INSTITUTO FEDERAL DE SANTA CATARINA

LUÍSA MACHADO

# FonPSNR: Proposta de Métrica Objetiva para Avaliação de Qualidade de Áudio

São José - SC

março/2024

# FONPSNR: PROPOSTA DE MÉTRICA OBJETIVA PARA AVALIAÇÃO DE QUALIDADE DE ÁUDIO

Monografia apresentada ao Curso de Engenharia de Telecomunicações do Campus São José do Instituto Federal de Santa Catarina para a obtenção do diploma de Engenheira de Telecomunicações.

Orientador: Prof. Marcos Moecke, Dr.

Coorientador: Prof. Roberto Wanderley da Nóbrega, Dr.

São José - SC

março/2024

Luísa Machado

# FonPSNR: Proposta de Métrica Objetiva para Avaliação de Qualidade de Áudio

Este trabalho foi julgado adequado para obtenção do título de Engenheira de Telecomunicações, pelo Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, e aprovado na sua forma final pela comissão avaliadora abaixo indicada.

São José - SC, março de 2024:

---

**Prof. Marcos Moecke, Dr.**
Orientador
Instituto Federal de Santa Catarina

---

**Prof. Roberto Wanderley da Nóbrega, Dr.**
Coorientador
Instituto Federal de Santa Catarina

---

**Prof. Fábio Alexandre de Souza, Dr.**
Instituto Federal de Santa Catarina

---

**Profª. Elen Macedo Lobato, Drª.**
Instituto Federal de Santa Catarina

# AGRADECIMENTOS

Os agradecimentos principais são direcionados a minha família, meu pai Vanderlei, minha mãe Luciana, minha irmã Juliana, meus avós Daniel e Maria Dorvalina, por todo amor, apoio e incentivo.

A todos os meus amigos, em especial a Karine, por estar comigo desde o ensino fundamental até hoje, nos melhores e piores momentos, por me motivar, sonhar e acreditar em mim sempre. Agradeço também aos amigos de graduação, em especial à Schaiana, Lucas, Anderson e Bruno.

Ao Daniel, por todo companheirismo, apoio, dedicação, respeito e união que tivemos durante esses quase sete anos. Não tenho como agradecer devidamente a todos os momentos que passamos juntos até aqui.

Agradeço a todos professores que tive até aqui. Um agradecimento especial a Marcos Moecke e Roberto Wanderley da Nóbrega que me orientaram nesse trabalho.

Por fim, gostaria de agradecer também as pessoas que perdi ao longo da minha vida, em especial minha tia Dircéia.

# RESUMO

Uma das etapas mais importantes antes de realizar a transmissão ou o armazenamento de um arquivo é a compressão deste, pois com essa etapa é possível otimizar recursos, como o tempo de transmissão e o uso de memória no armazenamento. Um áudio é uma informação muito sensível, na qual o tipo de codificação pode comprometer totalmente a qualidade do arquivo. Mensurar a qualidade de arquivos de áudio é uma tarefa bastante complexa, a métrica mais aceita atualmente é a MOS, uma métrica subjetiva com alto custo de aplicação. O objetivo do trabalho é propor e analisar uma métrica objetiva que incorpore um filtro ponderado por uma curva isofônica antes do cálculo da relação sinal-ruído de pico (PSNR), levando em conta os aspectos psicoacústicos da percepção auditiva humana na avaliação da qualidade de áudio.

**Palavras-chave**: Qualidade de áudio. Codificação de áudio. Codec. Vorbis. Opus.

# ABSTRACT

One of the most important steps before transmitting or storing a file is compressing it, because with this step it is possible to optimize resources, such as transmission time and memory usage in storage. Audio is very sensitive information, in which the type of encoding can completely compromise the quality of the file. Measuring the quality of audio files is a very complex task, the most accepted metric currently is MOS, a subjective metric with high application costs. The objective of the work is to propose and analyze an objective metric that incorporates a filter weighted by an isophonic curve before calculating the peak signal-to-noise ratio (PSNR), taking into account the psychoacoustic aspects of human auditory perception in the quality assessment audio.

**Keywords**: Audio quality. Audio coding. Codec. Vorbis. Opus.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS E SIGLAS

**AAC** *Advanced Audio Coding*.

**AOSV** Avaliação Objetiva de Sinais de Voz.

**CBR** *constant bit rate*.

**CELT** *Constrained Energy Lapped Transform*.

**Codec** codificador/decodificador.

**CSV** Comma-Separated Values.

**DB** decibéis.

**DI** *Distortion Index*.

**FAAC** *Freeware Advanced Audio Coder*.

**FFT** *Fast Fourier transform*.

**FIR** *Finite Impulse Response*.

**ITU-R** *International Telecommunication Union - Radiocommunication*.

**LP** *Linear Prediction*.

**MDCT** *Modified Discrete Cosine Transform*.

**MOQA** Medida Objetiva da Qualidade de Áudio.

**MOS** *mean opinion score*.

**MSE** *Mean-squared Error*.

**ODG** *Objective Difference Grade*.

**PEAQ** *Perceptual Evaluation of Audio Quality*.

**PSNR** *Peak Signal-to-Noise Ratio*.

**SNR** *Signal-to-Noise Ratio*.

**SPL** *sound pressure level*.

**THD** *total harmonic distortion*.

**VBR** *variable bit rate*.

**VoIP** *Voice over Internet Protocol*.

# SUMÁRIO

# 1 INTRODUÇÃO

No mundo das tecnologias há uma crescente busca por redução de custos, por exemplo, em uma transmissão de dados quanto maior for o tamanho desse dado maior será o tempo gasto para transmiti-lo ou armazená-lo, resultando em maior custo, por esta razão foram criados os codificadores de dados. De modo geral, um codificador tem como objetivo diminuir o tamanho de um arquivo, reduzindo o tempo de sua transmissão e, também, a quantidade de memória necessária para fazer o seu armazenamento, o que possibilita a redução de custos.

Com esse mesmo objetivo surgiram também os codificadores ou compressores de áudio, que de acordo com Spanias (2007), são algoritmos criados com o intuito de obter representações digitais mais compactas dos sinais de áudio com alta fidelidade para realizar a sua transmissão ou o seu armazenamento de forma mais eficiente. Sendo assim, esses compressores tem como principal objetivo reduzir ao máximo o número de bits utilizados na representação do sinal visando manter a sua qualidade, de modo que seja praticamente impossível distinguir o áudio de entrada original da saída codificada.

Codificadores de áudio podem ser classificados conforme o seu tipo de codificação, que pode ser *lossless* (sem perdas) ou *lossy* (com perdas) (SPANIAS, 2007). Os codificadores *lossless*, segundo a tradução do nome indica, não possuem perdas geradas no momento da compressão, ou seja, o áudio codificado é totalmente fiel ao original. Este tipo de codificador/decodificador (codec) trabalha em cima das redundâncias dos sinais no áudio, então, o tamanho do arquivo no final da compressão vai depender apenas da quantidade de informação redundante existente no áudio, sendo assim, quanto mais redundância possuir, menor será o tamanho do áudio codificado.

No entanto, nos codificadores *lossy* existem perdas causadas pela compressão, porém seu principal objetivo é que essas perdas devem alterar o mínimo possível de informação, de modo que o ouvido humano seja incapaz de detectar as diferenças entre o original e o comprimido. Por esta razão, para produzir um bom codificador com perdas é necessário conhecer primeiro o comportamento da audição humana, principalmente as suas características psicoacústicas, que auxiliam a detectar informações naturalmente ignoradas pelo ouvido humano.

Para a realização desse estudo foram selecionados dois codificadores de áudio, o Vorbis e o Opus. Todos esses codificadores foram implementados pela Xiph.Org Foundation, que é um grupo sem fins lucrativos com o objetivo de desenvolver *softwares* de código aberto e gratuitos para diversas aplicações.

O Vorbis, também conhecido como Ogg Vorbis, é um codec com perdas que é

otimizado para ser usado para comprimir tanto a fala quanto a música (XIPH.ORG, 2015). Uma curiosidade sobre o Vorbis, é que o serviço de *streaming* do Spotify (SPOTIFY, 2024) o usa na compressão de suas músicas, além dele, alguns jogos de *videogame* também o utilizam (XIPH.ORG, 2018).

O Opus, assim como o Vorbis, é um codificador com perdas. Este é um híbrido de outros dois codificadores mais antigos, o SILK que é do Skype e é otimizado para codificar a fala, e o *Constrained Energy Lapped Transform* (CELT) que, também, é da Xiph.Org Foundation e foi desenvolvido para tratar arquivos de música (XIPH.ORG, 2017). Portanto, o Opus é um compressor que trata fala e música obtendo uma boa qualidade em ambos.

Avaliar a qualidade de um áudio é uma tarefa bastante complexa, visto que existem várias características psicoacústicas que interferem no modo como o ouvido humano interpreta as informações sonoras captadas.

Atualmente, existe o modelo objetivo PEAQ definido pela *International Telecommunication Union - Radiocommunication* (ITU-R) que pode ser usada na avaliação de qualidade de áudios, dado que métricas objetivas tradicionais, como a PSNR, não são suficientes para classificar a qualidade de um áudio.

Segundo Palomar et al. (2008), a forma mais confiável de avaliação de qualidade para arquivos de áudios é feita utilizando as métricas subjetivas, como, por exemplo, a MOS, entretanto esse tipo de métrica consome bastante tempo e possui alto custo de implementação.

Tendo em vista estas informações, este trabalho propõem o desenvolvimento de uma métrica objetiva de avaliação de qualidade de áudio criando um filtro projetado a partir de uma curva isofônica associada com a relação sinal-ruído de pico (PSNR).

Existem alguns trabalhos que também se propuseram a desenvolver novas formas de analisar a qualidade de áudio, que é o caso da tese de doutorado de Barbedo (2004), "Avaliação objetiva de qualidade de sinais de áudio e voz". Nela o autor aborda a dificuldade em avaliar objetivamente a qualidade de um áudio e, a partir disso, desenvolve dois métodos objetivos de avaliação, a Medida Objetiva da Qualidade de Áudio (MOQA) e Avaliação Objetiva de Sinais de Voz (AOSV), sendo que o primeiro obteve os melhores resultados e foi patenteado pelo autor.

## 1.1 Objetivo geral

O objetivo geral deste trabalho é propor e desenvolver uma métrica objetiva de avaliação da qualidade de áudio levando em consideração a sensibilidade auditiva humana conforme representada pela curva isofônica, visando obter uma avaliação mais precisa se

baseando na percepção auditiva humana.

## 1.2 Objetivos específicos

- Realizar uma revisão da literatura sobre os codificadores de áudio Vorbis e Opus, incluindo suas características e aplicações relevantes.
- Analisar os parâmetros de ajuste de cada codificador com foco na compreensão de como eles afetam a qualidade de áudio resultante.
- Localizar os codificadores Vorbis e Opus já implementados e disponíveis para uso na realização de experimentos.
- Selecionar uma base de dados com arquivos de áudios para utilizar nos experimentos.
- Avaliar o desempenho da qualidade de áudio dos codificadores Vorbis e Opus variando as configurações de parâmetros.
- Investigar as métricas de avaliação da qualidade de áudio existentes.
- Propor e desenvolver uma nova métrica objetiva de avaliação da qualidade de áudio que incorpore uma curva isofônica complementada pela tradicional relação sinal-ruído de pico (PSNR).

# 2 REVISÃO BIBLIOGRÁFICA

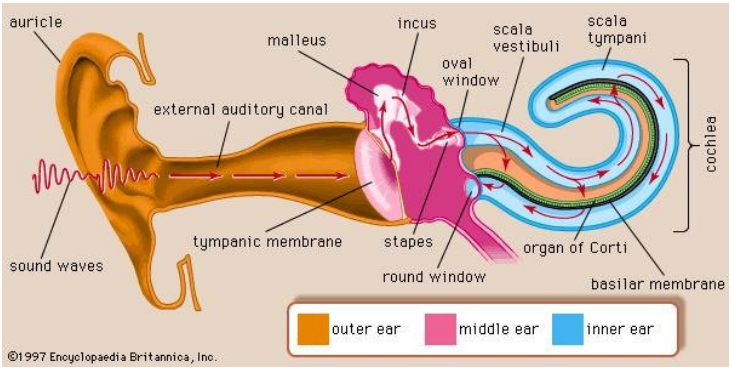
## 2.1 Sistema auditivo

O sistema auditivo humano é responsável por transformar as vibrações sonoras em impulsos nervosos que são enviados para o cérebro, o qual irá interpretar como um som (HAWKINS, 2018). Todo esse processo de tratamento do som é conhecido como audição.

A Figura 1 apresenta o sistema auditivo humano e o seu funcionamento. Observando a direção das setas vermelhas, a audição inicia quando as ondas sonoras (*sound waves*) entram no ouvido externo (*outer ear*) através do canal auditivo externo chegando na membrana timpânica, (*tympanic membrane*), mais conhecida como tímpano. O ouvido externo representa toda a parte que está em amarelo na imagem.

Continuando o processo de audição, o tímpano vibra na mesma frequência da onda sonora propagando essa vibração para os ossículos do ouvido médio (*middle ear*). O ouvido médio é toda a parte rosa da Figura 1. Esses ossículos são três pequenos ossos localizados enfileirados no ouvido médio, eles são chamados de martelo (*malleus*), bigorna (*incus*) e estribo (*stapes*). O som se propaga através desses ossos em direção ao ouvido interno (*inner ear*), a parte em azul da imagem, criando ondas nos fluidos da cóclea vibrando a membrana basilar estimulando as células sensoriais do órgão de Corti (*organ of Corti*), que, por sua vez, transmite a informação obtida para o cérebro.

Figura 1 – Sistema auditivo




Fonte: Hawkins (2018)

## 2.2 Características psicoacústicas

As características psicoacústicas da audição humana definem a percepção auditiva (SPANIAS, 2007). Essas características são exploradas pela maioria dos codificadores de áudio para obter maior compressão da informação sem afetar a sua qualidade. Estas características quando usadas por um codificador possibilitam que sejam eliminadas apenas as informações que não são detectadas pelo ouvido humano.

