Projeto 1 - Processamento de Sinais multimídia

Amélia O. F. da S

1 Objetivos

O projeto tem como objetivos:

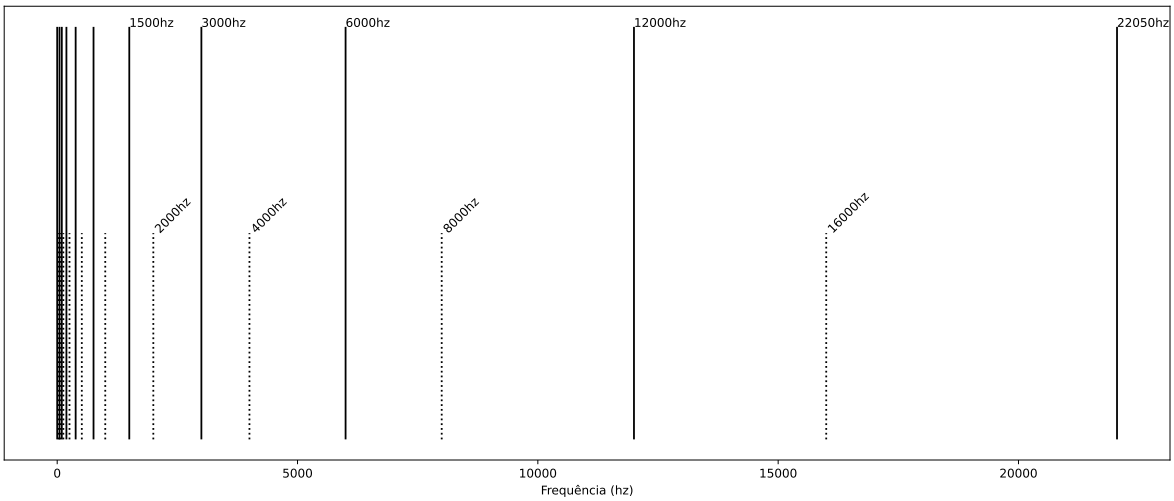
- Demonstração de domínio básico prático do conhecimento de filtros SLIT FIR via:
  - Implementação de um equalizador de bandas;
  - Implementação de um visualizador de energia de bandas.
- Demonstração de domínio básico teórico do conhecimento de filtros SLIT FIR pela descrição e justificativas teóricas das implementações.

2 Planejamento

2.1 Arquitetura de DSP

A fim cumprir os objetivos listados acima, proponho a arquitetura descrita na figura 3.

A aplicação processará sequencialmente as amostras de entrada e aplicará um filtro simplificado calculado a partir de 10 filtros passa-faixa dividindo o espectro entre as frequências requeridas no projeto. Para o cálculo dos valores do gráfico de barras, aplicaremos periodicamente uma FFT, somando as energias das faixas especificadas.



Pontilhadas: frequências "centrais"das bandas
Sólidas: frequências-limite das bandas

Figura 1: Bandas de filtragem

2.2 Detalhes e justificativa de implementação

Inicialmente propus a arquitetura ilustrada na figura 2, que consistia em aplicar separadamente os 10 filtros e guardar seus resultados em acumuladores para calcular a energia de cada banda.

Mesmo tendo uma complexidade maior que a FFT, intuitivamente imaginei que para a pequena quantidade de filtros que temos e para uma largura reduzida de filtro, este método poderia custar menos operações.

