FUNDAÇÃO ESCOLA TÉCNICA LIBERATO SALZANO VIEIRA DA CUNHA

LUCAS PILGER
SIBELLE SCHARLAU BOBRZYK
Turma: 4411

RELATÓRIO: WHISTLEBLOWING
TERCEIRO TRIMESTRE

Professor: Marcos Zuccolotto

Novo Hamburgo, RS.
Dezembro de 2024

1 O MISTÉRIO

Sussurros pelos corredores revelam uma enigmática reunião pré-conselho destinada a acelerar decisões cruciais. Os áudios anexos, envoltos em ruídos, guardam fragmentos do desconhecido. Apenas os verdadeiramente audaciosos ousarão decifrar esse mistério, utilizando uma ferramenta poderosa, digna de entidades investigativas como o FBI: a linguagem de programação Python.

2 LANTERNAS, LUPAS E FILTROS

2.1 FFT *(ou Luminol)*

Foi realizada uma análise do sinal utilizando a Transformada Rápida de Fourier (FFT), uma ferramenta matemática que permite visualizar o espectro de frequências do sinal. Essa verificação foi feita utilizando o código abaixo:

```
import numpy as np
import matplotlib.pyplot as plt
import wave
import scipy.fftpack as fft
```

A primeira parte do código importa as bibliotecas necessárias para o processamento do áudio e a visualização dos dados.

```
# Função para carregar o arquivo WAV
def carregar_audio(arquivo):
    with wave.open(arquivo, 'rb') as wav_file:
        # Obtendo as informações do arquivo de áudio
        n_channels = wav_file.getnchannels()  # Número de canais (1: mono, 2: estéreo)
        sample_rate = wav_file.getframerate()  # Taxa de amostragem (Hz)
        n_frames = wav_file.getnframes()  # Número de frames (amostras totais)

        # Lendo os frames de áudio
        frames = wav_file.readframes(n_frames)
        audio_data = np.frombuffer(frames, dtype=np.int16)

        # Se o áudio for estéreo, pega apenas o primeiro canal
        if n_channels == 2:
            audio_data = audio_data[::2]

        return audio_data, sample_rate
```

A função *carregar_audio* tem como objetivo carregar e processar o arquivo de áudio. Ela recebe como parâmetro o caminho do arquivo e, dentro da função, abre o arquivo *.wav* utilizando *wave.open()*, no modo de leitura binária *('rb')*. Com isso, a função consegue acessar informações importantes do arquivo, como o número de canais (mono ou estéreo), a taxa de amostragem (que define a quantidade de amostras por segundo) e o número total de frames, que representa o total de amostras do áudio. Em seguida, a função lê os dados do áudio com *wav_file.readframes(n_frames)* e os converte para um array de inteiros de 16 bits *(np.int16)*, que representam os valores de amplitude do áudio. Caso o áudio seja estéreo (possua dois canais), a função seleciona apenas o primeiro canal, utilizando *audio_data[::2]*, que pega os dados alternados do array. Finalmente, a função retorna os dados de áudio processados e a taxa de amostragem.

```
# Função para calcular a FFT e plotar
def plot_fft(audio_data, sample_rate, fmin=0, fmax=None):
    # Calculando a FFT
    fft_result = fft.fft(audio_data)

    # Calculando as frequências correspondentes
    n = len(audio_data)
    freqs = np.fft.fftfreq(n, d=1/sample_rate)

    # Pegando apenas a parte positiva das frequências
    pos_freqs = freqs[:n//2]
    pos_fft = np.abs(fft_result[:n//2])

    # Plotando a FFT
    plt.figure(figsize=(10, 6))
    plt.plot(pos_freqs, pos_fft)

    # Ajustando os limites do eixo X
    if fmax is None:
        fmax = np.max(pos_freqs)
    plt.xlim(fmin, fmax)  # Ajustando intervalo do eixo X em Hz

    # Títulos e rótulos
    plt.title("Espectro de Frequência do Áudio")
    plt.xlabel("Frequência (Hz)")
    plt.ylabel("Amplitude")
    plt.grid(True)
    plt.show()
```

A função *plot_fft* realiza o cálculo da Transformada Rápida de Fourier (FFT) e a visualização do espectro de frequência. Dentro dessa função, a FFT

é calculada sobre os dados de áudio utilizando *fft.fft(audio_data)*, o que converte o áudio do domínio do tempo para o domínio da frequência. Em seguida, as frequências associadas aos valores da FFT são calculadas com *np.fft.fftfreq(n, d=1/sample_rate)*, onde *n* é o número total de amostras do áudio e *d=1/sample_rate* é a distância entre amostras no tempo. O resultado dessa operação é um array de frequências. Como a FFT gera um espectro simétrico, a função seleciona apenas a metade positiva das frequências, ou seja, as primeiras *n//2* posições de *freqs* e *fft_result*, que representam as frequências de interesse.