Um conceito necessário para avançar nesta subseção é o *sound pressure level* (SPL), que pode ser traduzido como nível de pressão sonora. O SPL é uma métrica padrão utilizada para mensurar a intensidade em decibéis (dB) de um som em relação ao nível de referência internacional. Este nível é representado pela constante $P_0$ mostrada na Equação 2.1 (SPANIAS, 2007).

$$P_0 = 2 \cdot 10^{-5} \text{ N/m}^2 \tag{2.1}$$

A seguir serão descritas as seguintes características psicoacústicas: limiar absoluto de audição, banda crítica e mascaramento simultâneo.

### Limiar absoluto de audição

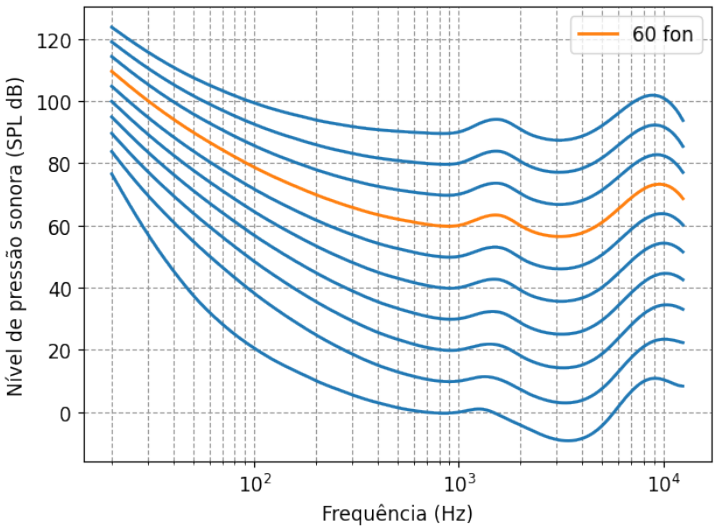
O limiar absoluto de audição define a quantidade de energia necessária para poder detectar um tom puro em um ambiente sem ruído (SPANIAS, 2007). A unidade de medida utilizada para representar esse limite é o dB SPL. Este limiar é calculado com base no nível de referência $P_0$, portanto o limiar em silêncio mede aproximadamente 0 dB SPL e o limiar de dor é igual ou superior 140 dB SPL.

A dependência entre o limiar absoluto de audição e a frequência foi descoberta por Fletcher, em 1940, após estudos sobre a percepção auditiva (SPANIAS, 2007). A Equação 2.2 apresenta uma fórmula que se aproxima do limiar de audição de um jovem que calcula o limiar de sensibilidade da audição.

$$T_q(f) = 3{,}64(f/1000)^{-0{,}8} - 6{,}5e^{-0{,}6(f/1000-3{,}3)^2} + 10^{-3}(f/1000)^4 \quad \text{(dB SPL)} \tag{2.2}$$

Em 1930, Fletcher e Munson mediram pela primeira vez as curvas isofônicas (SUZUKI; TAKESHIMA; KURAKATA, 2024). Essas curvas possibilitam entender esse comportamento em relação à percepção humana, sendo de grande importância para os estudos em acústicas. Desde então, as curvas isofônicas passaram por várias revisões ao longo dos anos. No caso deste trabalho foi considerada a revisão de 2003 representada na Figura 4, registrado como ISO226:2003.

Figura 2 – Curvas isofônicas




Fonte: Elaborada pelo autor baseada na ISO 226:2003.

### Bandas críticas

Como visto no início desta seção (seção 2.1), é na cóclea que acontece a conversão da frequência da onda sonora em impulsos nervosos. Segundo Spanias (2007), a cóclea pode ser considerada um banco de filtros passa banda, por isso foi criada a função largura de banda crítica, Equação 2.3, para quantificar as bandas passantes da cóclea.

$$BW_c(f) = 25 + 75[1 + 1{,}4(f/1000)^2]^{0{,}69} \quad \text{(Hz)} \tag{2.3}$$

### Mascaramento simultâneo

De acordo com Spanias (2007), o mascaramento simultâneo é um processo em que não é possível ouvir um som devido a inserção de um outro som simultaneamente. O mascaramento ocorre no sistema auditivo quando dois ou mais sons são captados, porém um deles possui características que anulam o outro som, por exemplo, quando dois sons com frequências muito próximas chegam ao ouvido, aquele com maior amplitude vai se sobrepor ao de menor amplitude, dando a sensação de que existe apenas um som.

## 2.3 Sinal de áudio

Um som ocorre a partir da vibração de algum objeto gerando ondas sonoras, por exemplo, a vibração das cordas vocais no momento da fala ou a vibração de algum instrumento musical. Essas ondas sonoras são conhecidas também como um sinal de áudio. O sinal de áudio pode ser gravado com o auxílio de um microfone, no qual as

ondas sonoras são convertidas em sinais elétricos para serem armazenadas ou transmitidas (BRITANNICA, 2019).

## 2.4 Codificadores de áudio

Os codificadores de áudio, também chamados de compressores de áudio, surgiram com a necessidade de reduzir ao máximo a quantidade de bits na representação de um áudio sem prejudicar a sua qualidade facilitando a sua transmissão ou o seu armazenamento (SPANIAS, 2007). Considera-se um bom codificador, aquele em que não há diferença audível perceptível entre o áudio original e o codificado.

Segundo Spanias (2007), uma forma de classificar os codificadores de áudio é de acordo com a sua técnica de tratamento do áudio. Essas técnicas de codificação podem ser do tipo que usa predição linear, transformada, sub-banda ou senoidal.

Outra forma de classificar os codecs é se existem perdas na codificação ou não (SPANIAS, 2007). Esse é o modo de classificação mais utilizado. Os codificadores sem perdas são conhecidos pelo termo em inglês *lossless*, enquanto os com perdas são chamados de codec *lossy*.

Codificadores *lossless* não possuem perdas de compressão mantendo a qualidade do áudio original. Apesar de não alterarem a qualidade do áudio, compressores deste tipo tem baixa taxa de compactação, ou seja, reduzem pouco o tamanho do áudio, pois eles apenas suprimem seus sinais redundantes, de modo que haverá maior compressão nos áudios que possuírem mais redundâncias.

No caso dos compressores *lossy* ocorrem perdas causadas pela codificação, porém estas perdas devem alterar o mínimo possível da qualidade do áudio. Os melhores codecs são aqueles que mais exploram as características psicoacústicas, sendo assim, as perdas ocorridas no processo são de informações que o ouvido humano não consegue detectar.

### 2.4.1 Codificadores selecionados

Com a finalidade de otimizar os estudos, foram selecionados apenas codificadores de código aberto e disponíveis gratuitamente para a instalação. Os codificadores escolhidos foram o Vorbis e o Opus. Ambos foram produzidos pela Xiph.Org Foundation, uma fundação sem fins lucrativos focado em desenvolver projetos de código aberto e gratuito.

#### Vorbis

O Vorbis é classificado como um codec do tipo com perdas de uso geral. Este codificador é projetado para trabalhar tanto com altas quanto com baixas taxas de amostragem, ou seja, consegue tratar a voz e a música de forma otimizada (XIPH.ORG, 2015). Sua

codificação utiliza características psicoacústicas que, por consequência, aumentam o nível de complexidade do seu algoritmo, enquanto sua decodificação possui baixa complexidade.

O Vorbis trabalha com *variable bit rate* (VBR), que significa que a taxa de bits é variável se ajustando conforme a necessidade, e seus pacotes também são ajustáveis, pois não possuem um tamanho definido (XIPH.ORG, 2015). Este codec possui dois tipos, o Vorbis I, que usa a *Modified Discrete Cosine Transform* (MDCT) para preparar o áudio para a codificação, e o Vorbis II, que implementa um banco de filtros *wavelet* híbrido substituindo a MDCT (XIPH.ORG, 2015). Este projeto trabalhará apenas com o Vorbis I.

Este codificador foi projetado para ser utilizado em conjunto com algumas ferramentas de transmissão, isto significa que tem o objetivo de preparar os áudios para serem transmitidos. O serviço de *streaming* de música do Spotify, por exemplo, utiliza o Vorbis para preparar as músicas antes de transmitir (SPOTIFY, 2024). Outro exemplo, são alguns jogos de *videogame* que também utilizam este compressor (XIPH.ORG, 2018).

Um parâmetro de codificação do Vorbis é a qualidade que varia entre $-2$ e 10. A qualidade é diretamente relacionada ao parâmetro taxa de bits variável, VBR, como mostra a Tabela 1, na qual pode se observar que conforme aumenta o valor da qualidade a VBR também aumenta.

Tabela 1 – Relação entre a qualidade e a VBR no codificador Vorbis

| Qualidade | VBR nominal [kbit/s] | Faixa de VBR [kbit/s] |
|---|---|---|
| $-2$ | 32 | ∼32 – ∼64 |
| $-1$ | 48 | ∼48 – ∼64 |
| 0 | 64 | ∼64 – ∼80 |
| 1 | 80 | ∼80 – ∼96 |
| 2 | 96 | ∼96 – ∼112 |
| 3 | 112 | ∼112 – ∼128 |
| 4 | 128 | ∼128 – ∼160 |
| 5 | 160 | ∼160 – ∼192 |
| 6 | 192 | ∼192 – ∼224 |
| 7 | 224 | ∼224 – ∼256 |
| 8 | 256 | ∼256 – ∼320 |
| 9 | 320 | ∼320 – ∼500 |
| 10 | 500 | ∼500 – ∼1000 |

Fonte: Dados extraídos de Hydrogenaudio (2018)

### Opus

O Opus é um codificador de áudio com perdas definido como um codec interativo, que pode ser utilizado em aplicações *Voice over Internet Protocol* (VoIP), videoconferências e bate-papo em jogos (XIPH.ORG, 2017). Este compressor é baseado em outros dois existentes: o SILK, do Skype, e o CELT, da Xiph.Org Foundation.

O SILK é um codificador baseado em *Linear Prediction* (LP), que o torna mais eficiente no tratamento de baixas frequências, ou seja, é um ótimo codificador de voz (XIPH.ORG, 2017). Assim como o Vorbis, este também trabalha com taxas variáveis (VBR).

Ao contrário do SILK, o CELT é um codec que utiliza a MDCT e, este é um codificador muito mais eficiente no tratamento de altas frequências, por exemplo: uma música (XIPH.ORG, 2017). O CELT trabalha com *constant bit rate* (CBR), ou seja, este codec opera com taxa de bits constante em toda a sua codificação.

O Opus possui uma camada que utiliza uma versão modificada do SILK e outra que se baseia no CELT, ambas as camadas conseguem operar tanto com VBR quanto com CBR (XIPH.ORG, 2017). Por este motivo, o Opus consegue obter uma boa qualidade de codificação de áudios de voz e de música.

## 2.5 Métodos de avaliação da qualidade do áudio

Avaliação da qualidade de um áudio é uma tarefa bem complexa, porém é muito importante para validar o funcionamento de um codificador de áudio. Essa avaliação pode ser feita de duas formas: objetiva ou subjetiva.

As medidas objetivas, como a *Signal-to-Noise Ratio* (SNR) e a *total harmonic distortion* (THD), comumente utilizadas na análise de sinais em geral, não obtém bons resultados quando aplicados na avaliação de um áudio. Os métodos objetivos que funcionam para avaliar a qualidade de um áudio são aqueles que melhor reproduzem a percepção auditiva humana.

A ITU-R define na recomendação BS.1387-1 (ITU, 2001), o modelo objetivo PEAQ, esse possui duas versões, uma baseada na *Fast Fourier transform* (FFT) e a outra baseada na FFT em conjunto com banco de filtros. Essa métrica tem como saída alguns dados referentes a qualidade do áudio, em destaque para as saídas *Distortion Index* (DI) e *Objective Difference Grade* (ODG), sendo que ambas são diretamente relacionadas. A saída DI é o índice de distorção e a saída ODG pode ser calculada a partir de DI e classifica a qualidade do áudio em uma escala que vai de $-4$ a 0, na qual o 0 significa que o áudio está com ótima qualidade, enquanto $-4$ significa um áudio de péssima qualidade.

Outros métodos de avaliação foram propostos no decorrer dos anos, como por exemplo Barbedo (2004) propôs em sua tese de doutorado dois métodos objetivos de avaliação, a MOQA e AOSV, usando como base o modelo PEAQ. O primeiro método foi desenvolvido para avaliar a qualidade de música e voz dentro da faixa de frequências de 20 Hz a 20 kHz, este método foi patenteado pelo autor. Enquanto o segundo método foi implementado para analisar um sinal de voz na mesma faixa de frequência da telefonia,

entre 300 Hz a 3,4 kHz, e, diferentemente do primeiro método, este não foi patenteado pelo autor.

Segundo Palomar et al. (2008), no caso da avaliação da qualidade de um áudio, os métodos subjetivos são os mais confiáveis, porém possuem um alto custo de realização e precisam de muito tempo para a sua execução.

A MOS é o principal método de avaliação subjetiva, ela consiste em realizar testes de audição subjetivos. Para sua realização é necessário organizar um grupo considerável de pessoas treinadas para ouvir e avaliar a qualidade do áudio. Em um breve resumo do funcionamento dessa métrica, várias pessoas devem receber diversos áudios sem o conhecimento dos padrões utilizados na sua geração, por exemplo qual o codec usado e com quais parâmetros o áudio foi gerado, e a partir disso ouvir e classificar a qualidade percebida. Os resultados obtidos no final da avaliação MOS são utilizados para formar um valor médio na escala de 1 a 5 para quantificar a qualidade do áudio, sendo que 5 representa a maior qualidade.

### 2.5.1 PSNR e MSE

Segundo Chandler e Hemami (2007), as métricas *Peak Signal-to-Noise Ratio* (PSNR) e *Mean-squared Error* (MSE) são métricas objetivas geralmente utilizadas para analisar a qualidade de imagens e vídeos após realizada a sua compressão em relação a sua versão original, sendo muito utilizadas para avaliar a qualidade dos codificadores de imagem. Essas medidas objetivas puras não são consideradas muito adequadas na avaliação de áudio, pois segundo Palomar et al. (2008) os métodos de avaliação subjetiva são mais confiáveis.

