

Universidade Federal de Sergipe

Relatório de Trabalho de Conclusão de Curso
(Construção de Sistema Embarcado para Classificação de Automóveis
Usando Áudio)

Departamento de Engenharia Elétrica

Orientador: Prof. Dr. Jugurta Rosa Montalvão Filho

Discente: Roberto Souza dos Santos Junior

São Cristóvão - SE

25 de abril de 2024

RESUMO

Este trabalho faz uso de microcontroladores para desenvolver um sistema classificador embarcado capaz de realizar amostragem de áudio e aplicar Análise de Componentes Principais (*PCA*) para classificar esses sinais como motos, carros ou ônibus. O sistema classificador construído com melhor desempenho é composto por dois microcontroladores *ESP32*, um capacitor eletrolítico e outro cerâmico, um microfone *MAX4466* e um computador pessoal. A partir de uma base de dados contendo 21 áudios, com número igual de motos, carros e ônibus gravados em condições semelhantes, o classificador aplicou *PCA* a 6 desses áudios (2 motos, 2 carros e 2 ônibus) para classificar o restante dos dados com 77,21% de acertos para motos, 80,47% para carros e 87,56% para ônibus.

Palavras-chave: Reconhecimento de Padrões, Sistemas Embarcados, Processamento de Áudio.

ABSTRACT

This work is about using microcontrollers to develop a classification embedded system capable of audio sampling and of classifying these audio signals as motorcycles, cars or buses through Principal Component Analysis (PCA). The classifying system with best performance is composed of two ESP32 microcontrollers, an electrolytic capacitor and a ceramic one, a MAX4466 microphone and a personal computer. Using a database containing 21 audio files, with equal number of motorcycles, cars and buses, sampled at similar conditions, the classifying system applied PCA to 6 of these audios (2 motorcycles, 2 cars and 2 buses) to classify the remaining data with 77,21% success for motorcycles, 80,47% for cars and 87,56% for buses.

Keywords: Pattern Recognition, Embedded Systems, Audio Processing.

Sumário

Lista de Figuras	3
1 Introdução	6
1.1 Reconhecimento de padrões	6
1.1.1 Sinais	7
1.1.2 Sinais Contínuos e Discretos	7
1.1.3 Sinais Analógicos e Digitais	9
1.1.4 Espaços vetoriais, subespaços e projeções	11
1.1.5 Autovalores e autovetores	12
1.1.6 Sinais aleatórios e covariância	13
1.1.7 Análise de Componentes Principais (<i>PCA</i>)	15
1.1.8 Transformada de Fourier	16
1.1.9 Transformada de Fourier discreta (<i>DFT</i>)	17
1.1.10 Interpretação matricial da <i>DFT</i>	19
1.2 Sistemas embarcados	19
1.2.1 Internet e protocolos de comunicação	20
1.2.2 Interface <i>socket</i>	22
1.2.3 Protocolo <i>I²S</i>	23
1.2.4 Protocolo <i>UART</i>	23
1.2.5 Conversão AD	24
2 Metodologia	25
2.1 Visão geral	25
2.2 Estrutura de programação	27
2.3 Programação do Sistema A	28
2.3.1 Coleta de amostras (Protocolo <i>I²S</i>)	28
2.3.2 Servidor <i>websocket TCP</i>	32
2.3.3 Cliente <i>websocket TCP</i>	35
2.4 Programação do Sistema B	35
2.4.1 Coleta de amostras (Conversor AD)	36
2.4.2 Comunicação <i>UART</i>	37
2.4.3 Servidor e Cliente	37

2.5	Processamento de áudio	37
2.5.1	Pré Processamento	37
2.5.2	Análise espectral e extração de características	39
2.5.3	Ajustes e normalização	42
2.5.4	Classificação	42
3	Resultados	46
3.1	Resultados com base de dados disponibilizada	47
3.1.1	Amostragem	47
3.1.2	Extração de Características	48
3.1.3	Classificação	49
3.2	Resultados com sistema A	50
3.2.1	Amostragem	50
3.2.2	Extração de Características	51
3.2.3	Classificação	52
3.3	Resultados com sistema B	53
3.3.1	Amostragem	54
3.3.2	Extração de Características	56
3.3.3	Classificação	57
4	Conclusão	59
5	Perspectiva de trabalhos futuros	60
5.1	Substituição do conversor AD no sistema B	60
5.2	Investigação do efeito das condições de gravação no desempenho da classificação	60
5.3	Integração de câmera ao sistema B	60
Referências		62