Em seguida, a função utiliza *plt.plot()* para criar o gráfico, onde no *eixo X* são plotadas as frequências e no *eixo Y* a amplitude das frequências. O gráfico é ajustado para mostrar o intervalo de frequências entre *fmin* e *fmax*, que são passados como parâmetros para a função. Se o valor de *fmax* não for especificado, ele é automaticamente definido como o valor máximo das frequências positivas.

```
# Caminho do arquivo WAV
arquivo_audio = 'vazado_3.wav'

# Carregando o áudio
audio_data, sample_rate = carregar_audio(arquivo_audio)

# Plotando a FFT e limitando o eixo X entre 0 e 2000 Hz
plot_fft(audio_data, sample_rate, fmin=0, fmax=5100)
```

Por fim, a execução principal do código carrega o arquivo de áudio especificado, utilizando a função *carregar_audio* para ler os dados e a taxa de amostragem do arquivo. Em seguida, a função *plot_fft* é chamada para calcular a FFT do áudio e plotar o espectro de frequência. O intervalo de frequências é ajustado para ser mostrado entre 0 e 5100 Hz, conforme os parâmetros passados para a função.

A partir do código descrito acima, durante a análise, foram observados picos de energia significativos em frequências muito baixas (graves) e muito altas (agudas). Esses picos indicaram a presença de ruídos dominantes nessas faixas,

que estavam fora da faixa de interesse do áudio, que é o que contempla a voz humana escondida.

Figura 1: FFT do áudio *Vazado_1.wav*

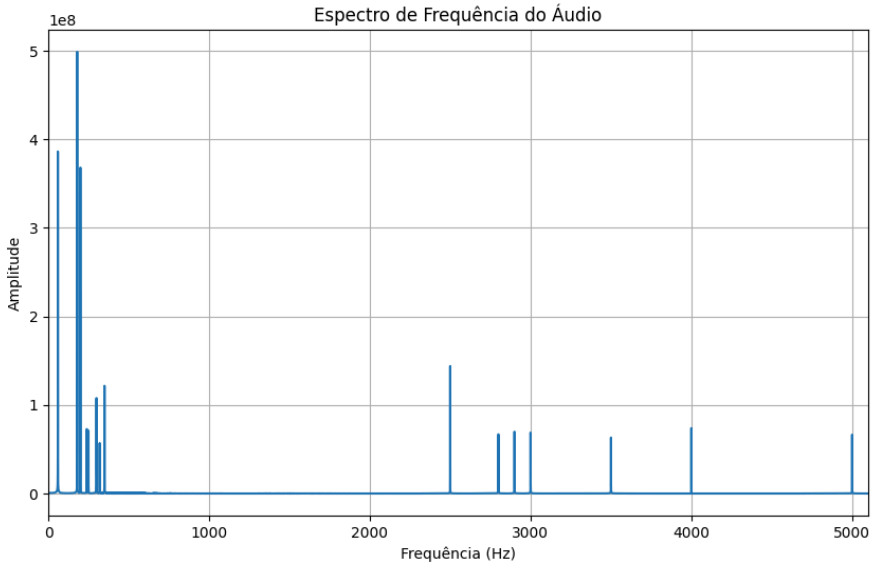
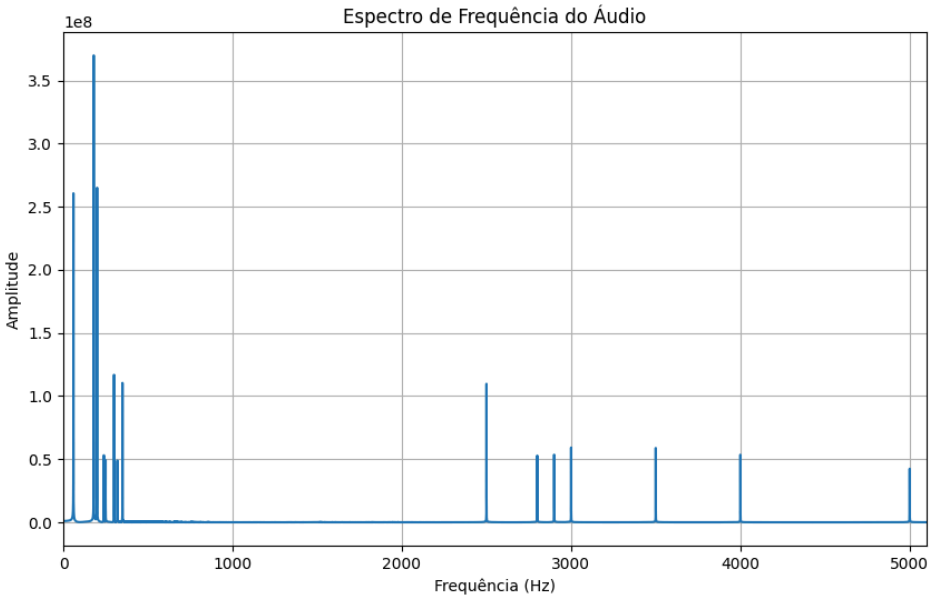


Figura 2: FFT do áudio *Vazado_3.wav*



Ao verificar esses picos de energia na FFT, ficou claro que os ruídos nas frequências graves e agudas estavam interferindo no sinal principal, gerando distorções e comprometendo a qualidade do áudio. Diante disso, foi decidido utilizar

um filtro passa-faixa, que é eficaz para atenuar ou eliminar essas frequências indesejadas, permitindo apenas a passagem das frequências dentro da faixa de interesse. Assim, o filtro passa-faixa foi aplicado para remover os ruídos agudos e graves, resultando em um áudio mais limpo e adequado para análise.