A PSNR é calculada pela Equação 2.4 e depende matematicamente da MSE. Na fórmula, $n$ representa o número de bits do sinal, sendo assim, a expressão $(2^n - 1)^2$ corresponde ao valor máximo do sinal (CHANDLER; HEMAMI, 2007).

$$\text{PSNR} = 10 \log_{10} \left( \frac{(2^n - 1)^2}{\text{MSE}} \right) \quad \text{(dB)} \tag{2.4}$$

A MSE é obtida através da Equação 2.5, na qual $N$ é o número total de amostras do sinal e $E_i$ significa o valor do erro de uma amostra do sinal, representada pela Equação 2.6, sendo $y_i$ o valor observado de uma amostra e $p_i$ o valor esperado dessa mesma amostra (CHANDLER; HEMAMI, 2007).

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} E_i^2 \tag{2.5}$$

$$E_i = y_i - p_i \tag{2.6}$$

## 2.6 Base de áudios

Para esse projeto foi selecionada uma base de arquivos que possui 40 amostras de áudio, de Kamedo (2014), dos mais variados tipos, nas quais 35 são amostras de música de variados gêneros e 5 das amostras são áudios de fala. Dentre esses 40 áudios foram selecionados os 5 áudios de fala que contém bastante voz para a realização dos testes com a métrica proposta. Todos os áudios selecionados desta base possuem a taxa de bit de 1411,2 kbps e seu tamanho está apresentado na Tabela 2.

Essa base foi disponibilizada e utilizada por Kamedo (2014) que realizou testes subjetivos de audição com estes áudios. Esses testes foram feitos às cegas, ou seja, dividiram as amostras em dois grupos e distribuíram aleatoriamente entre os ouvintes sem que saibam qual o codec foi utilizado no processamento de cada áudio. Os codificadores avaliados neste experimento foram o Opus, o *Advanced Audio Coding* (AAC) e o Ogg Vorbis com taxa de bits igual a 96 kbps, e o MP3 a um taxa de 128 kbps, além do codificador *Freeware Advanced Audio Coder* (FAAC) que já era conhecido por ser de qualidade inferior como ponto de referência para padronizar a classificação, no caso desde último codificador não foi definida uma taxa de bits específica.

Tabela 2 – Tamanho dos arquivos de voz descontando o tamanho do cabeçalho

| Nome do arquivo | Tamanho do arquivo [kB] |
|---|---|
| 4-Sound-English-male | 1592,56 |
| 12-German-male-speech | 1383,00 |
| 15-Good-evening | 1511,00 |
| 24-Greensleeves-Korean-male-speech | 1538,21 |
| 25-This-is-the-end | 1620,12 |

Fonte: Elaborada pelo autor.

## 2.7 Filtro de Resposta ao Impulso Finito (FIR)

*Finite Impulse Response* (FIR) é um tipo de filtro digital, descrito com uma resposta ao impulso finita, são estáveis e não recursivos pois não possuem realimentação. Segundo Shenoi (2010), o filtro FIR possui quatro tipos de filtros com diferentes características. Os tipos I e II possuem coeficientes simétricos, sendo que o tipo I é de ordem par e o tipo II tem ordem ímpar. Os tipos III e IV tem coeficientes assimétricos, onde o tipo III possui ordem par e tipo IV ordem ímpar (SHENOI, 2010).

Existem algumas janelas criadas por pesquisadores que são usadas no projeto de um filtro, como, por exemplo, a janela de Kaiser. A utilização da janela de Kaiser permite o controle sobre o tamanho da largura de banda do filtro projetado. Além disso, essa janela pode ser usada apenas no projeto de filtros do tipo I (SHENOI, 2010).

# 3 DESENVOLVIMENTO

O trabalho propõe além do estudo dos codificadores, avaliar algumas métricas objetivas e subjetivas de avaliação da qualidade de áudio. Foram selecionados os codificadores Vorbis e Opus para esse estudo. Utilizou-se a métrica subjetiva MOS e as objetivas PSNR, MSE, PEAQ e também a nova métrica proposta FonPSNR.

Na construção da FonPSNR foi utilizado um filtro FIR passa banda desenvolvido a partir da curva isofônica de 60 $fon$ com o intuito de ponderar o áudio, e, em seguida, passar essa informação pela PSNR e obter um valor para mensurar a qualidade desse áudio.

Para colocar em prática a criação da nova métrica definiu-se a utilização da linguagem de programação Python na versão 3.8.10, pois além de ser muito utilizada na área tecnológica, permite facilmente executar programas simulando uma linha de comando do Linux. Os códigos referentes às próximas seções estão disponíveis no Apêndice C.

## 3.1 Codificadores de áudio

Os codificadores escolhidos para este projeto, o Vorbis e o Opus, proporcionam uma qualidade de áudio elevada, embora operem com perdas. Eles estão disponíveis em diversos sistemas operacionais, como Linux e Windows. A seleção desses codificadores foi fundamentada em sua disponibilidade e excelência em qualidade, uma vez que a intenção não era implementá-los, mas sim utilizá-los no estudo de métricas e avaliação da qualidade de áudio.

Na manipulação dos dados utilizou-se pacotes de programas executáveis para sistema Linux dos codificadores predefinidos. O pacote Opus utilizado foi o `opus-tools` na versão 0.1.10-1 instalado a partir do terminal do Linux com o comando `sudo apt install opus-tools`. O pacote Vorbis usado foi o `vorbis-tools` na versão 1.4.0-11 instalado também via terminal a partir do comando `sudo apt install vorbis-tools`.

Com a finalidade de realizar várias operações com os codecs, criou-se um código em Python que recebe um diretório com uma base de áudios no formato WAVE, esses foram codificados e decodificados recursivamente utilizando os codificadores instalados variando um dos parâmetros de codificação, a taxa de bits. Esse código está integrado ao projeto que deve realizar a medição da qualidade dos áudios tratados utilizando as métricas objetivas.

## 3.2 Aplicação de métricas objetivas

Atualmente são poucas as métricas objetivas de avaliação de áudio que obtém bons resultados, pois essa avaliação depende de análises complexas das características psicoacústicas em relação a percepção humana do áudio. Pensando nisso, foi realizado um estudo para selecionar quais medidas poderiam ser utilizadas para realizar essa qualificação de forma objetiva.

Como dito na seção 2.5, a medida mais utilizada e que os estudiosos na área entendem como sendo a melhor é a métrica subjetiva MOS. As métricas subjetivas possuem alto custo e muito tempo de execução. Sabendo disso, foram utilizados os resultados da aplicação da MOS feita por Kamedo (2014) com o intuito de encontrar uma medida objetiva que obtenha resultados aproximados.

Com a finalidade de encontrar uma medida objetiva que possa ser usada como base na verificação dos resultados obtidos pela métrica proposta, utilizou-se os mesmos dados para aplicar as métricas PSNR, MSE e PEAQ com os seus dois retornos, ODG e DI. Devido a existirem poucos valores de MOS para essa comparação, optou-se por utilizar a PEAQ ODG por revelar resultados um pouco mais aproximados dos valores da MOS, mas principalmente por ser uma métrica objetiva de avaliação de áudio reconhecida e definida pela ITU-R (ITU, 2001).

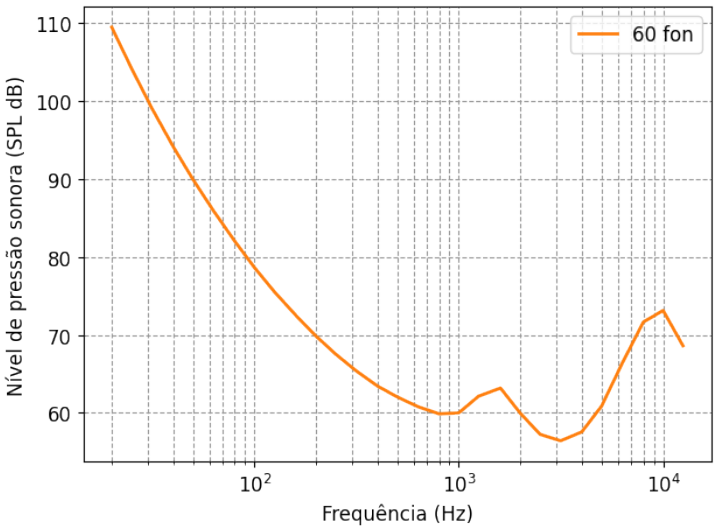
Foi desenvolvido o código para manipular cada áudio, no qual cada amostra será codificada e decodificada, para em seguida ser analisada utilizando as métricas e as respostas dessa análise são registradas em arquivos Comma-Separated Values (CSV). As métricas PSNR e MSE foram obtidas por meio de bibliotecas Python disponíveis, a PSNR é da biblioteca *skimage* e a MSE pertence a *sklearn*. A métrica PEAQ foi obtida através do código em C produzido por Gottardi (2015), com este código é gerado um arquivo executável que pode ser executado utilizando o terminal Linux.

## 3.3 PSNR ponderada com a curva de $60\ fon$ (FonPSNR)

Pensando em uma forma de ponderar os sinais que se encontram no espectro de frequência audível pelo ouvido humano, projetou-se um filtro FIR passa banda utilizando uma curva isofônica, a ideia é que esse filtro altere o sinal amplificando as frequências que são mais perceptíveis pelo ouvido humano. A curva de 60 $fon$ foi escolhida, Figura 3, por ser uma curva que compreende as frequências que o ouvido humano consegue interpretar, sendo bastante usada em testes de audiometria para identificar a presença de algum problema auditivo.

Iniciou-se o desenvolvimento do projeto do filtro, utilizando como base o código produzido por Mathias (2019) que gera várias curvas isofônicas com o propósito de utilizá-lo

Figura 3 – Curva isofônica de 60 $fon$



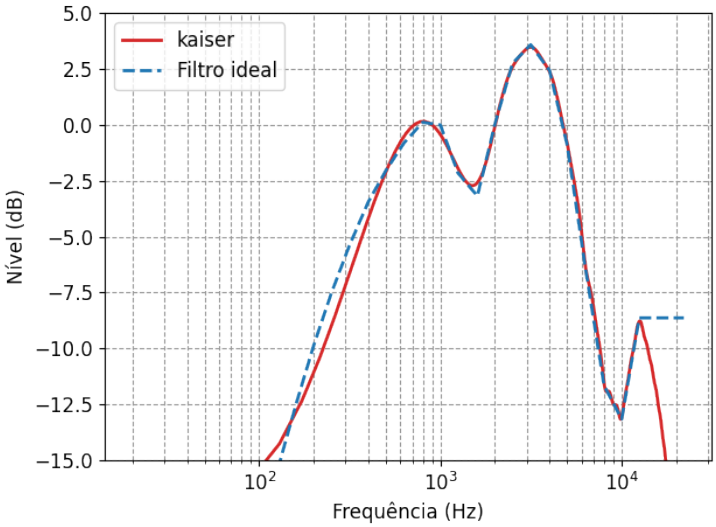

Fonte: Elaborada pelo autor.

na composição da função que deve produzir somente uma curva isofônica por vez, conforme ilustrado na Figura 3 na qual demonstra a curva de 60 $fon$ feita por esta função. Essa função retorna a curva em forma de dois vetores, sendo um vetor contendo as frequências e o outro as amplitudes da curva.

Implementou-se a função que aplica a curva isofônica de 60 $fon$ para projetar um filtro FIR passa banda do tipo I de ordem 100 utilizando a janela de Kaiser que deve possuir a resposta em frequência simétrica. Foi escolhida a janela de Kaiser para esse projeto pois possui transições mais suaves entre as bandas, obtendo um filtro mais próximo do filtro ideal. Esse filtro foi construído utilizando a função *signal.firwin2* da biblioteca *scipy*. A Figura 4 exibe o resultado da banda passante do filtro comparado ao filtro ideal pensado para este projeto. Os coeficientes obtidos no projeto deste filtro se encontram no Apêndice A.

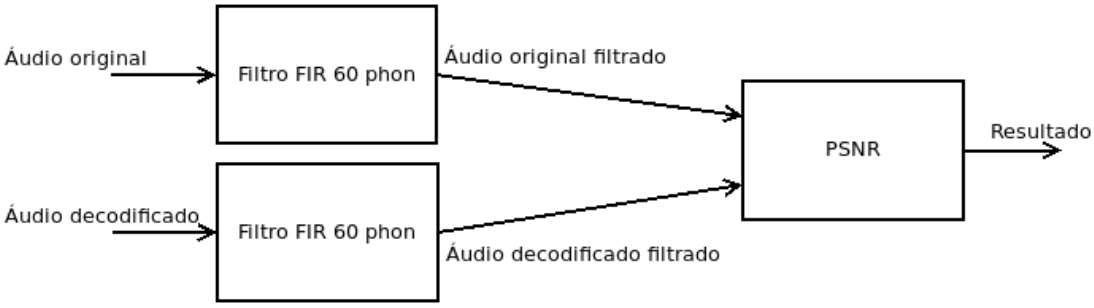
Finalizada a elaboração do filtro, foi desenvolvida a função que recebe dois áudios sendo um o original e o outro o manipulado por um codec, como ilustrado na Figura 5, para que ambos passem pelo filtro e posteriormente são aplicados na PSNR com o objetivo de compará-los.

Figura 4 – Resposta em frequência do filtro




Fonte: Elaborada pelo autor.

Figura 5 – Diagrama de blocos do código da FonPSNR




Fonte: Elaborada pelo autor.

# 4 RESULTADOS E ANÁLISE

O funcionamento do programa produzido para este projeto, inicia-se em utilizar uma base de áudio para manipulá-los utilizando os codecs Opus e Vorbis recursivamente variando a taxa de bits de cada áudio com o intuito de avaliar os arquivos manipulados utilizando as métricas PEAQ ODG e a FonPSNR. As taxas de bits foram variadas entre 45 e 500 *kpbs* seguindo as limitações dos codecs, para determinar isso foi utilizado como base as taxas de bits obtidas a partir da variação do parâmetro de qualidade do Vorbis. Os resultados obtidos pela análise são registrados em tabelas dentro de arquivos CSV.