Lista de Figuras

1.1	Diagrama de funcionamento de um laço indutivo. Fonte: [9]	6
1.2	Trecho de voz humana gravada por um <i>notebook</i> . Fonte: o próprio autor.	7
1.3	Curva traçada a caneta, ilustração de um sinal contínuo. Fonte: o próprio autor.	8
1.4	Ilustração da discretização de um sinal de voz.	9
1.5	Processo de digitalização e discretização de um sinal analógico contínuo com diferentes números de valores de amplitude possíveis.	10
1.6	Processo de digitalização e discretização de um sinal analógico contínuo com 65536 valores de amplitude possíveis.	10
1.7	Exemplo de dois subespaços em \mathbb{R}^3 . Uma reta e um plano que cruzam a origem. Fonte: o próprio autor.	11
1.8	Projeção de um vetor v num subespaço e a distância d até o mesmo subespaço. Fonte: o próprio autor.	12
1.9	Ilustração de variável aleatória como uma função X . Fonte: [8].	13
1.10	Exemplo de interpretação física da transformada de Fourier. A decomposição da luz branca em ondas eletromagnéticas de diferentes frequências. Fonte: [10].	16
1.11	Exemplo do efeito do preenchimento com zeros na construção do espectro do sinal. Fonte: o próprio autor.	18
1.12	Ilustração da estrutura <i>TCP/IP</i> . Fonte: o próprio autor.	21
1.13	Funcionamento básico da interface <i>socket</i> . Fonte: [3].	22
1.14	Ilustração da formatação do endereço <i>socket</i> . Fonte: [3].	23
1.15	Ilustração da transmissão de dados no protocolo <i>I²S</i> . Fonte: [6].	23
1.16	Ilustração da comunicação <i>UART</i> . Fonte: o próprio autor.	24
2.1	Diagrama do funcionamento do sistema classificador A. Fonte: o próprio autor.	25
2.2	Principais componentes utilizados no classificador.	26
2.3	Diagrama do funcionamento do sistema classificador B. Fonte: o próprio autor.	26
2.4	Microfone utilizado no sistema B. Fonte: o próprio autor.	27
2.5	Microfone utilizado no sistema B. Fonte: o próprio autor.	27
2.6	Imagens relacionadas à programação do ESP32.	27
2.7	Extensão oficial da <i>ESP-IDF</i> para <i>VS Code</i> . Fonte: o próprio autor.	28
2.8	Diagrama de pinos do INMP441. Fonte: o próprio autor.	29
2.9	Visualização do protocolo I ² S em diferentes modalidades. Fonte: [6]	29

2.10	Ilustração da comunicação I2S no formato <i>MSB</i> . Fonte: [2]	30
2.11	Ilustração de construção de recortes estacionários a partir da assinatura sonora	38
2.12	Diferentes recortes do espectro de frequências médio da moto da Figura 2.11.	40
2.13	Espectros médios de áudios pertencentes às três classes. Fonte: o próprio autor.	41
2.14	Valores residuais de comparação com a classe de motos para cada janela espectral. Em preto, janelas que foram utilizadas para realizar a <i>PCA</i> . Fonte: o próprio autor.	45
3.1	Organização dos arquivos da base de dados disponível <i>a priori</i> . Fonte: o próprio autor.	46
3.2	Implementação dos sistemas A e B.	46
3.3	Detalhes de regravação usando os sistemas A e B.	47
3.4	Formas de onda dos áudios disponíveis <i>a priori</i> . Fonte: o próprio autor.	47
3.5	Módulo do espectro de frequências médio e desvio padrão para as assinaturas sonoras da base de dados disponível <i>a priori</i> . Fonte: o próprio autor.	48
3.6	Gráfico decrescente dos autovalores de cada classe obtidos a partir da base de dados disponível. Fonte: o próprio autor.	48
3.7	Valores residuais e limiares de classificação para a base original. Fonte: o próprio autor.	49
3.8	Esquema elétrico do sistema A. Fonte: o próprio autor.	50
3.9	Base de dados reamostrada pelo sistema A. Fonte: o próprio autor.	51
3.10	Efeito da reamostragem do áudio da moto 0 pelo sistema A.	51
3.11	Módulo do espectro de frequências médio e desvio padrão para as assinaturas sonoras da base de dados reamostrada pelo sistema A. Fonte: o próprio autor.	52
3.12	Gráfico decrescente dos autovalores de cada classe obtidos a partir da base de dados reamostrada pelo sistema A. Fonte: o próprio autor.	52
3.13	Valores residuais e limiares de classificação para a base reamostrada pelo sistema A. Fonte: o próprio autor.	53
3.14	Ruído no sistema A.	53
3.15	Esquema elétrico do sistema B. Fonte: o próprio autor.	54
3.16	Base de dados reamostrada pelo sistema B. Fonte: o próprio autor.	54
3.17	Resultado da amostragem ao utilizar um <i>ESP32</i> para realizar a conversão AD e a comunicação soquete ao mesmo tempo.	55