2.2 FIR *versus* IIR

A escolha entre filtros FIR *(Finite Impulse Response*) e IIR (*Infinite Impulse Response*) foi baseada na necessidade de garantir que o áudio fosse suficientemente claro para a transcrição da mensagem, sem a exigência de perfeição sonora. O filtro FIR foi selecionado inicialmente devido à sua fase linear e estabilidade. Embora o filtro IIR seja mais eficiente em termos de custo computacional, ele pode introduzir distorções de fase e instabilidade devido ao *feedback*, o que poderia prejudicar a legibilidade do áudio.

Ambos os filtros foram avaliados empiricamente, testando suas características de desempenho e verificando, na prática, qual deles oferece a melhor qualidade. A partir dessa análise empírica, a escolha final foi pelo filtro FIR, que, embora mais custoso computacionalmente, garantiu a maior confiabilidade e a clareza necessária para a transcrição da mensagem.

2.3 Janelas

As janelas de filtro são funções multiplicativas aplicadas ao sinal antes de sua transformação, com o objetivo de minimizar os efeitos de descontinuidade nas extremidades e reduzir o vazamento espectral. Testamos várias janelas para avaliar sua eficácia na atenuação de frequências indesejadas e no custo computacional.

A janela retangular, por ser simples, tem o menor custo computacional, mas resulta em maior vazamento espectral, o que pode comprometer a clareza do sinal. A janela *Hamming* e a *Hanning* ainda não oferecem o nível de filtragem necessário para garantir a qualidade desejada. A janela *Bartlett*, mais suave, também foi testada, mas não proporcionou uma redução significativa no vazamento em comparação com outras opções.

A janela *Blackman-Harris* foi escolhida pois minimiza as frequências indesejadas fora da faixa de passagem e melhora a qualidade do sinal. Ela ofereceu

o melhor equilíbrio entre eficiência na filtragem e qualidade do áudio, sendo a mais adequada para garantir que a mensagem fosse transcrita de forma clara. A decisão levou em consideração tanto a eficácia na filtragem quanto o custo computacional do sistema.

2.4 Bandas de passagem e de rejeição

A banda de passagem é o intervalo de frequências que o filtro permite atravessar com mínima atenuação, enquanto a banda de rejeição compreende as frequências que o filtro bloqueia ou atenua significativamente. No trabalho, essas bandas foram configuradas para cada arquivo de áudio com base na análise espectral e também por tentativa e erro durante os testes práticos.

Por exemplo, no arquivo *vazado_01.wav*, foi aplicado um filtro passa-faixa com limites entre 700 Hz e 1500 Hz. Essa configuração foi escolhida para permitir a passagem de frequências relevantes ao sinal de interesse, rejeitando frequências abaixo de 700 Hz e acima de 1500 Hz, que correspondiam aos ruídos graves e agudos identificados na análise espectral. Já no caso do arquivo *vazado_03.wav*, o filtro foi ajustado para uma banda de passagem entre 600 Hz e 2200 Hz, ampliando o intervalo de frequências permitidas devido às características particulares desse sinal.

3 JUNTANDO AS PEÇAS

A filtragem foi implementada utilizando a linguagem de programação Python. A seguir, é apresentada uma descrição do desenvolvimento do código-fonte:

3.1 Importação das bibliotecas

```
import sys
import numpy as np
import soundfile as sf
import matplotlib.pyplot as plt
from scipy.signal import firwin, lfilter, welch
import sounddevice as sd
```

- ***NumPy:*** fornece funções matemáticas e operações vetoriais.
- ***SoundFile:*** leitura e escrita de arquivos de áudio em diversos formatos.
- ***matplotlib:*** visualização gráfica.
- ***Scipy.signal:*** subpacote do SciPy utilizado para processamento de sinais, incluindo filtragem transformadas e outros métodos de análise.
    - ***firwin:*** cria os coeficientes de um filtro FIR.
    - ***lfilter:*** aplica o filtro FIR aos dados do áudio.
    - ***welch*****:** calcula a densidade espectral de potência.
- ***SoundDevice:*** reprodução e gravação de áudios.

3.2 Aplicação do filtro passa-faixa FIR ao áudio

```
def apply_fir_bandpass_filter(audio, lowcut, highcut, sr, numtaps):
    nyquist = 0.5 * sr  # Frequência de Nyquist
    low = lowcut / nyquist
    high = highcut / nyquist

    # Filtro FIR com uma janela Blackman-Harris
    fir_coefficients = firwin(numtaps, [low, high], pass_zero=False, window='blackmanharris')

    # Aplicar o filtro passa-faixa
    filtered_audio = lfilter(fir_coefficients, 1.0, audio)

    return filtered_audio
```

A função *apply_fir_bandpass_filter* aplica um filtro passa-faixa FIR (*Finite Impulse Response)* ao áudio, utilizando uma janela *Blackman-Harris* para calcular os coeficientes do filtro. Ela recebe o áudio, as frequências de corte, a taxa de amostragem e o número de coeficientes como parâmetros, e retorna o áudio filtrado.