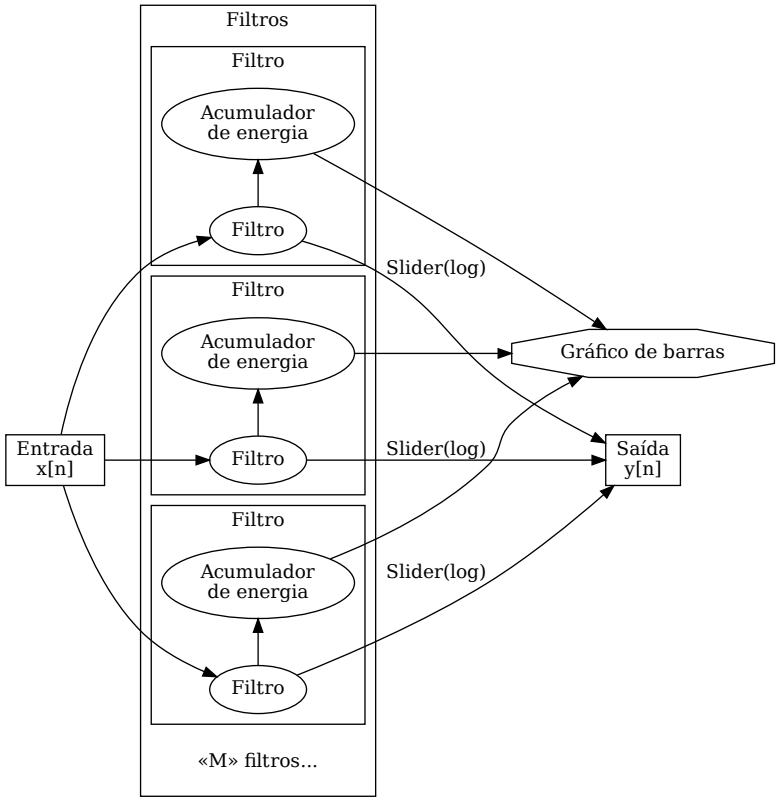


Figura 2: Má arquitetura proposta

Buscando uma comparação mais concreta entre essa proposta e a descrita acima, ilustrada na figura 3, fiz uma análise mais detalhada.

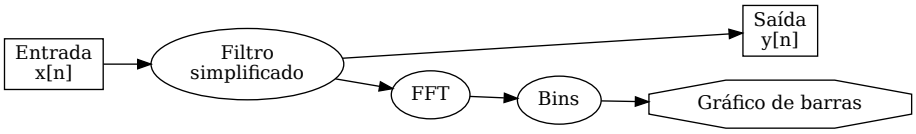


Figura 3: Nova arquitetura proposta

Na arquitetura sem FFT, para uma quantidade de amostras S e M filtros FIR de largura W, o número de operações a serem feitas para calcular todos os valores necessários será:

- Multiplicações: $S(WM)$
  - WM para multiplicação das amostras pelos coeficientes em $(\sum_i^W a_i x[n-i])$ e ponderação dos sliders;

    (Para um peso w, $w\sum_i^W a_i x[n-i] = \sum_i^W (a_i w)x[n-i]$, então pré-calculamos $a_i w$)
- Somas/Subtrações: $S(WM + M + 2M)$
  - $WM + M$ para os filtros:
    - WM para o somatório $\sum_i^W a_i x[n-i]$;
    - M para somar todos.

  - $2M$ para o acumulador de energia:
    * 1 soma para adicionar a amostra atual;
    * 1 subtração para remover a última amostra.

Para a arquitetura com FFT, teríamos, presumindo $S = 2^p$ e um número de amostragens do gráfico de energia ao longo do *chunk* de memória $G = 2^k, k < p$, gerando "pacotes"de $L = S/G$:

- Multiplicações: $S(W) + G\frac{L}{2}log_2 L$
  - W para multiplicação das amostras pelos coeficientes;
  - $\frac{L}{2}log_2 L$ para a FFT.
- Somas/subtrações: $S(W) + G(Llog_2 L + L)$
  - W para o somatório de $y[n]$.
  - $Llog_2 L$ para a FFT;
  - L para aglomeração das "bins"da FFT.

Presumindo 4096 amostras, podemos explorar as complexidades das duas implementações para diversas combinações dos parâmetros W e G.

Para **todas** as combinações exploradas (todos W entre 1 e 500, todos G entre 1 e 11), o método com FFT foi superior (vide `misc/operacoes.py`).