## 4.1 Comparação das métricas de avaliação estudadas

O primeiro experimento consistiu em encontrar uma métrica objetiva que pudesse suprir a falta da MOS. A fim de simplificar a análise dos resultados, separou-se apenas os áudios de voz para utilizar no experimento, desse modo os dados apresentados neste projeto são todos relacionados aos áudios de voz retirados da base de dados.

A Tabela 3 apresenta a comparação entre a MOS e os dados obtidos pelas métricas objetivas. Para montar essa tabela, foram extraídos os dados de MOS e os áudios codificados fornecidos pelo site de Kamedo (2014). Nessa etapa, observou-se ser necessário decodificar os arquivos antes de analisá-los, em vista disso, utilizou-se os decodificadores Vorbis e Opus descritos na seção 3.1. Esses decodificadores não possuem a mesma versão dos codecs utilizados pelo autor.

Tabela 3 – Comparação entre a MOS obtida por Kamedo (2014) para taxas predeterminadas em relação as métricas objetivas calculadas

| Nome do arquivo | Codec | MOS | PEAQ ODG | PEAQ DI | MSE | PSNR |
|---|---|---|---|---|---|---|
| 4-Sound-English-male | Opus | 4,97 | -0,362 | 1,839 | 0,000205 | 42,898 |
| | Vorbis | 3,77 | -0,533 | 1,527 | 0,00018 | 43,530 |
| 12-German-male-speech | Opus | 4,82 | -0,286 | 1,991 | 0,000084 | 46,767 |
| | Vorbis | 3,59 | -0,360 | 1,832 | 0,00008 | 46,996 |
| 15-Good-evening | Opus | 4,31 | -1,574 | 0,299 | 0,000513 | 38,923 |
| | Vorbis | 4,24 | -0,720 | 1,249 | 0,00074 | 37,321 |
| 24-Greensleeves-Korean-male-speech | Opus | 5,00 | -0,346 | 1,865 | 0,000066 | 47,841 |
| | Vorbis | 4,23 | -0,449 | 1,664 | 0,00004 | 50,180 |
| 25-This-is-the-end | Opus | 4,89 | -0,862 | 1,069 | 0,000271 | 41,687 |
| | Vorbis | 4,20 | -0,733 | 1,232 | 0,00024 | 42,304 |

Fonte: Elaborada pelo autor.

A partir dos dados da Tabela 3, observa-se que esses valores não foram suficientes para concluir qual métrica poderia substituir a MOS, pois cada uma possui sua própria

Tabela 4 – Escalas das métricas

| Métrica | Mínimo | Máximo |
|---|---|---|
| MOS | 1 | 5 |
| PEAQ ODG | −4 | 0 |

Fonte: Elaborada pelo autor.

escala, na PSNR quanto maior o resultado melhor é o sinal, e as escalas da MOS e PEAQ ODG se encontram na Tabela 4.

Outro ponto importante na análise dessas informações são as variações em que os dados divergem entre si, como, por exemplo, as informações sobre os arquivos "12-German-male-speech" e "15-Good-evening". Nos resultados obtidos para o arquivo "12-German-male-speech" pode-se verificar que existe uma diferença de 1,23 pontos entre os valores de MOS de cada codificador, sendo que na PEAQ ODG a diferença é de 0,074, ou seja, há uma variação grande entre esses valores, principalmente, considerando que ambas as métricas possuem escalas com o mesmo tamanho, a primeira variando entre 1 e 5 e a outra de −4 a 0. Além disso, observa-se também que os resultados obtidos pela PSNR contrariam a MOS apontando o Vorbis como o melhor codec. Enquanto no caso do áudio "15-Good-evening", conclui-se que o Opus obteve melhor resultado nas métricas MOS e PSNR, porém o resultado obtido pela PEAQ ODG indica o Vorbis como o melhor codificador.

Em vista disso, o critério usado para definir a medida objetiva de qualidade para testar o segundo experimento foi o fato da PEAQ ser uma métrica objetiva reconhecida e documentada pela ITU-R.

## 4.2 Análise do resultado da métrica FonPSNR

A Tabela 5 mostra os resultados da avaliação da qualidade do áudio "4-Sound-English-male" variando a taxa de bits nas codificações com Vorbis e Opus. Verificando a PEAQ ODG do caso do codificador Vorbis da tabela, pode-se ver que nas menores taxas de bits qualquer variação entre elas gera uma diferença bastante significativa no resultado da avaliação, em contrapartida, nas maiores taxas essa diferença se mostra mais sutil, podendo considerar que o ouvido humano não seria capaz de perceber a alteração entre elas.

Existem algumas inconsistências nos valores da PEAQ, por exmplo, no caso do compressor Vorbis nas taxas de bits de 256, 320 e 500 *kpbs* seria esperado que esses valores fossem crescendo junto com a taxa, porém há uma queda no valor da qualidade correspondente a taxa de 320 *kpbs*. Outro caso desse ocorre nas taxas de 224 a 320 *kpbs* do codificador Opus.

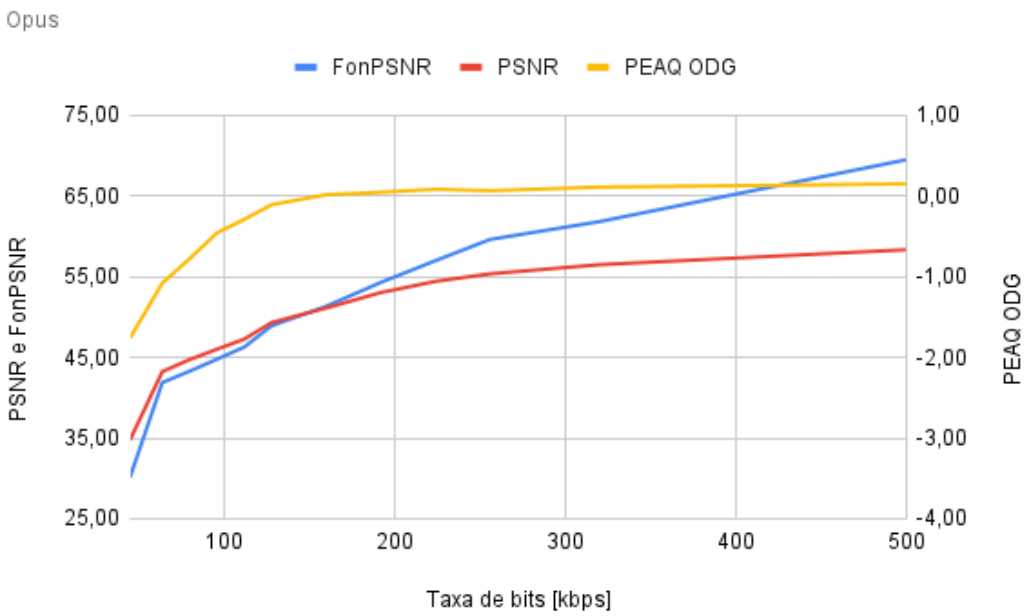
Tabela 5 – Resultado da avaliação da qualidade do áudio “4-Sound-English-male” variando a taxa de bits

| | Opus | | | Vorbis | | |
|---|---|---|---|---|---|---|
| Taxa de bits [kbps] | FonPSNR | PSNR | PEAQ ODG | FonPSNR | PSNR | PEAQ ODG |
| 45 | 30,11 | 34,75 | -1,76 | 34,35 | 37,13 | -3,53 |
| 64 | 41,88 | 43,24 | -1,08 | 36,65 | 40,03 | -1,63 |
| 80 | 43,29 | 44,73 | -0,78 | 38,69 | 42,03 | -0,86 |
| 96 | 44,77 | 46,01 | -0,46 | 39,96 | 43,16 | -0,58 |
| 112 | 46,28 | 47,26 | -0,29 | 40,57 | 44,02 | -0,42 |
| 128 | 48,91 | 49,31 | -0,11 | 40,82 | 44,58 | -0,33 |
| 160 | 51,33 | 51,11 | 0,01 | 42,09 | 46,06 | -0,29 |
| 192 | 54,28 | 53,03 | 0,05 | 45,00 | 48,83 | -0,19 |
| 224 | 56,98 | 54,41 | 0,08 | 47,57 | 51,09 | -0,01 |
| 256 | 59,61 | 55,35 | 0,07 | 51,08 | 54,17 | 0,08 |
| 320 | 61,81 | 56,48 | 0,11 | 55,99 | 57,96 | -0,12 |
| 500 | 69,49 | 58,34 | 0,15 | 62,88 | 62,96 | 0,05 |

Fonte: Elaborada pelo autor.

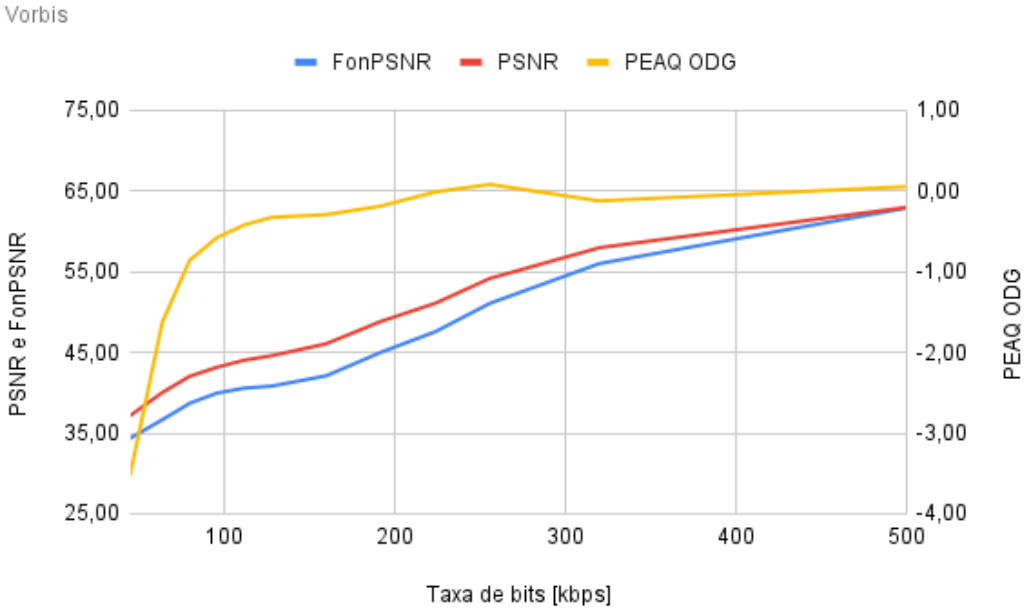
Baseando-se nas informações da Tabela 5, foram montadas as Figuras 6 e 7 para representar as variações entre as métricas de qualidade com a taxa de bits. A partir da análise dessas imagens em conjunto com a Tabela 5, é possível perceber que a PEAQ possui menor variação quando a taxa de bits atinge aproximadamente 128 *kbps*, isso pode representar que em taxas acima desse valor o ouvido humano não deve notar grandes diferenças na qualidade dos áudios.

Figura 6 – Variação das métricas de qualidade em relação a taxa de bits: áudio “4-Sound-English-male” codificado com Opus




Fonte: Elaborada pelo autor.

Figura 7 – Variação das métricas de qualidade em relação a taxa de bits: áudio "4-Sound-English-male" codificado com Vorbis




Fonte: Elaborada pelo autor.

A Tabela 6 apresenta os valores calculados das correlações entre a taxa de bits e as métricas objetivas. A partir dessa tabela pode-se concluir que existe uma boa correlação entre esses dados quando variada a taxa de bits e a FonPSNR, principalmente quando comparada a correlação da PSNR pura, podendo concluir que realmente ocorreu a melhora na correlação. Com base nessa tabela, a correlação entre a PEAQ ODG e a taxa de bits não obteve um resultado satisfatório.

Tabela 6 – Comparação das correlações entre a taxa de bits e as métricas objetivas

| Nome do arquivo | Codec | FonPSNR | PEAQ ODG | PSNR |
|---|---|---|---|---|
| 4-Sound-English-male | Opus | 0,927 | 0,684 | 0,844 |
| | Vorbis | 0,988 | 0,591 | 0,977 |

Fonte: Elaborada pelo autor.

Outra análise feita foi a correlação entre a PEAQ ODG e a FonPSNR, pois essa PEAQ é uma medida objetiva reconhecida. Essa correlação está apresentada na Tabela 7, na qual pode ser observar que o valor das correlações é relativamente ruim, não sendo suficiente para concluir se a nova medida funciona.

Levando em consideração as análises anteriores sobre a relação entre a taxa de bits e as medidas objetivas, a PEAQ ODG mostrou que a partir de 128 *kbps* a variação da qualidade é imperceptível, com essa informação construiu-se a Tabela 8 onde foram usados apenas a faixa de 45 a 128 *kbps* para calcular as correlações. Nesse caso, obteve-se um

Tabela 7 – Comparação das correlações entre a PEAQ ODG e a FonPSNR

| Nome do arquivo | Codec | FonPSNR |
|---|---|---|
| 4-Sound-English-male | Opus | 0,887 |
| | Vorbis | 0,643 |

Fonte: Elaborada pelo autor.

valor de correlação bastante significativo em ambos os codificadores.

Tabela 8 – Comparação das correlações entre a PEAQ ODG e a FonPSNR dos arquivos com taxa de bits máxima de 128 *kbps*

| Nome do arquivo | Codec | FonPSNR |
|---|---|---|
| 4-Sound-English-male | Opus | 0,967 |
| | Vorbis | 0,967 |

Fonte: Elaborada pelo autor.

No entanto, considerando que houve algumas inconsistências encontradas nos resultados da PEAQ, para comprovar a real eficácia da nova métrica seria necessário obter os valores de MOS correspondentes aos áudios codificados com uma variedade de taxas de bits, por exemplo, assim como feito nesse experimento, pois não é possível afirmar que o programa da PEAQ obtida para esse trabalho funciona como está definida na recomendação da ITU-R.

Esse experimento foi realizado também em mais quatro arquivos de áudio, que obtiveram resultados similares, esses podem ser encontrados no Apêndices B.