3.3 Função para salvar o áudio filtrado em um arquivo

```
def save_audio(filtered_audio, sr, filename):
    sf.write(filename, filtered_audio, sr)
```

A função *save_audio* salva o áudio filtrado em um arquivo WAV utilizando a biblioteca *soundfile*. Ela recebe o áudio filtrado, a taxa de amostragem e o nome do arquivo de destino.

3.4 Função para carregar e processar áudio

```
def load_and_process_audio(audio_path, lowcut, highcut, sr, numtaps):
    audio, sr = sf.read(audio_path)
    audio = np.nan_to_num(audio, nan=0.0, posinf=0.0, neginf=0.0)
    filtered_audio = apply_fir_bandpass_filter(audio, lowcut, highcut, sr, numtaps)
    filtered_audio = filtered_audio / np.max(np.abs(filtered_audio))  # Normalização
    return audio, filtered_audio, sr

# Definição de limites
audio_paths = {
    "vazado_01": {"path": "vazado_1.wav", "lowcut": 700, "highcut": 1500},
    "vazado_03": {"path": "vazado_3.wav", "lowcut": 600, "highcut": 2200}
}
numtaps = 201  # Número de coeficientes
```

A função *load_and_process_audio* começa carregando o áudio usando a função *sf.read*, que lê o arquivo especificado no caminho e retorna duas variáveis: o áudio em forma de array de amostras e a taxa de amostragem (*sr*). A taxa de amostragem define quantas amostras são capturadas por segundo no áudio. Em seguida, a função utiliza *np.nan_to_num* para tratar quaisquer valores inválidos que possam existir no áudio, como *NaN* ou infinitos, substituindo-os por zero. Isso é importante para evitar erros durante o processamento.

Após essa etapa, a função chama a função *apply_fir_bandpass_filter*, que aplica um filtro FIR passa-faixa ao áudio. O filtro tem como parâmetros as frequências de corte, *lowcut* e *highcut*, que definem a faixa de frequências permitidas pelo filtro, e o número de coeficientes do filtro, *numtaps*, que determina a precisão e o alcance.

Após a aplicação, o áudio filtrado é normalizado para garantir que o seu valor máximo absoluto não ultrapasse 1.0. Isso é feito dividindo o áudio filtrado pelo seu valor máximo absoluto, utilizando a expressão *filtered_audio / np.max(np.abs(filtered_audio))*.

3.5 Função para plotar os gráficos

```
def plot_graphs(audio, filtered_audio, sr):
    time = np.linspace(0, len(audio) / sr, num=len(audio))

    plt.figure(figsize=(14, 6))
    plt.subplot(2, 1, 1)
    plt.plot(time, audio, color='blue')
    plt.title("Waveform Original", fontsize=20)
    plt.subplot(2, 1, 2)
    plt.plot(time, filtered_audio, color='green')
    plt.title("Waveform Filtrado", fontsize=20)
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(14, 10))
    plt.subplot(2, 1, 1)
    plt.specgram(audio, NFFT=2048, Fs=sr, noverlap=1024, cmap='viridis')
    plt.title("Espectrograma Original", fontsize=20)
    plt.subplot(2, 1, 2)
    plt.specgram(filtered_audio, NFFT=2048, Fs=sr, noverlap=1024, cmap='viridis')
    plt.title("Espectrograma Filtrado", fontsize=20)
    plt.tight_layout()
    plt.show()

    f_audio, Pxx_audio = welch(audio, sr, nperseg=1024)
    f_filtered, Pxx_filtered = welch(filtered_audio, sr, nperseg=1024)

    plt.figure(figsize=(14, 6))
    plt.semilogy(f_audio, Pxx_audio, label='Original')
    plt.semilogy(f_filtered, Pxx_filtered, label='Filtrado')
    plt.title("Densidade Espectral de Potência", fontsize=20)
    plt.xlabel("Frequência (Hz)")
    plt.ylabel("Densidade de Potência")
    plt.legend()
    plt.tight_layout()
    plt.show()
```

A função *plot_graphs* tem como objetivo gerar visualizações gráficas que ajudam a analisar as diferenças entre o áudio original e o áudio filtrado. Ela exibe três tipos principais de gráficos: o *waveform* (forma de onda), o espectrograma e a densidade espectral de potência.

Primeiro, no gráfico *waveform*, ela cria um vetor de tempo *time* para as amostras de áudio com base na taxa de amostragem (*sr*) e no número de amostras do áudio. O vetor de tempo é calculado com a fórmula *np.linspace(0, len(audio) / sr, num=len(audio))*, que cria uma sequência de tempos que correspondem a cada amostra do áudio.

Em seguida, a função cria o espectrograma dos áudios. O espectrograma é uma representação gráfica das frequências presentes no áudio ao longo do tempo. Ele é feito usando *plt.specgram()*, que recebe o áudio, a taxa de amostragem (Fs=sr), o número de pontos de FFT (*NFFT=2048*), e o número de pontos de sobreposição entre os segmentos de tempo (*noverlap=1024*). Essa função gera uma imagem onde o eixo x representa o tempo, o eixo y as frequências e as cores indicam a intensidade de cada frequência em cada ponto no tempo.