N_{naive}/N_{fft}	**Multiplicações**	**Somas**
Média	9.68	9.64
Máxima	9.98	10
Mínima	1.53	3.07

Tabela 1: Razão entre o número de operações da abordagem sem FFT e com FFT (N_{naive}/N_{fft})

Podemos perceber, porém, que mesmo usando 1/10 dos filtros (por juntá-los em um só), o método com a FFT **não é 10 vezes mais rápido**, ou seja, para menos filtros (ou um só), talvez a intuição inicial estivesse certa (não explorei essas possibilidades).

2.3 Arquitetura da aplicação

A aplicação se dividirá em três componentes principais:

- Provedor de informação dos sliders;
  - Codificado em Python;
  - Apresentará a interface de alteração dos filtros para o usuário;
  - Proverá os coeficientes do filtro para o processador de sinais via FIFO/Pipe UNIX.
- Host de filtragem;
  - Codificado em C;
  - Receberá coeficientes de filtragem via FIFO/Pipe UNIX;
  - Receberá um feed de som do PulseAudio, aplicará o filtro FIR caracterizado pelos parâmetros guardados e enviará um feed de som de volta ao PulseAudio;
- Visualizador de gráfico de barras.
  - Codificado em Python;
  - Receberá um feed de som do PulseAudio e gerará um gráfico de barras de intensidade de cada banda.

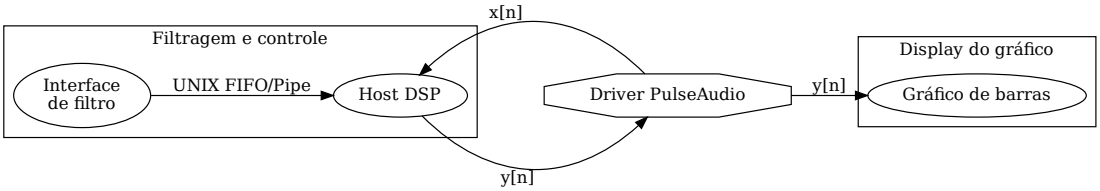


Figura 4: Processos e conexões relevantes

Destes processos, dois grupos são basicamente independentes:

- Processos de controle e filtragem
  - Lidam com a filtragem do áudio e configuração do filtro;
  - Lidam com a interação com o usuário.
- Processo de display do gráfico
  - Lida com o cálculo das intensidades de cada *bin* de frequências.
  - Lida com a ilustração das intensidades com um gráfico de barras.

O isolamento dos dois grupos, bem como o desenho do host de DSP de forma "reativa"(funciona continuamente até que alguma alteração venha da interface, com mínima interrupção) permite que o feed de som seja contínuo, sem falhas provindas de gargalos de processamento, e que o display visual, para o qual continuidade não é tão importante, possa lidar com o feed de som em seu próprio "ritmo".

3 Desenho de filtro

10 filtros passa-faixa de largura 100 foram desenhados para cobrir o espectro de acordo com a figura 1. Cada filtro for definido com base em sua resposta ao impulso $h[n] = \frac{sin(\omega_{c2}n) - sin(\omega_{c1}n)}{\pi n}$. Os parâmetros ω foram calculados a partir das frequências pela expressão $\omega = \frac{2f\pi}{f_s}$, que leva a frequência de nyquist para π, nosso ω máximo.

Para "juntar"os filtros, basta somar todas suas representações em resposta impulsiva (coeficientes a_i), pois sabemos que $\sum_i^n a_i x[n-i] + \sum_i^n b_i x[n-i] = \sum_i^n (a_i + b_i) x[n-i]$.

A resposta impulsiva do filtro resultante da soma de todos os filtros está ilustrada na figura 6.

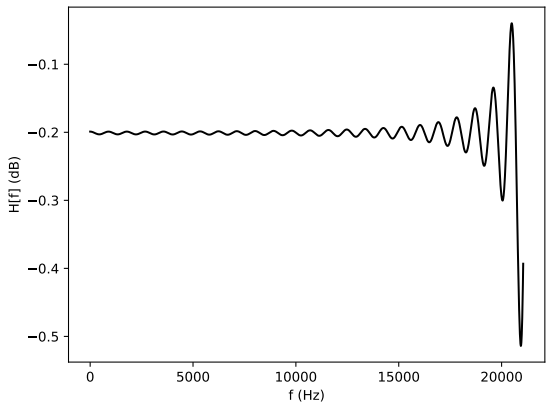


Figura 6: Resposta impulsiva do filtro final sem janelamento

Tanto no filtro individual como na resposta impulsiva do filtro final, temos imperfeições indesejáveis provindas da restrição dos filtros a 100 amostras. Para minimizar esse problema, os filtros foram multiplicados por funções de janelamento.