# 5 CONCLUSÃO

O objetivo deste trabalho foi propor e desenvolver uma nova métrica de avaliação da qualidade de áudio de forma objetiva, a FonPSNR, usando a curva isofônica para ponderar o sinal levando em conta a sensibilidade da audição humana.

Neste projeto foi realizado um estudo sobre os codificadores Opus e Vorbis, no qual foi visto sobre a existência de parâmetros de codificação. Ambos os codec possuem o parâmetro de codificação taxa de bits, portanto esse foi definido para ser utilizado neste projeto. Com o intuito de usar nesse experimento, foram encontrados os programas dos codificadores Opus e Vorbis compatíveis com o sistema operacional Linux.

Durante a etapa de análise dos resultados obtidos, foi constatado que os resultados das métricas objetivas existentes não foram suficientes para gerar uma conclusão definitiva sobre o funcionamento da métrica FonPSNR. Realizar a avaliação da qualidade de um áudio é bastante complexa, para ter mais garantia nos resultados deste trabalho, alguns dos áudios codificados foram ouvidos em diferentes meios para checagem da qualidade, porém não foi possível notar uma grande variação neles.

Para obter uma análise mais conclusiva para esse projeto seria necessário utilizar uma base de dados de áudios com uma avaliação MOS associada, na qual os dados fossem codificados variando os parâmetros relacionados a qualidade, como, por exemplo, a taxa de bits, garantindo que a mesma versão do codec fosse usado em todo o processo. Nesse caso os testes não precisariam ser feitos para mais de um compressor, visto que o objetivo deve ser provar o funcionamento da FonPSNR.

A aplicação do método MOS neste projeto foi descartada, devido ao seu alto custo e tempo de execução. Portanto, o foco foi procurar esse cenário pronto para ser aplicado nos experimentos de validação da FonPSNR, porém, até a finalização da fase de testes, não foram encontrados cenários com esse formato.

Outro ponto importante visto nesse trabalho, foi a visualização da relação entre a PEAQ e a taxa de bits nos maiores valores de taxa que a partir de 128 *kbps*, a variação da qualidade se tornou praticamente constante, sendo assim, as variações nas maiores taxas são consideradas imperceptíveis pelo ouvido humano. Vale ressaltar que o valor de taxa de bits 128 *kbps* como limite máximo para a qualidade se tornar constante, foi observada nos resultados mas não foi comprovada neste trabalho, é de grande importância realizar um estudo mais aprofundado nesse tema para definir qual o valor de taxa deve ser considerado como esse valor máximo, no qual ainda é possível notar alguma diferença entre os áudios.

Independentemente das dificuldades encontradas, a partir dos resultados obtidos

pode-se concluir que a métrica FonPSNR se revelou promissora quando analisada a correlação entre a medida e a taxa de bits. Ao comparar essa informação com a correlação da PSNR pura com a taxa de bits, nota-se que a FonPSNR realmente obteve uma melhora no resultado. Todavia, quando analisada a medida proposta e a PEAQ, obteve-se uma baixa correlação entre os dados, tornando-se inconclusiva a análise do funcionamento da métrica FonPSNR.

## 5.1 Trabalhos futuros

Tendo em vista as conclusões obtidas nesse trabalho, fica como trabalho futuro encontrar uma base de áudios com arquivos codificados em diferentes taxas de bit com avaliação MOS feita com apenas um tipo de codificador para a validação da métrica FonPSNR.

Outro trabalho futuro, seria produzir uma base de áudios variando os parâmetros relacionados a qualidade e aplicar o método de avaliação MOS, de modo que possa ser utilizada na validação da métrica FonPSNR, comparando, também, com a PEAQ.

Uma terceira sugestão, seria avaliar a possibilidade de usar outras curvas isofônicas como ponderação para o filtro visando encontrar a curva mais adequada para a métrica.

Por fim, encontrar o limite máximo da taxa de bits em que não seja possível diferenciar a qualidade do áudio para identificar em qual momento os resultados da avaliação da métrica deveriam ser mais constantes.

# REFERÊNCIAS


BARBEDO, J. G. A. *Avaliação objetiva de qualidade de sinais de áudio e voz*. Tese (Doutorado) — Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas, jul 2004. 12, 19

BRITANNICA, E. *Microphone*: Electroacoustic device. Encyclopædia Britannica, Inc, 2019. Disponível em: <https://www.britannica.com/technology/microphone-electroacoustic-device>. Acesso em: 29 jun. 2019. 17

CHANDLER, D. M.; HEMAMI, S. S. Vsnr: A wavelet-based visual signal-to-noise ratio for natural images. *IEEE*, IEEE Transactions on Image Processing, v. 16, p. 2284–2298, 2007. 20

GOTTARDI, G. *peaqb-fast*. 2015. Disponível em: <https://github.com/akinori-ito/peaqb-fast/tree/master>. Acesso em: 07 fev. 2024. 23

HAWKINS, J. E. *Human ear*: Anatomy. Encyclopædia Britannica, Inc, 2018. Disponível em: <https://www.britannica.com/science/ear>. Acesso em: 24 jun. 2019. 14

HYDROGENAUDIO. *Recommended Ogg Vorbis*. 2018. Disponível em: <https://wiki.hydrogenaud.io/index.php?title=Recommended_Ogg_Vorbis>. Acesso em: 24 fev. 2024. 18

ITU. *Recommendation ITU-R BS.1387-1: Method for objective measurements of perceived audio quality*. [S.l.], 2001. 19, 23

KAMEDO. *Results of the public multiformat listening test (July 2014)*. 2014. Disponível em: <https://listening-test.coresv.net/results.htm>. Acesso em: 27 jun. 2019. 7, 21, 23, 26

MATHIAS, S. R. *Equal-loudness contours*. 2019. Disponível em: <https://crackedbassoon.com/writing/equal-loudness-contours>. Acesso em: 23 fev. 2024. 23

PALOMAR, D. et al. Objective assessment of audio quality. In: *IET Irish Signals and Systems Conference (ISSC 2008)*. [S.l.: s.n.], 2008. p. 37–42. ISSN 0537-9989. 12, 20

SHENOI, B. A. *Introduction to digital signal processing and filter design*. [S.l.]: Wiley-India, 2010. 21

SPANIAS, A. *Audio signal processing and coding*. Hoboken, N.J: Wiley-Interscience, 2007. ISBN 9780471791478. 11, 15, 16, 17

SPOTIFY. *Formatos de arquivo de áudio para o Spotify*. 2024. Disponível em: <https://support.spotify.com/br-pt/artists/article/audio-file-formats/>. Acesso em: 22 fev. 2024. 12, 18

SUZUKI, Y.; TAKESHIMA, H.; KURAKATA, K. Revision of iso 226 "normal equal-loudness-level contours"from 2003 to 2023 edition: The background and results. ACOUSTICAL SOCIETY OF JAPAN, v. 45, n. 1, p. 1–8, 2024. ISSN 1346-3969. 15

XIPH.ORG, F. *Vorbis I specification*. 2015. Disponível em: <https://xiph.org/vorbis/doc/Vorbis_I_spec.html>. Acesso em: 27 jun. 2019. 12, 17, 18

XIPH.ORG, F. *Opus Codec*. 2017. Disponível em: <http://opus-codec.org/>. Acesso em: 27 jun. 2019. 12, 18, 19

XIPH.ORG, F. *Games that use Vorbis*. 2018. Disponível em: <https://wiki.xiph.org/Games_that_use_Vorbis>. Acesso em: 22 fev. 2024. 12, 18

# Apêndices

# APÊNDICE A – COEFICIENTES DO FILTRO PROJETADO

Tabela 9 – Coeficientes do filtro projetado com a curva de 60 $fon$

| Posição | Coeficientes | | | | |
|---|---|---|---|---|---|
| [0 : 4] | -0.00168 | -0.00175 | -0.00178 | -0.00128 | -0.00102 |
| [5 : 9] | -0.00112 | -0.00149 | -0.00125 | -0.00125 | -0.00154 |
| [10 : 14] | -0.00211 | -0.00233 | -0.00312 | -0.00400 | -0.00508 |
| [15 : 19] | -0.00590 | -0.00701 | -0.00734 | -0.00787 | -0.00884 |
| [20 : 24] | -0.01070 | -0.01200 | -0.01395 | -0.01541 | -0.01565 |
| [20 : 29] | -0.01397 | -0.01406 | -0.01486 | -0.01506 | -0.01283 |
| [30 : 34] | -0.01116 | -0.00821 | -0.00566 | -0.00279 | -0.00104 |
| [30 : 39] | 0.00398 | 0.00656 | 0.00748 | 0.00669 | 0.00878 |
| [40 : 44] | -0.00096 | -0.01694 | -0.03477 | -0.04599 | -0.07585 |
| [45 : 49] | -0.08455 | -0.05006 | 0.05306 | 0.07708 | 0.22126 |
| [50 : 54] | 0.44700 | 0.22126 | 0.07708 | 0.05306 | -0.05006 |
| [55 : 59] | -0.08455 | -0.07585 | -0.04599 | -0.03477 | -0.01694 |
| [60 : 64] | -0.00096 | 0.00878 | 0.00669 | 0.00748 | 0.00656 |
| [65 : 69] | 0.00398 | -0.00104 | -0.00279 | -0.00566 | -0.00821 |
| [70 : 74] | -0.01116 | -0.01283 | -0.01506 | -0.01486 | -0.01406 |
| [75 : 79] | -0.01397 | -0.01565 | -0.01541 | -0.01395 | -0.01200 |
| [80 : 84] | -0.01070 | -0.00884 | -0.00787 | -0.00734 | -0.00701 |
| [85 : 89] | -0.00590 | -0.00508 | -0.00400 | -0.00312 | -0.00233 |
| [90 : 94] | -0.00211 | -0.00154 | -0.00125 | -0.00125 | -0.00149 |
| [95 : 99] | -0.00112 | -0.00102 | -0.00128 | -0.00178 | -0.00175 |
| [100] | -0.00168 | | | | |

Fonte: Elaborada pelo autor.

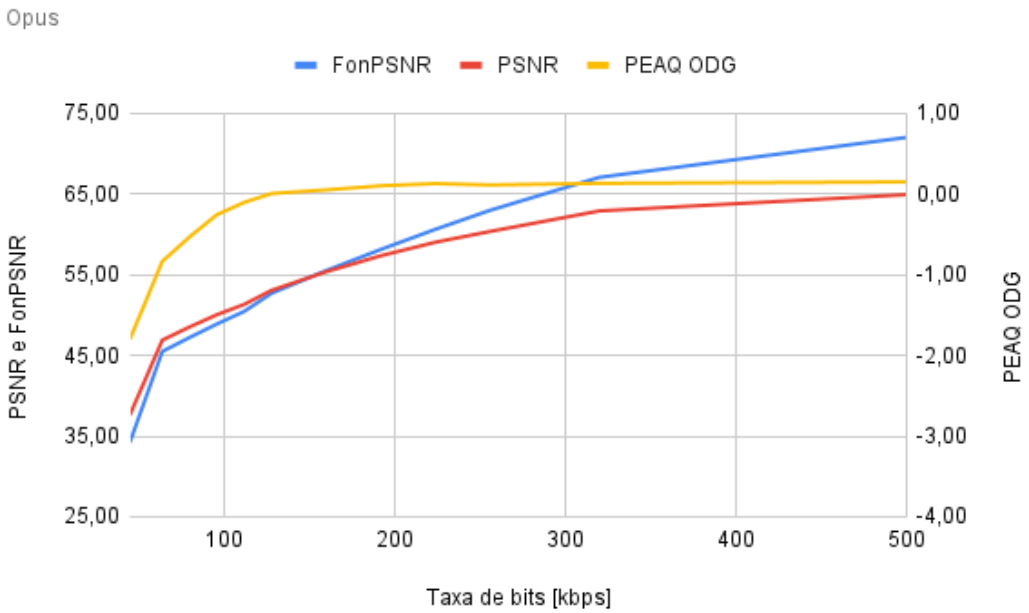
# APÊNDICE B – RESULTADO DA AVALIAÇÃO DA QUALIDADE DE ÁUDIOS

Tabela 10 – Resultado da avaliação da qualidade do áudio “12-German-male-speech” variando a taxa de bits

| | Opus | | | Vorbis | | |
|---|---|---|---|---|---|---|
| Taxa de bits [kbps] | FonPSNR | PSNR | PEAQ ODG | FonPSNR | PSNR | PEAQ ODG |
| 45 | 34,23 | 37,62 | -1,80 | 38,06 | 39,79 | -3,29 |
| 64 | 45,48 | 46,89 | -0,83 | 40,44 | 42,84 | -1,43 |
| 80 | 47,22 | 48,53 | -0,53 | 42,63 | 45,03 | -0,64 |
| 96 | 48,89 | 50,04 | -0,26 | 43,75 | 46,18 | -0,46 |
| 112 | 50,45 | 51,32 | -0,10 | 44,64 | 47,36 | -0,32 |
| 128 | 52,69 | 53,06 | 0,00 | 44,90 | 47,82 | -0,27 |
| 160 | 55,51 | 55,36 | 0,05 | 46,15 | 49,61 | -0,31 |
| 192 | 58,12 | 57,34 | 0,10 | 49,19 | 52,85 | -0,11 |
| 224 | 60,62 | 59,02 | 0,13 | 51,96 | 55,36 | 0,06 |
| 256 | 62,97 | 60,35 | 0,11 | 56,87 | 59,67 | 0,15 |
| 320 | 67,05 | 62,89 | 0,13 | 62,80 | 63,61 | 0,13 |
| 500 | 72,00 | 64,91 | 0,15 | 68,19 | 67,23 | 0,16 |

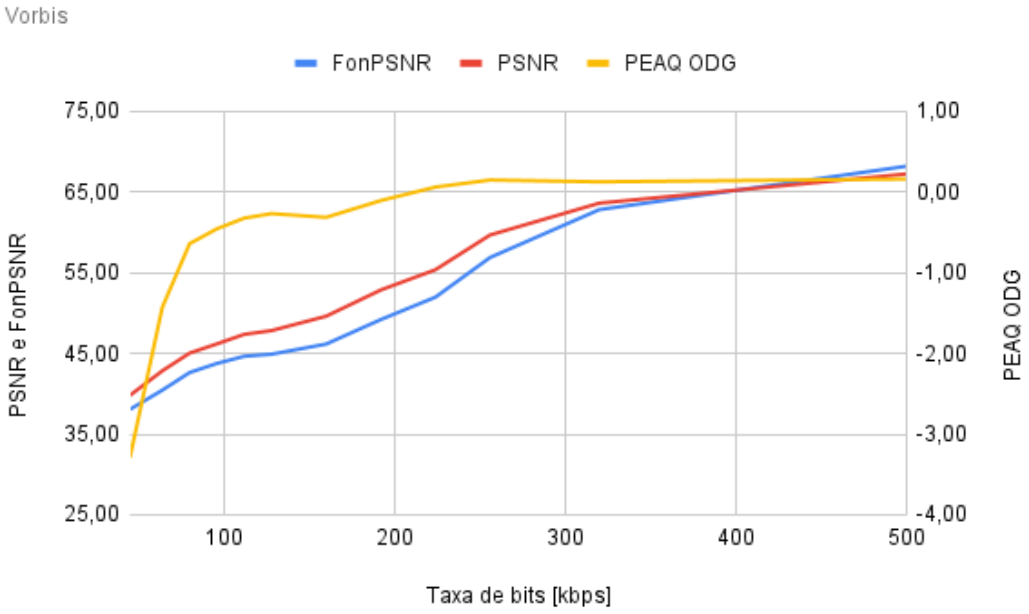
Fonte: Elaborada pelo autor.