A terceira visualização gerada é a densidade espectral de potência, que mostra quanta energia (ou potência) do sinal está presente em cada faixa de frequência. Esse gráfico é produzido com a função *welch*, que recebe o áudio, a taxa de amostragem *(sr)* e o número de pontos em cada segmento da análise (definido com *nperseg=1024*). Ela retorna os valores de frequência *f_audio* e a densidade de potência *Pxx_audio* para o áudio original e de forma semelhante para o áudio filtrado.

3.6 Reprodução e informações adicionais

```python
# Função para reproduzir áudio
def play_audio(audio, sr, is_filtered=False):
    sd.play(audio, sr)
    sd.wait()

    if is_filtered:
        display_audio_info(audio, sr)

# Função para exibir informações sobre o áudio filtrado
def display_audio_info(audio, sr):
    duration = len(audio) / sr
    print(f"\nDuração do áudio: {duration:.2f} segundos")
    f_audio, Pxx_audio = welch(audio, sr, nperseg=1024)
    max_freq = f_audio[np.argmax(Pxx_audio)]  # Frequência com maior potência
    print(f"Frequência dominante (máxima potência): {max_freq:.2f} Hz")
    print(f"Desvio padrão do áudio: {np.std(audio):.4f}")
    print(f"Taxa de amostragem do áudio: {sr} Hz")
```

A função play_audio reproduz o áudio usando a biblioteca *sounddevice*. Ela também chama a função *display_audio_info* se o áudio for filtrado, para exibir informações adicionais sobre o áudio filtrado:

1. **Duração do áudio:** a duração é calculada com base no número total de amostras do áudio e na taxa de amostragem (*sr*). A fórmula usada é *duration = len(audio) / sr*, que divide o número de amostras pelo número de amostras por segundo (taxa de amostragem). O resultado é o tempo total em segundos que o áudio dura.
2. **Frequência dominante:** a frequência dominante é a frequência com a maior potência no áudio, ou seja, a frequência que mais contribui para o espectro de energia.
3. **Desvio padrão do áudio:** o desvio padrão é uma medida da dispersão das amostras do áudio em relação à média. Ele é calculado com a função *np.std(audio)*, que retorna o desvio padrão dos valores das amostras de áudio.

3.7 Execução principal que salva o áudio filtrado e menu

```
# Execução principal
if __name__ == "__main__":
    while True:
        print("\nEscolha o áudio para processar:")
        for idx, key in enumerate(audio_paths.keys()):
            print(f"{idx + 1} - {key}")
        print("0 - Sair")

        choice = input("Digite sua escolha: ")
        if choice == "0":
            print("Encerrando o programa.")
            break

        if choice not in map(str, range(1, len(audio_paths) + 1)):
            print("Opção inválida. Tente novamente.")
            continue

        selected_audio = list(audio_paths.keys())[int(choice) - 1]
        audio, filtered_audio, sr = load_and_process_audio(
            audio_paths[selected_audio]["path"],
            audio_paths[selected_audio]["lowcut"],
            audio_paths[selected_audio]["highcut"],
            None,
            numtaps
        )

        while True:
            print("\nEscolha uma ação:")
            print("1 - Reproduzir áudio original")
            print("2 - Reproduzir áudio filtrado")
            print("3 - Exibir gráficos")
            print("0 - Voltar ao menu de seleção de áudio")

            action = input("Digite sua escolha: ")
            if action == "0":
                break
            elif action == "1":
                play_audio(audio, sr)
            elif action == "2":
                play_audio(filtered_audio, sr, is_filtered=True)
            elif action == "3":
                plot_graphs(audio, filtered_audio, sr)
            else:
                print("Opção inválida. Tente novamente.")

        save_audio(filtered_audio, sr, f"{selected_audio}_filtrado.wav")
        print(f"Áudio filtrado salvo como {selected_audio}_filtrado.wav")
```

Na execução principal do código, o programa começa solicitando ao usuário que escolha um dos áudios para processar. O usuário é apresentado com duas opções de áudio, *vazado_01* e *vazado_03,* que são identificadas por um número. Cada áudio tem seus parâmetros definidos, como os limites de frequência *(lowcut e highcut)*, e o usuário pode escolher qual áudio deseja processar, ou optar por sair ao digitar 0. Se uma escolha válida for feita, o áudio é carregado e o processo de

filtragem é iniciado. Caso contrário, o programa solicita que o usuário insira uma opção válida.

Após selecionar um áudio, o programa entra em um novo menu, que oferece quatro opções de ação. A primeira opção permite ao usuário reproduzir o áudio original, ou seja, o áudio sem nenhum filtro aplicado. A segunda opção permite reproduzir o áudio já filtrado. A terceira opção exibe gráficos que mostram a forma de onda e o espectrograma dos áudios original e filtrado, além da densidade espectral de potência de ambos. A quarta opção retorna o usuário ao menu anterior, onde ele pode selecionar outro áudio para processar.