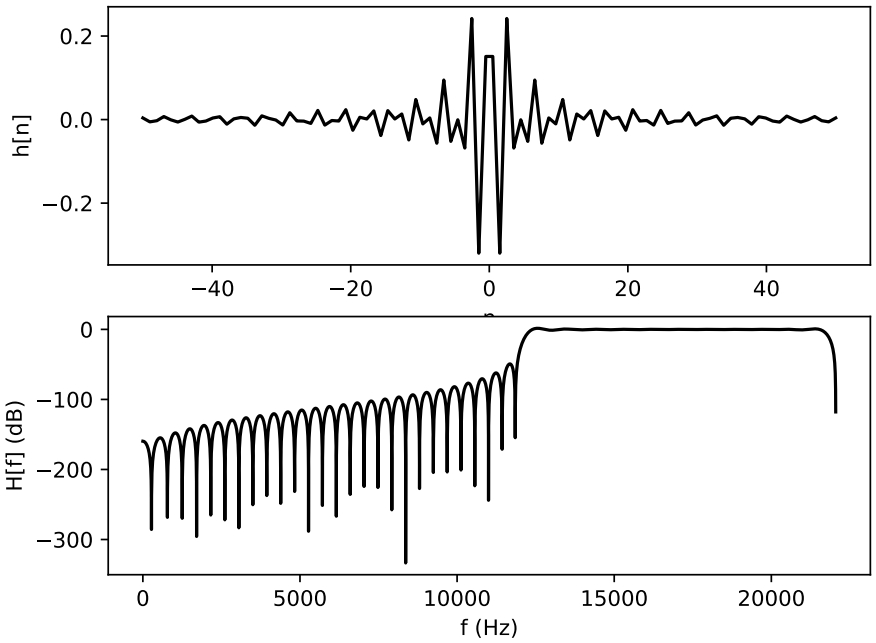


Figura 5: Filtro da faixa 12k-22k

A função de janelamento utilizada foi uma combinação (multiplicação) da janela de Blackmann-Nuttall (vide equação abaixo) com um janelamento exponencial simples ($e^{k\frac{n}{N}}$).

$$w[n] = a_0 - a_1 \cos\left(\frac{2\pi n}{N}\right) + a_2 \cos\left(\frac{4\pi n}{N}\right) - a_3 \cos\left(\frac{6\pi n}{N}\right)$$

$$a_0 = 0.3635819; \quad a_1 = 0.4891775; \quad a_2 = 0.1365995; \quad a_3 = 0.0106411$$

A janela resultante é bastante "lisa"(com poucas oscilações) e a redução da resposta em frequência fora da área de interesse é de quase 100dB!