3.18	Efeito da reamostragem do áudio da moto 0 pelo sistema B.	55
3.19	Áudio amostrado quando há erro na comunicação <i>UART</i> . Fonte: o próprio autor.	56
3.20	Módulo do espectro de frequências médio e desvio padrão para as assinaturas sonoras da base de dados reamostrada pelo sistema B. Fonte: o próprio autor. .	56
3.21	Gráfico dos autovalores de cada classe obtidos a partir da base de dados reamostrada pelo sistema B. Fonte: o próprio autor.	57
3.22	Valores residuais e limiares de classificação para a base reamostrada pelo sistema B. Fonte: o próprio autor.	57
5.1	Conversor AD <i>ADS1115</i> . Fonte: o próprio autor.	60
5.2	<i>ESP32-CAM</i> . Fonte: o próprio autor.	61

1 Introdução

Este trabalho lida com a classificação de automóveis, que possui relação com a detecção de automóveis, importante para gerenciamento de tráfego através de, por exemplo, contagem de veículos e otimização de tempos de sinais de tráfego [5].

Assim como técnicas baseadas em processamento de imagem, as técnicas que utilizam o som são alternativas a métodos invasivos como o laço indutivo (Figura 1.1).

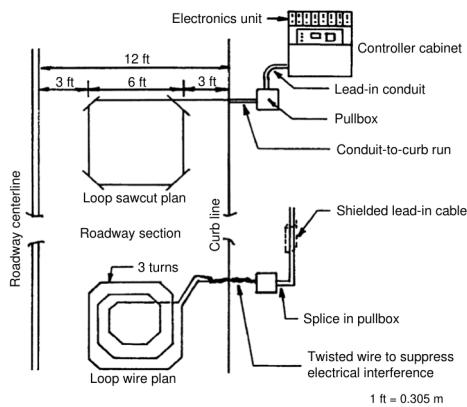


Figura 1.1: Diagrama de funcionamento de um laço indutivo. Fonte: [9]

Sob condições similares, tipos de automóveis parecidos emitem sons parecidos, o que indica uma assinatura sonora específica para cada tipo de veículo. Além disso, o processamento de áudio oferece ainda a vantagem de menor custo computacional quando comparado ao processamento de imagem.[5].

O trabalho de desenvolvimento do sistema classificador de automóveis aqui relatado pode ser dividido em duas áreas principais: reconhecimento de padrões e sistemas embarcados. Uma breve introdução sobre estas pode ser encontrada nas duas seções subsequentes.

1.1 Reconhecimento de padrões

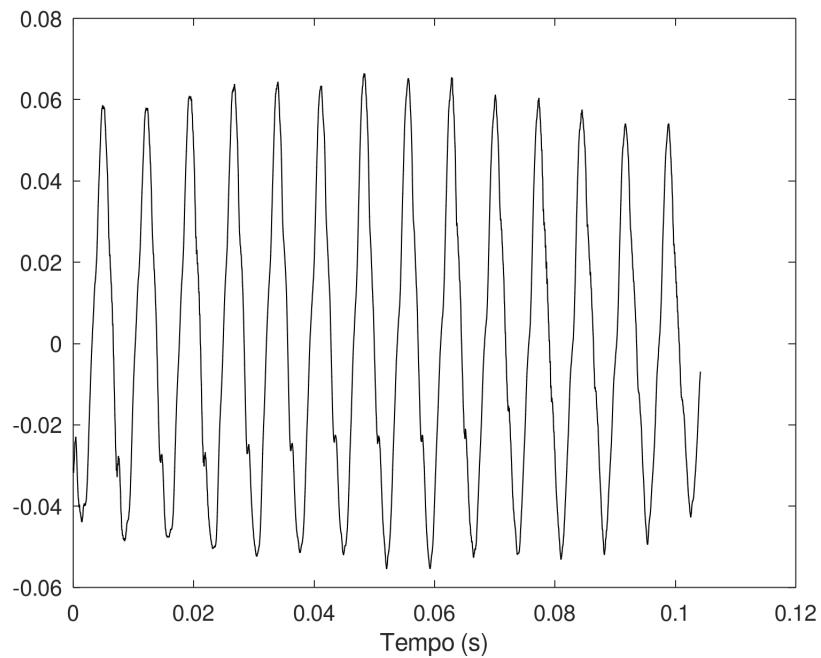
Reconhecimento de padrões é um campo do conhecimento que lida com a tarefa de processar dados organizados em vetores para extrair características e usá-las em problemas de classificação. Seja para diagnosticar doenças, reconhecer faces, identificar falantes ou muitas outras aplicações.