Se o usuário optar por uma das opções de reprodução ou exibição de gráficos, o programa executa a função correspondente, seja para tocar o áudio ou gerar os gráficos. Ao final de cada ciclo de ações, o áudio filtrado é salvo em um novo arquivo com o *sufixo _filtrado.wav*, garantindo que a versão processada do áudio seja armazenada.

4 CONFISSÕES

4.1 Transcrições

A seguir, são apresentadas as transcrições de cada áudio.

4.1.1 *Vazado_1.wav*

Todos esses que aí estão
Atravancando meu caminho,
vocês passarão...
Eu passarinho!

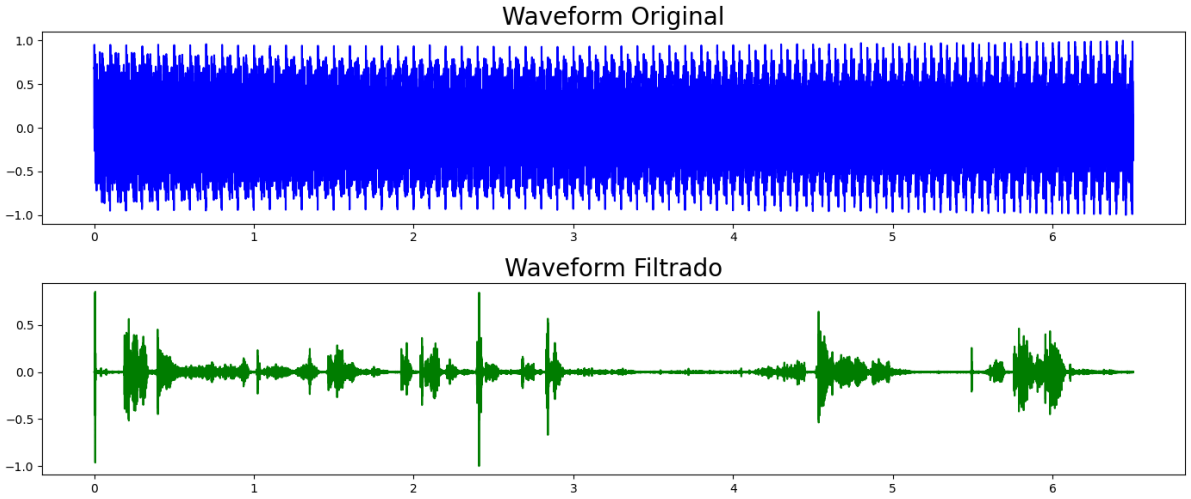
4.1.2 *Vazado_3.wav*

Isso de querer ser exatamente aquilo que a gente é, ainda vai nos levar além.