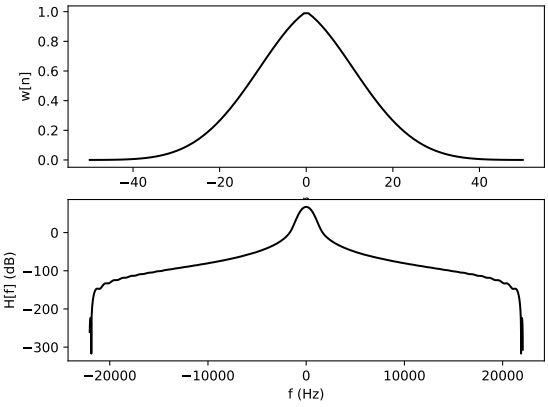


Figura 7: Função de janelamento utilizada

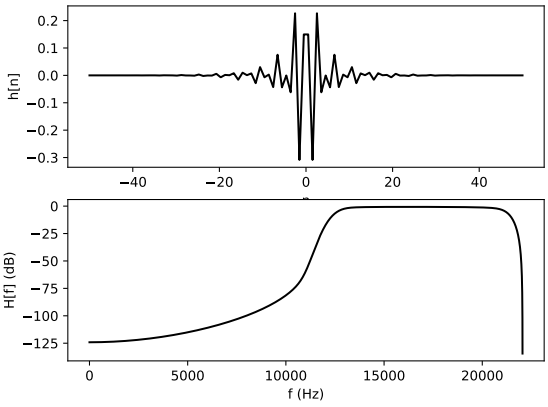


Figura 8: Filtro multiplicado pelo janelamento

A resposta impulsiva resultante dos novos filtros janelados possui bem menos oscilações indesejáveis, como perceptível na figura 9.

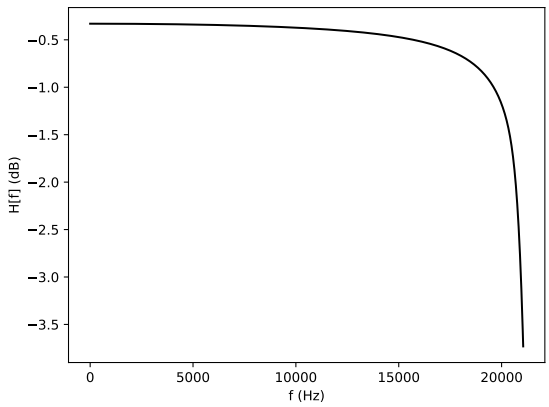


Figura 9: Resposta impulsiva do filtro final com janelamento

As transições entre bandas do filtro janelado são um pouco menos "bruscas"(*sharp*), mas em compensação, sofrem muito menos com "ringing".

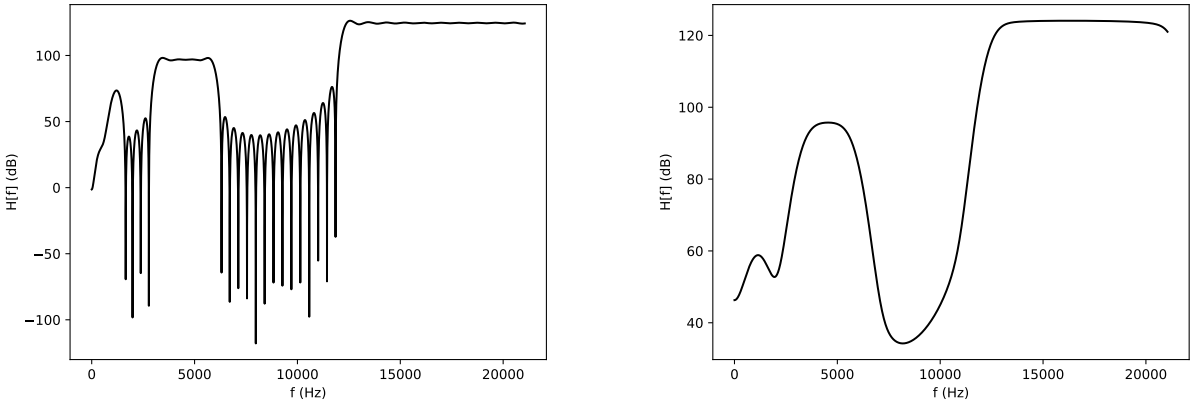


Figura 10: Resposta em frequência dos filtros sem e com janelamento
(intensificando as bandas ímpares por 2^i e atenuando as pares)

4 Considerações finais

Considero cumpridos os objetivos do projeto. Explorei os tópicos de interesse do projeto em profundidade adequada teoricamente e construí, com o conhecimento adquirido e visando demonstrá-lo, uma aplicação usável e funcional.

Todos os arquivos-fonte necessários para compilar a aplicação e para a geração deste relatório estão disponíveis (sob licença GPL) no repositório do GitHub abaixo:

`https://github.com/m3101/PSM_2022_3_1.`

`Obrigada pelo tempo e pelo conhecimento.`