Encontram-se nesta seção alguns conceitos deste campo multidisciplinar, que serão utilizados no trabalho.

1.1.1 Sinais

Um sinal é qualquer coisa que contenha informação, geralmente, como no caso deste texto, sinais são alguma grandeza medida ao longo tempo. Por exemplo, a voz de uma pessoa pode ser capturada pelo microfone do celular e armazenada em sua memória. Neste caso, a vibração do ar provocada pela fala ao longo do tempo seria um sinal (Figura 1.2).

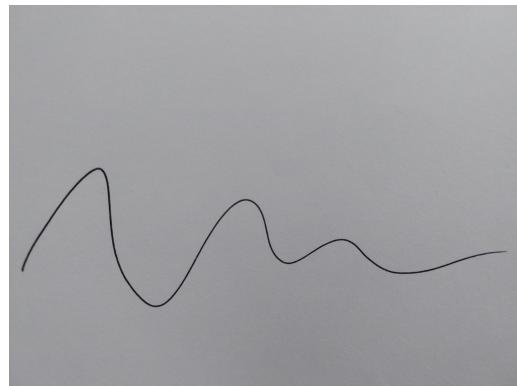
Figura 1.2: Trecho de voz humana gravada por um *notebook*. Fonte: o próprio autor.



1.1.2 Sinais Contínuos e Discretos

Uma curva traçada a caneta pode ser considerada um sinal contínuo (Figura 1.3). Teoricamente, existem infinitos valores entre quaisquer dois pontos de um sinal contínuo, pois a escala pode ser feita infinitamente maior.

Figura 1.3: Curva traçada a caneta, ilustração de um sinal contínuo. Fonte: o próprio autor.



Isso faz com que todo sinal armazenado num computador seja um sinal discreto, pois não há memória infinita. Logo, toda vez que vemos a ilustração de um sinal contínuo numa tela de computador ou celular, trata-se apenas de uma ilusão causada por alta resolução.

Diferentemente de sinais contínuos, sinais discretos são aqueles definidos em instantes discretos no tempo, seja pela natureza do fenômeno que o produz, seja por conta de algum processo de discretização.

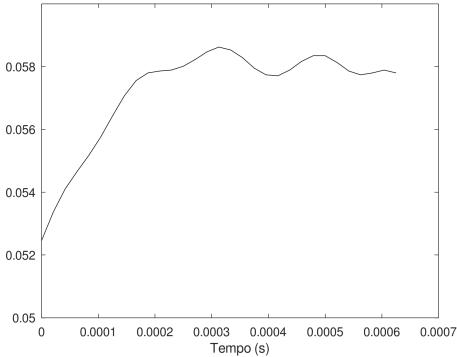
A conta de energia elétrica ao longo do tempo pode ser considerada um sinal discreto, uma vez que ela assume apenas um valor a cada mês. Não existe uma transição contínua entre o valor de um mês e o do próximo.

Quando um sinal contínuo é armazenado em uma memória, necessariamente passa por um processo de discretização. Os infinitos valores tornam-se então coleções finitas¹ de valores ou vetores (Figura 1.4).

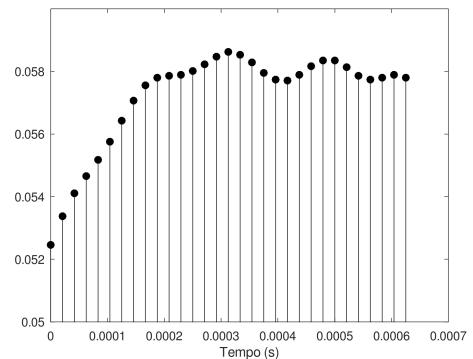
¹Sinais discretos também podem ser conjuntos infinitos de valores ao longo do tempo, mas isso não acontece numa memória de computador.

Figura 1.4: Ilustração da discretização de um sinal de voz.