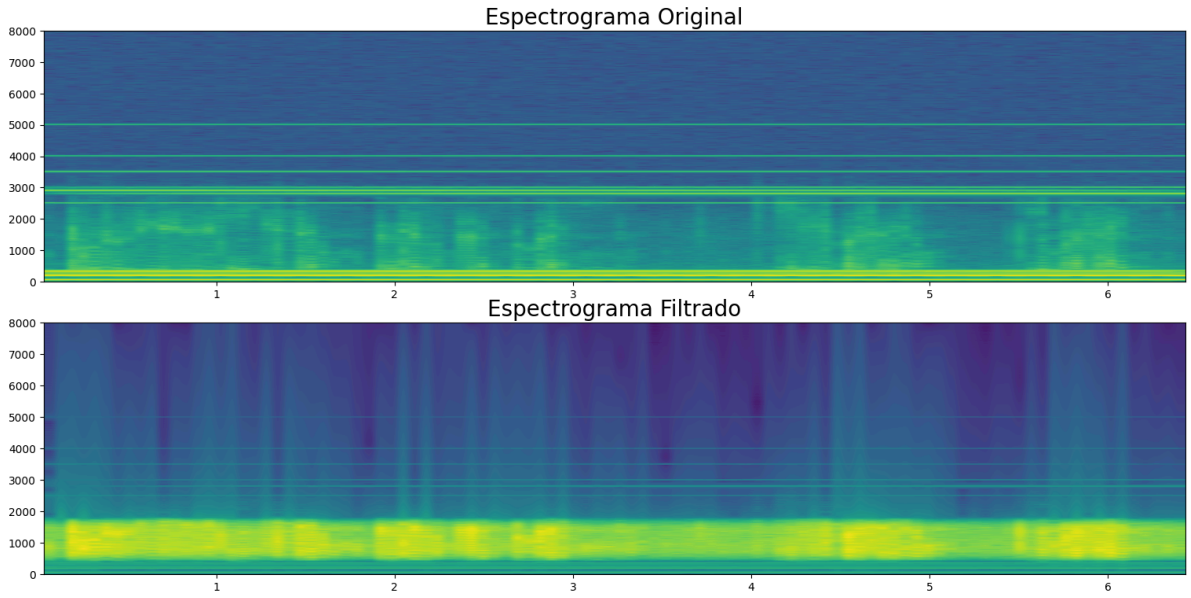
4.2 Gráficos

4.2.1 *Vazado_1.wav*

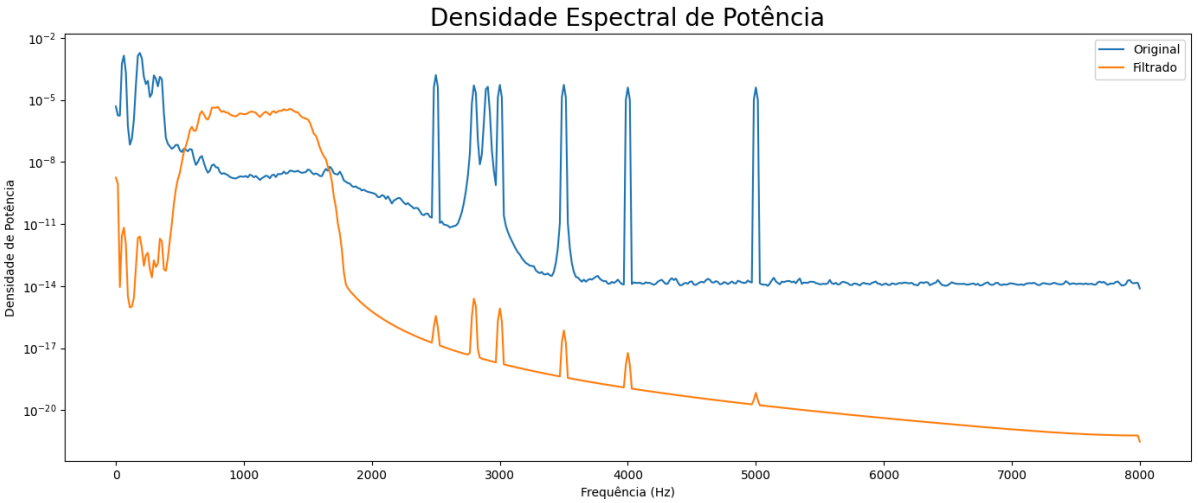
4.2.1.1 *Waveform*: a partir das formas de ondas, é possível perceber que houve uma diminuição do ruído presente no áudio original.



4.2.1.2 Espectograma: como verificado no gráfico abaixo, as frequências fora da banda de passagem estabelecidas foram removidas mantendo, em suma, a locução humana.

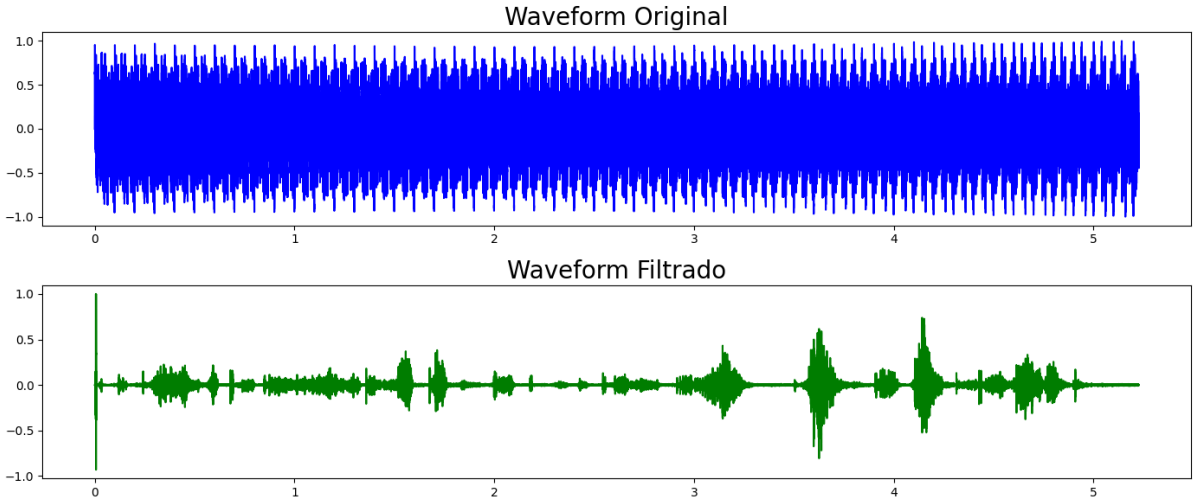


4.2.1.3 Densidade espectral de potência: observa-se que as frequências fora da banda de passagem definida pelo filtro foram significativamente atenuadas no áudio filtrado.

