(filtered_yf))
```

```
    return filtered_data

filtered_data1 = process_audio(data1, samplerate1)
filtered_data2 = process_audio(data2, samplerate2)
filtered_data3 = process_audio(data3, samplerate3)

sf.write('filtered_diapason.wav', filtered_data1, samplerate1)
sf.write('filtered_distorta.wav', filtered_data2, samplerate2)
sf.write('filtered_pulita_semplice.wav', filtered_data3, samplerate3)
```

Iniziamo spiegando nel dettaglio tutti gli aspetti cruciali. all'inizio dobbiamo necessariamente importare le librerie necessarie: **`soundfile`** ci sarà utile per la lettura e la scrittura dei file audio, **`numpy`** per le operazioni numeriche, **`matplotlib`** per la visualizzazione grafica, **`scipy.fft`** per la trasformata di Fourier e **`scipy.signal`** per l'identificazione dei picchi.

Abbiamo tre file audio che sono come degli input del programma, questi sono: **`"diapason.wav"`**, **`"distorta.wav"`** e **`"pulita_semplice.wav"`**. I loro path relativi ci serviranno per accedervi durante l'esecuzione del programma Questi vengono letti con **`sf.read`**, che restituisce i dati del segnale e la frequenza di campionamento per ciascun file. Nel nostro caso i dati audio possiedono un segnale che è rappresentato come un array bidimensionale, con un canale per ogni colonna.

La funzione principale **`process_audio`** è quella che poi gestirà quasi tutti gli aspetti cruciali di questo programma, questa infatti elabora i dati del file audio fornito come input. Se il segnale è stereo, come nel nostro caso, i due canali vengono separati. Per rappresentare l'andamento del segnale nel tempo, viene calcolato un array **`time`** basato sulla frequenza di campionamento.

Si generano quindi due grafici della waveform, uno per il canale sinistro e uno per il destro, utilizzando **`matplotlib`**. Successivamente ci calcoliamo la trasformata di Fourier sul primo canale per analizzare le componenti in frequenza del segnale. La funzione **`fft`** esegue la trasformata di Fourier, e **`fftfreq`** calcola le frequenze corrispondenti. La potenza del segnale nel dominio delle frequenze viene determinata come il quadrato della magnitudine della FFT **`(np.abs(yf[:N // 2]) ** 2)`**. Questa potenza viene poi tracciata graficamente.

La funzione **`find_peaks`** della libreria **`scipy.signal`** viene impiegata per rilevare i picchi nell'array che rappresenta i valori di potenza associati alle frequenze del segnale. Il parametro **`power`** contiene questi valori di potenza, mentre **`height=0.1`** specifica una soglia minima per considerare un punto come un picco, eliminando così rumore o fluttuazioni. Il parametro **`distance=100`** garantisce che i picchi rilevati siano sufficientemente distanziati tra loro, evitando di segnalare come distinti picchi molto vicini. La funzione restituisce un array contenente gli indici dei picchi rilevati nell'array **`power`**. Questi indici vengono

successivamente utilizzati per estrarre informazioni fondamentali. Le frequenze corrispondenti ai picchi vengono ricavate dall'array **`xf`**, che rappresenta le frequenze associate ai punti calcolati dalla FFT, e vengono memorizzate in **`peak_frequencies`**. Conseguentemente i valori di potenza associati ai picchi vengono estratti dall'array **`power`** e memorizzati in **`peak_powers`**. In questo modo, il codice isola le frequenze dominanti nel segnale e ne misura l'intensità. Questi dati sono essenziali per le operazioni successive come la filtrazione o la conversione delle frequenze in note musicali.

Le frequenze dei picchi vengono poi convertite in note musicali utilizzando la funzione **`frequency_to_note`**, questa prende come input una frequenza in Hertz e la converte in una nota musicale utilizzando la frequenza standard di riferimento per il La4, che è 440 Hz. Per fare ciò, calcola la distanza in semitoni tra la frequenza data e il La4 utilizzando il logaritmo in base 2, che permette di gestire il rapporto tra le frequenze.

Successivamente, determina la nota musicale corrispondente selezionandola da una lista di 12 note (Do, Do#, Re, ecc.) in base alla distanza calcolata, con il modulo **`% 12`** che identifica la posizione nella scala musicale. L'ottava della nota viene calcolata dividendo la distanza totale in semitoni per 12, sommandola all'ottava base del La4.

Il programma stampa poi le frequenze dei picchi e le note corrispondenti, fornendo una chiara corrispondenza tra il contenuto spettrale del segnale e le note musicali. Per applicare un filtro passa-basso, viene applicata una maschera alla trasformata di Fourier del segnale audio per filtrare il picco principale. Inizialmente, si crea un array chiamato **`mask`**, lungo quanto **`yf`**, in cui tutti gli elementi sono impostati a 1 e sono di tipo complesso. Questo serve a mantenere tutte le frequenze inalterate.

Successivamente, si identifica il picco principale della trasformata (rappresentato dalla posizione **`peaks[0]`**) e si imposta a 0 il corrispondente valore della maschera. Questo annulla la componente del segnale associata a quella frequenza.

La maschera viene poi applicata alla trasformata di Fourier **`yf`** tramite una moltiplicazione elemento per elemento. In questo modo, tutte le frequenze restano invariate, tranne quella associata al picco principale, che viene eliminata.

Infine, si calcola l'inversa della trasformata di Fourier (**`ifft`**) per tornare al dominio del tempo, ricostruendo il segnale filtrato. Viene preso solo il valore reale del risultato, poiché i dati audio originali sono reali. La funzione restituisce il segnale filtrato.

Conclusioni

Con questa attività di laboratorio, abbiamo sviluppato un programma versatile che, grazie all’impiego delle librerie soundfile, numpy, scipy e matplotlib, è in grado di soddisfare le

richieste iniziali, offrendo una soluzione efficace per l'elaborazione del segnale audio digitale. Il programma ci ha permesso di elaborare file audio applicando diverse tecniche di analisi e filtraggio. Siamo riusciti a leggere e visualizzare i dati audio sotto forma di waveform, e ad applicare strumenti avanzati come la trasformata di Fourier, consentendoci di analizzare il segnale in modo più approfondito e di esaminare le sue componenti in frequenza.

Bibliografia

1. NumPy - NumPy User Guide: https://numpy.org/doc/stable/user/index.html#user Documentazione ufficiale della libreria NumPy. (Accesso: Dicembre 2024)
2. Ambrosi, G. (2024). Dispense sulle Trasformata di Fourier. Università degli Studi di Perugia
3. Matplotlib: https://matplotlib.org/stable/users/index.html# Documentazione ufficiale di Matplotlib (Accesso: Dicembre 2024)
4. Soundfile - Documentation. https://pypi.org/project/soundfile/
5. Scipy - Scipy Documentation User Guide. https://docs.scipy.org/doc/scipy/tutorial/index.html#user-guide La documentazione ufficiale di SciPy (Accesso: Dicembre 2024) (Accesso: Dicembre 2024)