Figura 8 – Variação das métricas de qualidade em relação a taxa de bits: áudio “12-German-male-speech” codificado com Opus




Fonte: Elaborada pelo autor.

Figura 9 – Variação das métricas de qualidade em relação a taxa de bits: áudio "12-German-male-speech" codificado com Vorbis



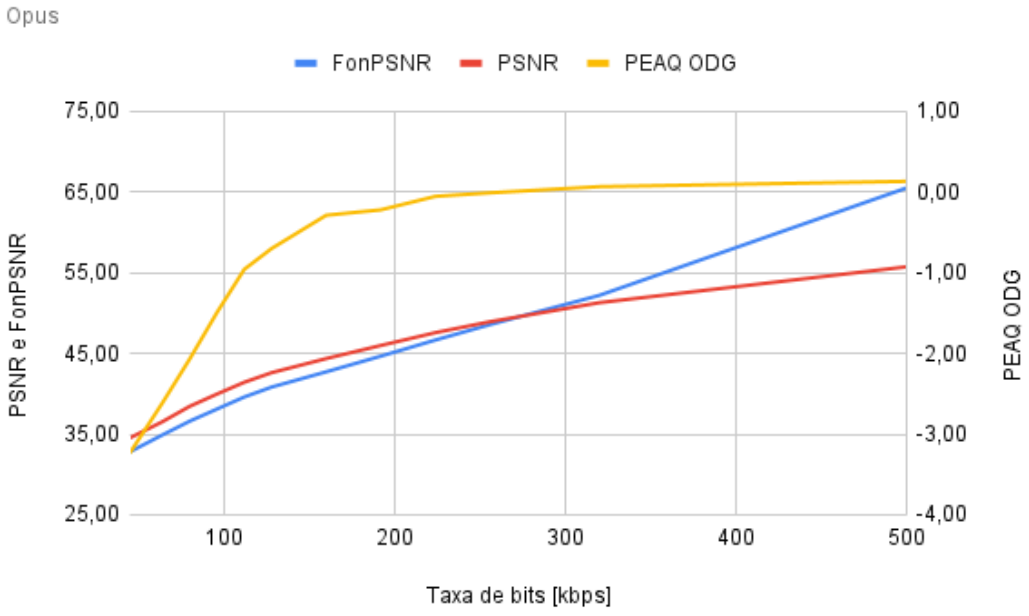

Fonte: Elaborada pelo autor.

Tabela 11 – Resultado da avaliação da qualidade do áudio “15-Good-evening” variando a taxa de bits

| | Opus | | | Vorbis | | |
|---|---|---|---|---|---|---|
| Taxa de bits [kbps] | FonPSNR | PSNR | PEAQ ODG | FonPSNR | PSNR | PEAQ ODG |
| 45 | 32,80 | 34,52 | -3,23 | 28,50 | 29,24 | -2,96 |
| 64 | 34,89 | 36,53 | -2,62 | 30,99 | 32,08 | -2,20 |
| 80 | 36,60 | 38,44 | -2,07 | 32,85 | 34,58 | -1,38 |
| 96 | 38,10 | 39,96 | -1,49 | 35,57 | 37,27 | -0,88 |
| 112 | 39,59 | 41,42 | -0,96 | 38,37 | 39,90 | -0,56 |
| 128 | 40,82 | 42,61 | -0,70 | 38,76 | 40,46 | -0,50 |
| 160 | 42,71 | 44,35 | -0,29 | 40,08 | 42,13 | -0,22 |
| 192 | 44,65 | 45,98 | -0,22 | 42,26 | 44,76 | -0,07 |
| 224 | 46,65 | 47,57 | -0,05 | 44,36 | 46,50 | 0,07 |
| 256 | 48,57 | 48,92 | -0,01 | 47,00 | 49,34 | 0,09 |
| 320 | 52,18 | 51,28 | 0,07 | 51,89 | 52,79 | 0,11 |
| 500 | 65,49 | 55,72 | 0,13 | 58,62 | 58,63 | 0,15 |

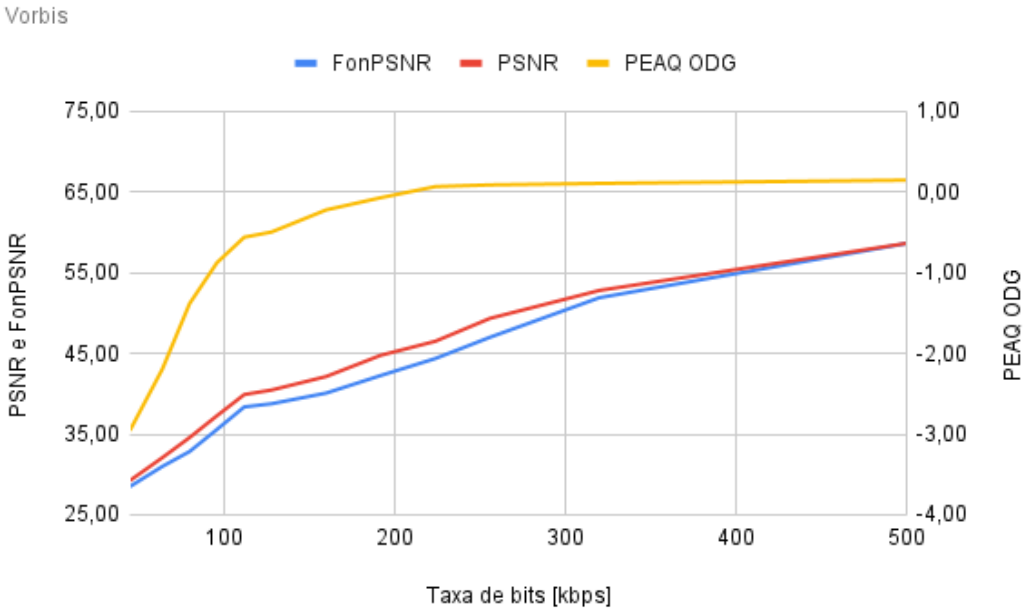
Fonte: Elaborada pelo autor.

Figura 10 – Variação das métricas de qualidade em relação a taxa de bits: áudio “15-Good-evening” codificado com Opus




Fonte: Elaborada pelo autor.

Figura 11 – Variação das métricas de qualidade em relação a taxa de bits: áudio “15-Good-evening” codificado com Vorbis



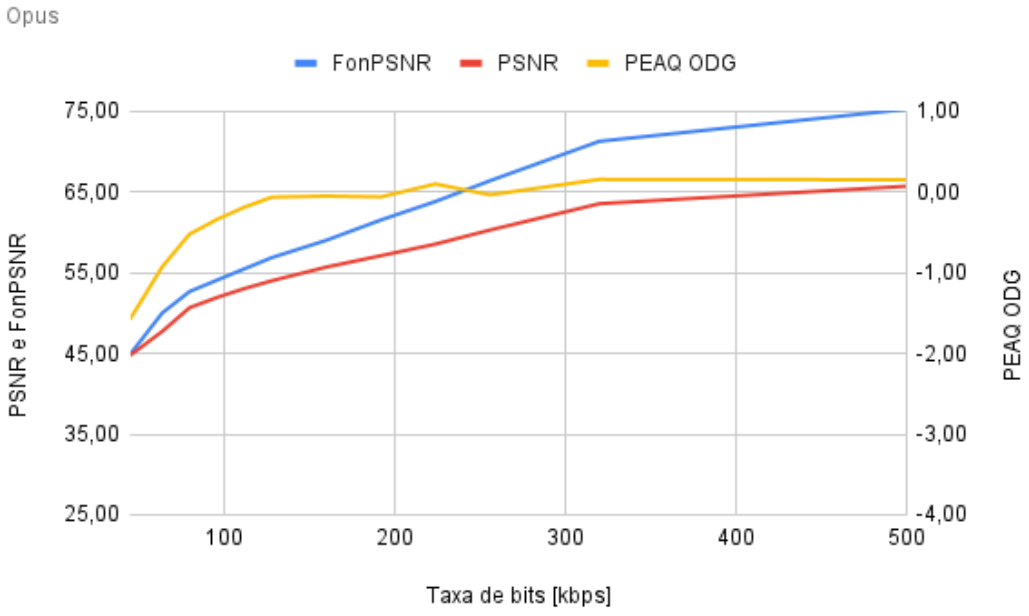

Fonte: Elaborada pelo autor.

Tabela 12 – Resultado da avaliação da qualidade do áudio “24-Greensleeves-Korean-male-speech” variando a taxa de bits

| | Opus | | | Vorbis | | |
|---|---|---|---|---|---|---|
| Taxa de bits [kbps] | FonPSNR | PSNR | PEAQ ODG | FonPSNR | PSNR | PEAQ ODG |
| 45 | 44,85 | 44,73 | -1,58 | 39,24 | 41,18 | -3,40 |
| 64 | 50,02 | 47,73 | -0,92 | 41,06 | 43,85 | -1,82 |
| 80 | 52,66 | 50,67 | -0,52 | 43,86 | 46,27 | -0,96 |
| 96 | 54,07 | 51,92 | -0,34 | 46,36 | 48,40 | -0,66 |
| 112 | 55,46 | 53,02 | -0,19 | 47,38 | 49,95 | -0,50 |
| 128 | 56,85 | 53,99 | -0,07 | 46,89 | 50,08 | -0,36 |
| 160 | 59,01 | 55,69 | -0,05 | 48,81 | 51,99 | -0,30 |
| 192 | 61,52 | 57,10 | -0,06 | 55,11 | 57,36 | -0,08 |
| 224 | 63,81 | 58,54 | 0,10 | 60,66 | 61,19 | 0,13 |
| 256 | 66,39 | 60,28 | -0,04 | 63,20 | 63,38 | 0,10 |
| 320 | 71,27 | 63,54 | 0,16 | 66,99 | 66,08 | 0,10 |
| 500 | 75,24 | 65,71 | 0,15 | 70,95 | 69,20 | 0,15 |

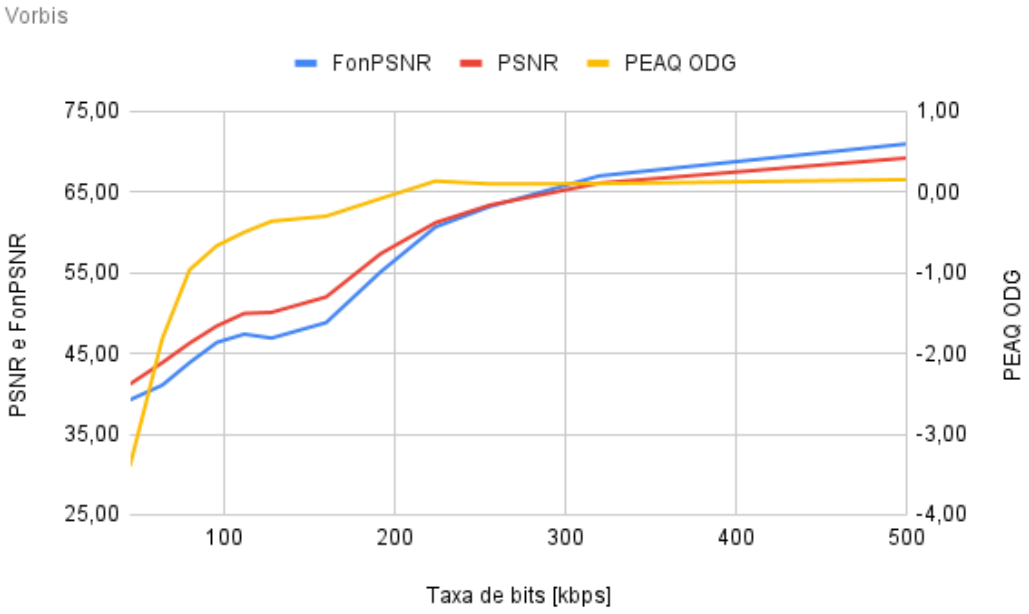
Fonte: Elaborada pelo autor.

Figura 12 – Variação das métricas de qualidade em relação a taxa de bits: áudio “24-Greensleeves-Korean-male-speech” codificado com Opus




Fonte: Elaborada pelo autor.