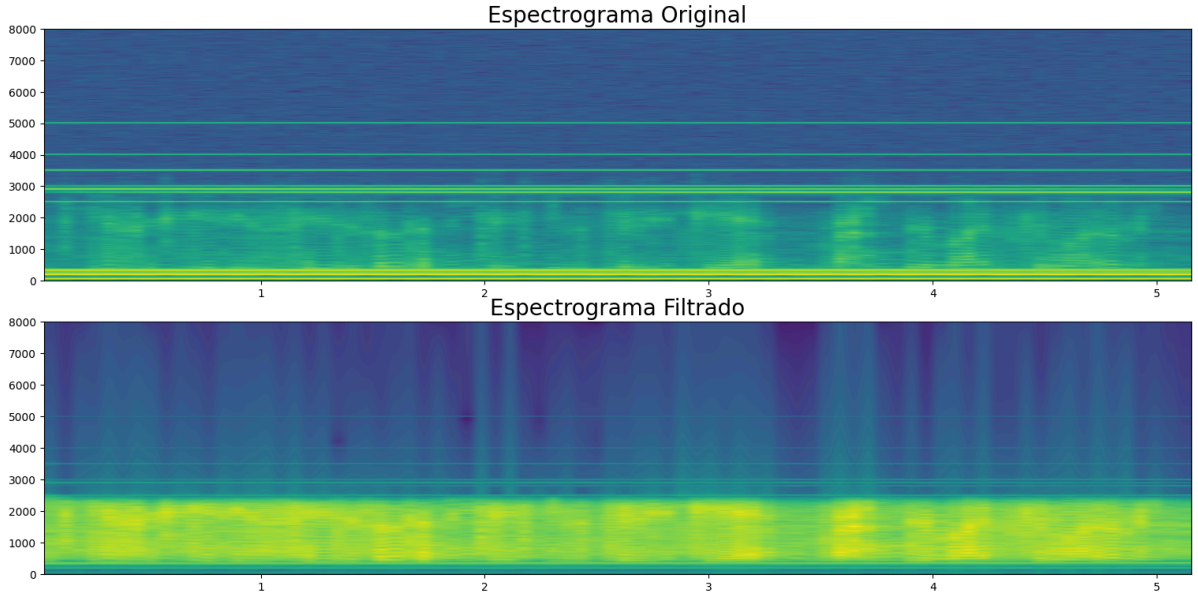


4.2.2 *Vazado_3.wav*

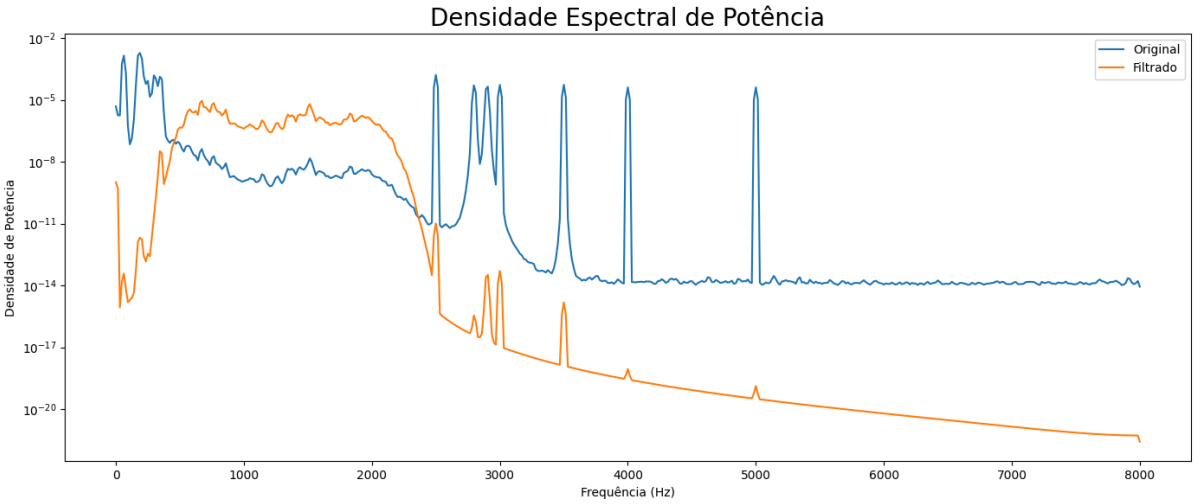
4.2.2.1 *Waveform*: a partir das formas de ondas, é possível perceber que houve uma diminuição do ruído presente no áudio original.



4.2.2.2 Espectograma: como verificado no gráfico abaixo, as frequências fora da banda de passagem estabelecidas foram removidas mantendo, em suma, a locução humana.



4.2.2.3 Densidade espectral de potência: observa-se que as frequências fora da banda de passagem definida pelo filtro foram significativamente atenuadas no áudio filtrado.



5 CASO ARQUIVADO

Os corredores da Liberato sempre foram mais do que simples passagens. Ao longo dos anos, acumularam segredos que ecoavam entre suas paredes, como se estivessem testando a atenção dos que por ali passavam. Durante semanas, algo incomum pairava no ar: uma mensagem oculta, um mistério que envolvia os rumores do pré-conselho.

Foi então que os alunos da Eletrônica, habituados a trabalhar com o incompreensível, decidiram enfrentar o enigma. No laboratório, onde as luzes fracas dos osciloscópios iluminavam rostos concentrados, eles começaram sua investigação. Não era uma simples curiosidade. E, aos poucos, a névoa foi se dissipando. As frequências começaram a se alinhar, até que as mensagens começaram a surgir. Mario Quintana e Paulo Leminski surgiram em palavras e versos.

Quando o mistério finalmente se revelou, o silêncio tomou finalmente conta do laboratório. Os corredores, que até então haviam sido cúmplices de um segredo, pareciam respirar aliviados. E os alunos, agora levavam consigo a certeza de que a Liberato sempre tem algo a dizer para quem se atreve a ouvir, mesmo que em sussurros entre seus corredores.