(a) Ilustração de um sinal contínuo de fala. Fonte: o próprio autor.



(b) Sinal de fala discretizado. Fonte: o próprio autor.



O trecho da Figura 1.4b pode ser escrito como um vetor

$$s = [0,0525; 0,0534; \dots; 0,0578] \quad (1)$$

Equações matemáticas que descrevem ou transformam sinais contínuos precisam ser também adaptadas para tratar seus pares discretos e este aspecto será abordado ainda na introdução deste trabalho.

1.1.3 Sinais Analógicos e Digitais

Um sinal analógico é aquele cuja amplitude pode assumir qualquer valor pertencente a um domínio contínuo de valores.

Assim como um computador não pode registrar infinitos valores ao longo do tempo, ele também não está preparado para armazenar um espectro infinito de valores de amplitude.

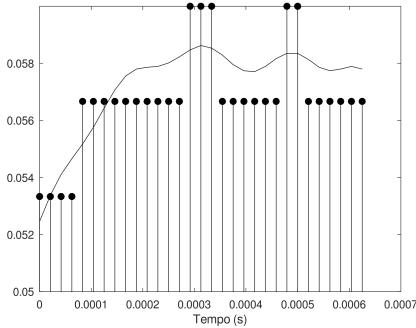
Antes de ser registrado numa memória digital, um sinal analógico passa pelo processo de discretização em amplitude, também conhecido como quantização, onde cada medição de amplitude é aproximada por um dos valores possíveis para aquele sistema digital.

Quanto maior a quantidade de valores possíveis, melhor é a aproximação do sinal original. A Figura 1.5a mostra o processo de digitalização² de um sinal onde há apenas 4 valores possíveis entre 0,05 e 0,06 igualmente espaçados, enquanto na Figura 1.5b, havia 16 valores possíveis.

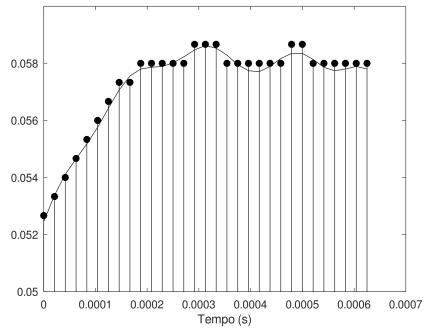
²Sinais digitais podem, por definição, ser também contínuos no tempo, mas não é esse o caso aqui retratado.

Figura 1.5: Processo de digitalização e discretização de um sinal analógico contínuo com diferentes números de valores de amplitude possíveis.

(a) Digitalização com 4 valores de amplitude possíveis. Fonte: o próprio autor.

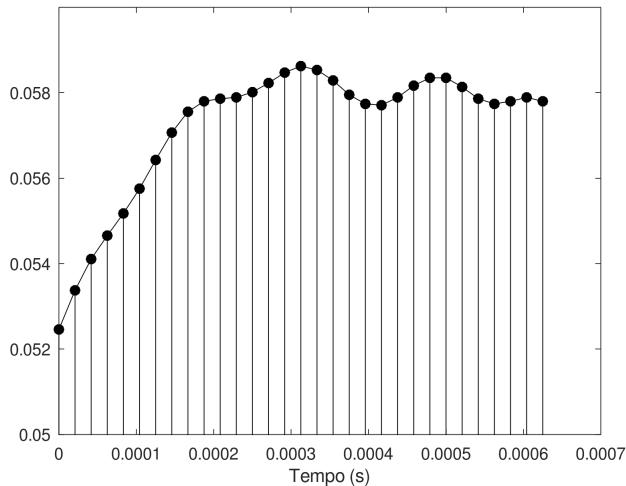


(b) Digitalização com 16 valores de amplitude possíveis. Fonte: o próprio autor.



Sinais de áudio digitais podem, por exemplo, ter sido digitalizados com 65536 valores possíveis (Figura 1.6).

Figura 1.6: Processo de digitalização e discretização de um sinal analógico contínuo com 65536 valores de amplitude possíveis.



Esta variável (número de valores de amplitude possíveis) está associada à resolução de um Conversor Analógico para Digital (Seção 1.2.5), uma das grandezas usadas para caracterizar suas conversões.

Uma vez discretizados e digitalizados, a linguagem vetorial da álgebra linear e o poder computacional de sistemas embarcados se tornam úteis na representação e operações com os sinais.