Figura 13 – Variação das métricas de qualidade em relação a taxa de bits: áudio “24-Greensleeves-Korean-male-speech” codificado com Vorbis



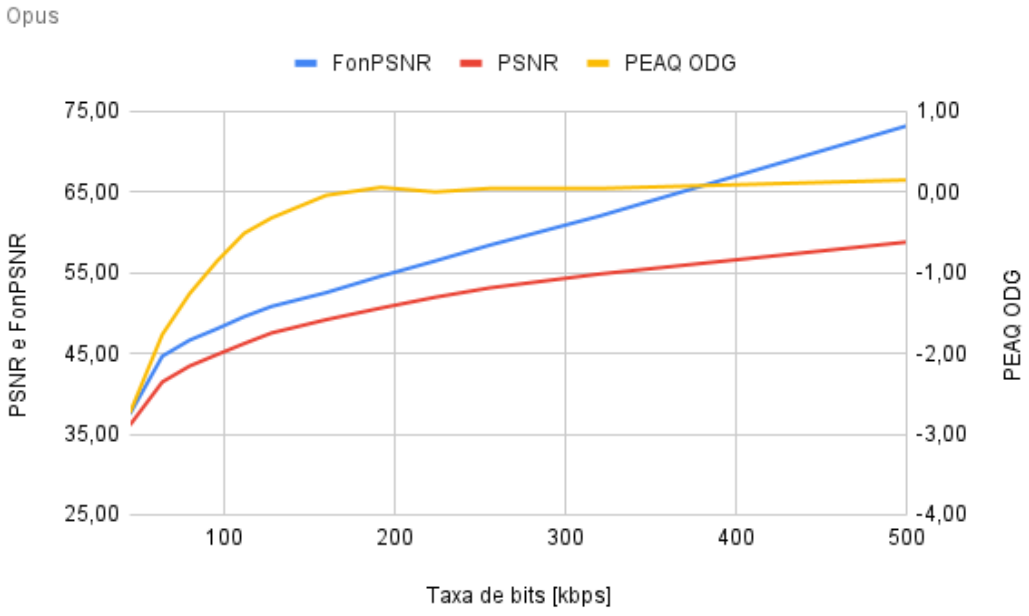

Fonte: Elaborada pelo autor.

Tabela 13 – Resultado da avaliação da qualidade do áudio “25-This-is-the-end” variando a taxa de bits

| | Opus | | | Vorbis | | |
|---|---|---|---|---|---|---|
| Taxa de bits [kbps] | FonPSNR | PSNR | PEAQ ODG | FonPSNR | PSNR | PEAQ ODG |
| 45 | 37,48 | 36,11 | -2,74 | 38,90 | 37,03 | -3,46 |
| 64 | 44,65 | 41,46 | -1,77 | 40,00 | 38,80 | -2,39 |
| 80 | 46,64 | 43,43 | -1,26 | 41,65 | 40,31 | -1,47 |
| 96 | 48,04 | 44,84 | -0,86 | 43,45 | 42,33 | -0,83 |
| 112 | 49,55 | 46,22 | -0,51 | 44,82 | 44,02 | -0,50 |
| 128 | 50,82 | 47,53 | -0,32 | 45,48 | 45,09 | -0,28 |
| 160 | 52,52 | 49,17 | -0,04 | 46,49 | 46,22 | -0,20 |
| 192 | 54,56 | 50,61 | 0,06 | 49,48 | 48,61 | -0,10 |
| 224 | 56,46 | 51,96 | 0,00 | 51,89 | 50,52 | 0,06 |
| 256 | 58,42 | 53,12 | 0,04 | 55,00 | 53,35 | 0,08 |
| 320 | 62,00 | 54,83 | 0,04 | 60,48 | 56,73 | 0,09 |
| 500 | 73,18 | 58,79 | 0,15 | 67,29 | 60,72 | 0,15 |

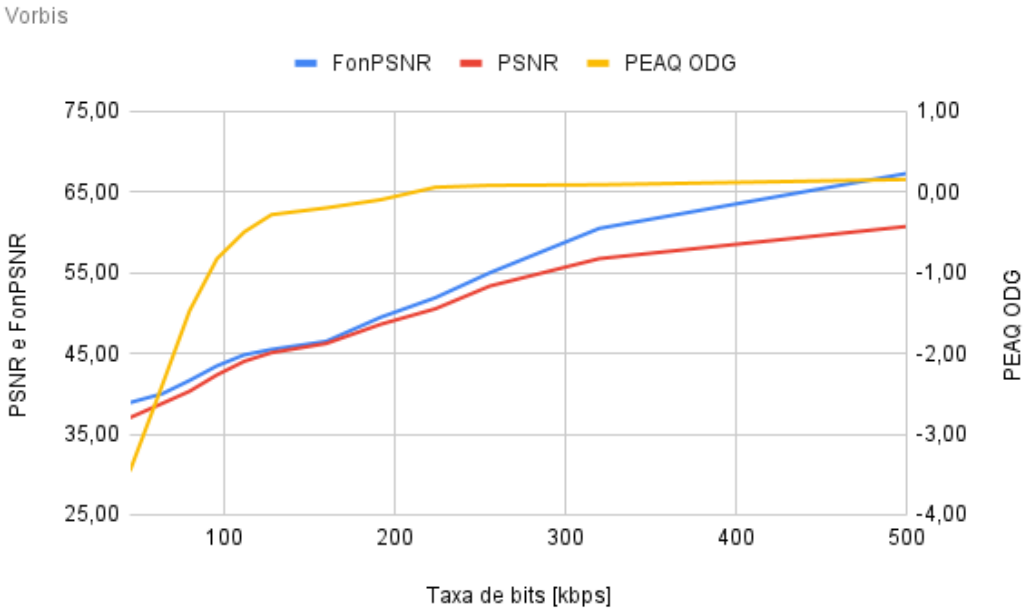
Fonte: Elaborada pelo autor.

Figura 14 – Variação das métricas de qualidade em relação a taxa de bits: áudio “25-This-is-the-end” codificado com Opus




Fonte: Elaborada pelo autor.

Figura 15 – Variação das métricas de qualidade em relação a taxa de bits: áudio “25-This-is-the-end” codificado com Vorbis




Fonte: Elaborada pelo autor.

# APÊNDICE C – CÓDIGOS DO PROJETO

Este projeto foi disponibilizado também via GitHub em:

- https://github.com/luisamachado/TCC-Projeto-FonPSNR

Código C.1 – home/TCC/README.md

```
# TCC: Projeto FonPSNR

## Execução do programa

Para execução deste projeto utilize o seguinte comando:

```python3 handle_change_params.py```
```

Código C.2 – home/TCC/fonpsnr/__init__.py

```
from .fonpsnr import FonPSNR

__all__ = [
    "FonPSNR"
]
```

Código C.3 – home/TCC/fonpsnr/fonpsnr.py

```
import numpy as np
from scipy import (interpolate, signal)
from skimage.metrics import peak_signal_noise_ratio as psnr


class FonPSNR:
    def __init__(self, fon=60):
        self.f, self.af, self.Lu, self.Tf = self._return_const_iso_226_2003()
        self.fir = self.create_fir_filter(fon)

    @staticmethod
    def _return_const_iso_226_2003():
        """Retorna as constantes determionadas pela ISO 226 de 2003"""
        f = np.array([
            20,   25, 31.5,   40,   50,   63,   80,   100,   125,  160,
            200,  250,  315,  400,  500,  630,  800,  1000,  1250, 1600,
            2000, 2500, 3150, 4000, 5000, 6300, 8000, 10000, 12500
        ])
```
```

```python

        af = np.array([
            0.532, 0.506, 0.480, 0.455, 0.432, 0.409, 0.387, 0.367, 0.349, 0.330,
            0.315, 0.301, 0.288, 0.276, 0.267, 0.259, 0.253, 0.250, 0.246, 0.244,
            0.243, 0.243, 0.243, 0.242, 0.242, 0.245, 0.254, 0.271, 0.301
        ])

        Lu = np.array([
            -31.6, -27.2, -23.0, -19.1, -15.9, -13.0, -10.3,  -8.1, -6.2, -4.5,
            -3.1,  -2.0,  -1.1,  -0.4,   0.0,   0.3,   0.5,   0.0, -2.7, -4.1,
            -1.0,   1.7,   2.5,   1.2,  -2.1,  -7.1, -11.2, -10.7, -3.1
        ])

        Tf = np.array([
            78.5, 68.7, 59.5, 51.1, 44.0, 37.5, 31.5, 26.5, 22.1, 17.9,
            14.4, 11.4,  8.6,  6.2,  4.4,  3.0,  2.2,  2.4,  3.5,  1.7,
            -1.3, -4.2, -6.0, -5.4, -1.5,  6.0, 12.6, 13.9, 12.3
        ])
        return f, af, Lu, Tf

    def _equal_loudness_contour(self, fon, frequencies=None):
        """ Retorna uma curva isofônica.

            Args:
                fon (float): Valor fon da curva.
                frequencies (:obj:`np.ndarray`, optional): Frequências para
                    avaliar. Se não for aprovado, todos os 29 pontos do
                    padrão ISO serão retornados. Quaisquer frequências não
                    presentes no padrão são encontradas através de
                    interpolação spline.

            Returns:
                Lp (np.ndarray): valores em db SPL.
        """
        assert 0 <= fon <= 90, f"{fon} is not [0, 90]"
        Af = (
            4.47e-3 * (10 ** (0.025 * fon) - 1.15)
            + (0.4 * 10 ** (((self.Tf + self.Lu) / 10) - 9)) ** self.af
        )
        Lp = ((10.0 / self.af) * np.log10(Af)) - self.Lu + 94

        if frequencies is not None:
            assert frequencies.min() >= self.f.min(), "Frequencies are too low"
            assert frequencies.max() <= self.f.max(), "Frequencies are too high"
            tck = interpolate.splrep(self.f, Lp, s=0)
            Lp = interpolate.splev(frequencies, tck, der=0)
        return Lp
```

```

    def _generate_curva_fon(self, fon):
        x = self.f
        curve_fon = self._equal_loudness_contour(fon, x)

        x = np.insert(x, 0, 0)
        x = np.append(x, 22050)
        curve_fon = np.insert(curve_fon, 0, curve_fon[0])
        curve_fon = np.append(curve_fon, 0)

        return x, curve_fon

    def create_fir_filter(self, fon=60):
        x, curve = self._generate_curva_fon(fon)
        fs = 44100.0
        gain = 10**((fon - curve) / 20)
        freq = x / (fs / 2)
        gain[0] = 0
        gain[-1] = 0
        numtaps = 101
        fir = signal.firwin2(
            numtaps, freq, gain, window=("kaiser", 0.5), antisymmetric=False)
        return fir

    def fonpsnr(self, original_data, dec_data):
        original_filtered = signal.filtfilt(self.fir, 1, original_data)
        coded_filtered = signal.filtfilt(self.fir, 1, dec_data)
        data_range = original_filtered.max() - original_filtered.min()
        result = psnr(original_filtered, coded_filtered, data_range=data_range)
        return result
```

Código C.4 – home/TCC/analyzer/__init__.py

```
from .audio_analyzer import AudioAnalyzer

__all__ = [
    "AudioAnalyzer"
]
```

Código C.5 – home/TCC/analyzer/audio_analyzer.py

```
import audiofile
import csv
import locale
import os
import audio_metadata

from sklearn.metrics import mean_squared_error as mse
```

```
8  from skimage.metrics import peak_signal_noise_ratio as psnr
9
10 from fonpsnr import FonPSNR
11
12 locale.setlocale(locale.LC_ALL, 'pt_BR.utf8')
13
14 class AudioAnalyzer:
15     def __init__(self, param_type="param_type"):
16         self.fonpsnr = FonPSNR(fon=60)
17         self.param_type = param_type
18         self.fieldnames = [
19             "filename",
20             self.param_type,
21             "original_data_size",
22             "original_bitrate",
23             "FonPSNR",
24             "PSNR",
25             "PEAQ_ODG",
26             "PEAQ_DI",
27             "MSE",
28         ]
29
30     @staticmethod
31     def adjust_special_characters(value):
32         new_value = value.replace("\'", "\\\'")
33         new_value = new_value.replace(" ", "\ ")
34         new_value = new_value.replace(",", "\,")
35         new_value = new_value.replace("(", "\(")
36         return new_value.replace(")", "\)")
37
38     def _convert_number_to_locale(self, number):
39         number_float = float(number)
40         number_locale = locale.str(number_float)
41         return number_locale
42
43     def _read_file(self, filename):
44         data, _ = audiofile.read(filename)
45         return data
46
47     def _read_metadata_file_original(self, filename, info_list):
48         metadata_original = audio_metadata.load(filename)
49         info_list["original_data_size"] = metadata_original.streaminfo._size
50         info_list["original_bitrate"] = metadata_original.streaminfo.bitrate
51
52     def calculator(self, original_data, dec_data, info_list):
53         if original_data.size == dec_data.size:
54             psnr_result = psnr(original_data, dec_data)
```

```
55            info_list["PSNR"] = self._convert_number_to_locale(psnr_result)
56            mse_result = mse(original_data, dec_data)
57            info_list["MSE"] = self._convert_number_to_locale(mse_result)
58            fonpsnr_result = self.fonpsnr.fonpsnr(original_data, dec_data)
59            info_list["FonPSNR"] = self._convert_number_to_locale(fonpsnr_result)
60
61        def comparator_peaq(self, original_file_path, dec_file_path, info_list):
62            command_peaqb = (
63                "./peaqb -r %(original_file_path)s -t %(dec_file_path)s > peaq-result.
    txt"
64            )
65            command_odg = (
66                "cat peaq-result.txt | grep ODG | sed -n -e 's/^.*ODG: //p' " +
67                "| awk '{ sum += $1; n++ } END { if (n > 0) print sum / n; }'"
68            )
69            command_di = (
70                "cat peaq-result.txt | grep DI | sed -n -e 's/^.*DI: //p' " +
71                "| awk '{ sum += $1; n++ } END { if (n > 0) print sum / n; }'"
72            )
73            name_files = {
74                "original_file_path": self.adjust_special_characters(original_file_path
    ),
75                "dec_file_path": self.adjust_special_characters(dec_file_path)
76            }
77            run_peaqb = os.popen(command_peaqb % name_files)
78            run_peaqb.close()
79            pipe_odg = os.popen(command_odg)
80            result_command_odg = pipe_odg.read().replace("\n", "")
81            info_list["PEAQ_ODG"] = self._convert_number_to_locale(result_command_odg)
82            pipe_odg.close()
83            pipe_di = os.popen(command_di)
84            result_command_di = pipe_di.read().replace("\n", "")
85            info_list["PEAQ_DI"] = self._convert_number_to_locale(result_command_di)
86            pipe_di.close()
87            run_peaq_result = os.popen("rm peaq-result.txt")
88            run_peaq_result.close()
89
90        def extract_infos(self, audio_info):
91            self.dec_dir_path = audio_info["dec_dir_path"]
92            self.codec_type = audio_info["codec_type"]
93            self.param_codec = audio_info.get("param_codec", "")
94            self.param_value = audio_info.get("param_value")
95            self.dec_filename_list = os.listdir(self.dec_dir_path)
96
97        def generate_csv(self, codec_type, partial_filename, table):
98            try:
99                csv_file = f"comparator-{codec_type}-{partial_filename}.csv"
```

```
100             with open(csv_file, "w") as csvfile:
101                 writer = csv.DictWriter(csvfile, self.fieldnames, delimiter=';')
102                 writer.writeheader()
103                 for row in table:
104                     for data in row.values():
105                         writer.writerow(data)
106         except IOError:
107             print("I/O error")
108 
109     def analyzer(self, original_filename, audio_base_dir_path, audio_info):
110         spreadsheet = {}
111         self.extract_infos(audio_info)
112         info_list = {}
113         partial_filename, _ = os.path.splitext(original_filename)
114         info_list["filename"] = partial_filename
115         info_list[self.param_type] = self.param_value
116         codec_params = (
117             f"{self.codec_type} {self.param_codec} {self.param_value}"
118             if self.param_codec and f"{self.param_value}"
119             else f"{self.codec_type}"
120         )
121         original_file_path = os.path.join(audio_base_dir_path, original_filename)
122         dec_file_path = os.path.join(self.dec_dir_path, original_filename)
123         self._read_metadata_file_original(original_file_path, info_list)
124         original_data = self._read_file(original_file_path)
125         dec_data = self._read_file(dec_file_path)
126         self.calculator(original_data, dec_data, info_list)
127         self.comparator_peaq(original_file_path, dec_file_path, info_list)
128         line_name = f"{partial_filename} {codec_params}"
129         spreadsheet[line_name] = info_list
130         return spreadsheet
131 
132     def handle_analyzer(
133             self, audio_decode_path, params_coding, table, original_filename,
    audio_base_path):
134         param_codec = params_coding["param_codec"]
135         params_value = params_coding["params_value"]
136         codec_type = params_coding["codec_type"]
137         for value in params_value:
138             config_type = param_codec.replace("--", "")
139             decoded_path = f"{audio_decode_path}{config_type}_{value}/"
140             analysis_params = {
141                 "dec_dir_path": decoded_path,
142                 "codec_type": codec_type,
143                 "param_codec": param_codec,
144                 "param_value": value,
145             }
```

```
146             table.append(self.analyzer(original_filename, audio_base_path,
    analysis_params))
147
148     def audio_analyzer(self, audio_decode_path, params_coding):
149         audio_base_path = params_coding["audio_base_path"]
150         audio_base_filename_list = os.listdir(audio_base_path)
151         for original_filename in audio_base_filename_list:
152             table = []
153             partial_filename, _ = os.path.splitext(original_filename)
154             self.handle_analyzer(
155                 audio_decode_path, params_coding, table, original_filename,
    audio_base_path)
156             self.generate_csv(params_coding["codec_type"], partial_filename, table)
```

Código C.6 – home/TCC/codec/codec_audio.py

```
1  import os
2  import subprocess
3  from handle_folder import handle_folder
4
5  def adjust_special_characters(value):
6      """ Ajusta os caracteres especiais do nome de arquivo para
7          o formato aceito pela linha de comando do Linux
8
9          Args:
10             filename: Nome do arquivo
11
12         Returns:
13             new_filename: Nome do arquivo ajustado
14
15     """
16     new_value = value.replace("\'", "\\\'")
17     new_value = new_value.replace(" ", "\ ")
18     new_value = new_value.replace(",", "\,")
19     new_value = new_value.replace("(", "\(")
20     return new_value.replace(")", "\)")
21
22
23 def check_file_exist(filename_list, codec_path):
24     """ Verifica se os arquivos codificados/decodificados já existem no diretório
25
26         Args:
27             filename_list: Lista com os nomes dos arquivos
28             codec_path: Caminho da pasta que será verificada a existência dos
29                         arquivos da lista 'filename_list'
30
31         Returns:
32             count_exist: Número correspondente a quantidade de arquivos da lista
```

```
33                          'filename_list' que está presente na pasta 'codec_path'
34     """
35     codec_path_list = os.listdir(codec_path)
36     count_exist = 0
37     for name in filename_list:
38         name, _ = os.path.splitext(name)
39         for name_codec in codec_path_list:
40             name_codec, _ = os.path.splitext(name_codec)
41             if name_codec == name:
42                 count_exist = count_exist + 1
43     return count_exist
44
45
46 def recursive_audio_encoder(params_coding, encode_config):
47     dir_main_path = params_coding["dir_main_path"]
48     audio_base_path = params_coding["audio_base_path"]
49     codec_dir = params_coding["codec_type"]
50     param_codec = params_coding["param_codec"]
51     params_value = params_coding["params_value"]
52     audio_encode_path = f"{dir_main_path}/{codec_dir}_encode_audio/"
53     handle_folder.create_folder(audio_encode_path)
54     for value in params_value:
55         config_type = param_codec.replace("--", "")
56         encode_path = f"{audio_encode_path}{config_type}_{value}/"
57         handle_folder.create_folder(encode_path)
58         encode_params = {
59             "param_codec": param_codec,
60             "param_value": value,
61         }
62         encode_audios(
63             audio_base_path, encode_path, encode_config, encode_params)
64
65     print("
       ------------------------------------------------------------------------------")
66     print(f"Os diretórios com os arquivos codificados com {codec_dir} se encontram
       em:\n")
67     print(audio_encode_path)
68     print("
       ------------------------------------------------------------------------------")
69     return audio_encode_path
70
71
72 def encode_audios(audio_base_path, encode_path, encode_config, encode_params):
73     """ Codifica todos os áudios no formato wav de um diretório
74
75         Args:
76             audio_base_path: Caminho do diretório com os áudios do tipo wav
```

```
77              encode_path: Caminho do diretório que ficarão os arquivos codificados
78              encode_config: Configurações do comando do codificador
79              encode_params: Parâmetros de codificação
80      """
81      command = encode_config["command"]
82      extension_codec = encode_config["extension"]
83      param_codec = encode_params.get("param_codec", "")
84      param_value = encode_params.get("param_value", "")
85
86      filename_list = os.listdir(audio_base_path)
87      count_exist = check_file_exist(filename_list, encode_path)
88      if count_exist == len(filename_list):
89          return
90
91      for name in filename_list:
92          original_filename = os.path.join(audio_base_path, name)
93          adjusted_filename = adjust_special_characters(name)
94          original_adjusted_filename = os.path.join(audio_base_path,
        adjusted_filename)
95
96          filename, _ = os.path.splitext(adjusted_filename)
97          new_filename = f"{filename}{extension_codec}"
98          encode_filename = os.path.join(encode_path, new_filename)
99          params_command = {
100             "audiofile": encode_filename,
101             "audiofile_wav": original_adjusted_filename,
102             "param_codec": param_codec,
103             "param_value": param_value,
104         }
105         try:
106             subprocess.check_call(command % params_command, shell=True, stderr=
        subprocess.DEVNULL)
107         except subprocess.CalledProcessError:
108             print("
        ------------------------------------------------------------------------------")
109             print("Ocorreu algum erro no processo")
110             print(f"Verifique o arquivo: '{original_filename}'")
111             print("
        ------------------------------------------------------------------------------")
112
113
114 def recursive_audio_decoder(audio_encode_path, params_coding, decode_config):
115     dir_main_path = params_coding["dir_main_path"]
116     codec_dir = params_coding["codec_type"]
117     param_codec = params_coding["param_codec"]
118     params_value = params_coding["params_value"]
119     audio_decode_path = f"{dir_main_path}/{codec_dir}_decode_audio/"
```

```
120     for value in params_value:
121         config_type = param_codec.replace("--", "")
122         encode_path = f"{audio_encode_path}{config_type}_{value}/"
123         decoded_path = f"{audio_decode_path}{config_type}_{value}/"
124         handle_folder.create_folder(audio_decode_path)
125         handle_folder.create_folder(decoded_path)
126         decode_audios(encode_path, decoded_path, decode_config)
127 
128     print("
    ------------------------------------------------------------------------------")
129     print(f"Os diretórios com os arquivos decodificados com {codec_dir} se
    encontram em:\n")
130     print(audio_decode_path)
131     print("
    ------------------------------------------------------------------------------")
132     return audio_decode_path
133 
134 
135 def decode_audios(encode_path, decode_path, decode_config):
136     """ Decodifica todos os áudios no formato ogg ou opus de um diretório
137 
138     Args:
139         encode_path: Caminho do diretório com os áudios do tipo ogg ou opus
140         decode_path: Caminho do diretório que ficarão os arquivos decodificados
141         decode_config: Configurações do comando do decodificador
142     """
143     command = decode_config["command"]
144     dir_decod_len = len(os.listdir(decode_path))
145     if dir_decod_len != 0:
146         return
147 
148     filenames_encod = os.listdir(encode_path)
149     for filename in filenames_encod:
150         encode_filename = os.path.join(encode_path, filename)
151         adjusted_filename = adjust_special_characters(filename)
152         filename, _ = os.path.splitext(adjusted_filename)
153         new_filename = f"{filename}.wav"
154         params_command = {
155             "audiofile": os.path.join(encode_path, adjusted_filename),
156             "audiofile_wav": os.path.join(decode_path, new_filename),
157         }
158 
159         try:
160             subprocess.check_call(command % params_command, shell=True, stderr=
    subprocess.DEVNULL)
161         except subprocess.CalledProcessError:
162             print("
```

```
    ------------------------------------------------------------------------------")
            print("Ocorreu algum erro no processo")
            print(f"Verifique o arquivo: '{encode_filename}'")
            print("
    ------------------------------------------------------------------------------")
```

Código C.7 – home/TCC/handle_folder/handle_folder.py

```
import os


def is_folder(dir_name):
    return os.path.isdir(dir_name)


def check_folder(dir_name):
    if not is_folder(dir_name):
        print(f"Diretório '{dir_name}' não encontrado")
        return


def create_folder(dir_name):
    if is_folder(dir_name):
        return

    os.mkdir(dir_name)
```

Código C.8 – home/TCC/handle_change_params.py

```
import os

from analyzer import AudioAnalyzer
from codec import codec_audio
from handle_folder import handle_folder


def handle_codec_vorbis(dir_main_path, audio_base_path):
    codec_type = "vorbis"
    param_type = "bitrate [kbps]"
    param_codec = "--bitrate"
    params_value = [45, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 500,]
    params_audio_coding = {
        "dir_main_path": dir_main_path,
        "audio_base_path": audio_base_path,
        "param_codec": param_codec,
        "params_value": params_value,
        "codec_type": codec_type,
    }
```

```
    encode_audio_config = {
        "command":
            "oggenc %(audiofile_wav)s --output=%(audiofile)s %(param_codec)s %(
    param_value)s",
        "extension": ".ogg",
        "codec_type": codec_type,
    }
    decode_audio_config = {
        "command": "oggdec %(audiofile)s --output=%(audiofile_wav)s",
        "extension": ".ogg",
        "codec_type": codec_type,
    }
    audio_encode_path = codec_audio.recursive_audio_encoder(
        params_audio_coding, encode_audio_config)
    audio_decode_path = codec_audio.recursive_audio_decoder(
        audio_encode_path, params_audio_coding, decode_audio_config)
    audio_analyzer = AudioAnalyzer(param_type)
    audio_analyzer.audio_analyzer(audio_decode_path, params_audio_coding)


def handle_codec_opus(dir_main_path, audio_base_path):
    codec_type = "opus"
    param_type = "bitrate [kbps]"
    param_codec = "--bitrate"
    params_value = [45, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 500,]
    params_audio_coding = {
        "dir_main_path": dir_main_path,
        "audio_base_path": audio_base_path,
        "param_codec": param_codec,
        "params_value": params_value,
        "codec_type": codec_type,
    }
    encode_audio_config = {
        "command":
            "opusenc %(audiofile_wav)s %(audiofile)s %(param_codec)s %(param_value)
    s",
        "extension": ".opus",
        "codec_type": codec_type,
    }
    decode_audio_config = {
        "command": "opusdec %(audiofile)s %(audiofile_wav)s",
        "extension": ".opus",
        "codec_type": codec_type,
    }

    audio_encode_path = codec_audio.recursive_audio_encoder(
        params_audio_coding, encode_audio_config)
```

```
65     audio_decode_path = codec_audio.recursive_audio_decoder(
66         audio_encode_path, params_audio_coding, decode_audio_config)
67     audio_analyzer = AudioAnalyzer(param_type)
68     audio_analyzer.audio_analyzer(audio_decode_path, params_audio_coding)
69
70
71 def __main__():
72     print("""
73
       ------------------------------------------------------------------------------
74           Projeto desenvolvido para codificar e analisar áudios do diretório '
       audio_base'
75
       ------------------------------------------------------------------------------
76           Os áudios do diretório 'audio_base' são codificados e decodificados
77           usando os codecs Vorbis e Opus variando a taxa de bits (bitrate).
78           Em seguida, cada áudio em seu formato original e decodificado passa
79           pelas métricas do analisador e os resultados são registrados em uma
80           planilha para análise posterior.
81
       ------------------------------------------------------------------------------
82     """)
83     dir_main_path = os.path.dirname(os.path.realpath(__file__))
84     audio_base_path = f"{dir_main_path}/audio_base/"
85
86     print(f"O diretório {audio_base_path} será analisado\n")
87
88     if not handle_folder.is_folder(audio_base_path):
89         print(f"O diretório {audio_base_path} não foi encontrado\n")
90         exit()
91
92     handle_codec_opus(dir_main_path, audio_base_path)
93     handle_codec_vorbis(dir_main_path, audio_base_path)
94
95
96 __main__()
```