1.1.4 Espaços vetoriais, subespaços e projeções

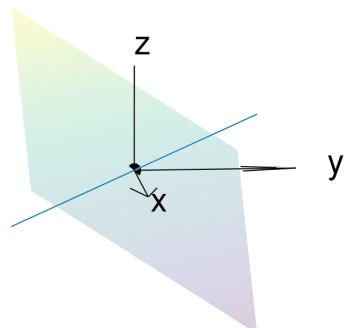
Assim como um conjunto de três números reais pode ser visto como um vetor em um espaço de três dimensões, é possível ter um sinal de áudio com N amostras interpretado como um vetor em um espaço de N dimensões, mesmo que a visualização não seja possível.

Isso é relevante pois as ferramentas da álgebra linear estão disponíveis tanto em espaços de três dimensões quanto de infinitas. Algumas dessas ferramentas que este trabalho utiliza são os conceitos de subespaços e projeções.

Dado um espaço vetorial qualquer, um subespaço é um conjunto não vazio que satisfaz requerimentos de um espaço vetorial, ou seja combinações lineares de seus elementos, continuam no subespaço [11].

Por exemplo, no caso de um espaço tridimensional \mathbb{R}^3 (ou seja, vetores reais com 3 componentes), são considerados subespaços os planos ou retas que cruzam a origem (Figura 1.7).

Figura 1.7: Exemplo de dois subespaços em \mathbb{R}^3 . Uma reta e um plano que cruzam a origem. Fonte: o próprio autor.



Neste trabalho, será importante definir uma maneira de medir a distância entre um vetor qualquer e um dado subespaço. Isso pode ser feito através de projeções.

Encontrar a projeção de um vetor v num subespaço b significa encontrar o ponto p no subespaço que está mais próximo desse vetor. Esse problema é análogo à minimização de erros quadrados na álgebra linear e possui uma solução bem conhecida, utilizando a matriz de projeção:

$$P = A(A^T A)^{-1} A^T \quad (2)$$

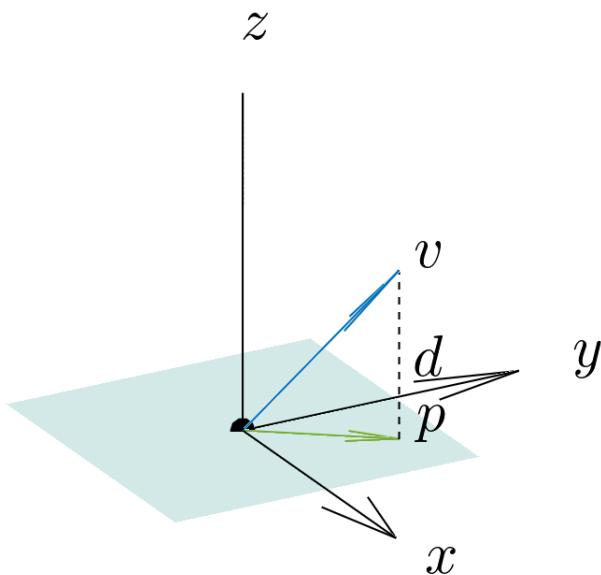
, sendo A uma matriz de colunas linearmente independentes que constituem o subespaço. Estas precisam ser linearmente independentes para garantir a invertibilidade de $(A^T A)$.

E assim, a projeção desejada é

$$p = Pv \quad (3)$$

A diferença entre o vetor v e sua projeção p no subespaço pode ser usada como uma medida de distância d entre o vetor e o subespaço (Figura 1.8).

Figura 1.8: Projeção de um vetor v num subespaço e a distância d até o mesmo subespaço. Fonte: o próprio autor.



Pode-se dizer que essa diferença mede o quanto o vetor v "parece" com vetores que se encontram no plano xy . E essa é a importância deste tópico. Pois ao encontrar um subespaço que representa um conjunto de dados (ou um conjunto de vetores), é possível medir o quanto outros dados desconhecidos "se parecem" com esse conjunto.

1.1.5 Autovetores e autovalores

Os autovetores e autovalores surgem do seguinte problema na álgebra linear:

$$Ax = \lambda x \quad (4)$$

, ou seja, dada uma matriz quadrada A de tamanho n , quais vetores x e escalares λ satisfazem a igualdade.

As incógnitas x e λ são respectivamente os autovetores e autovalores da matriz A e podem ser encontrados ao seguir os passos:

- Computar o determinante de $A - \lambda I$, sendo I a matriz identidade;
- Encontrar as n raízes do polinômio resultante. Estas raízes são os autovalores;
- Para cada autovalor, resolver $(A - \lambda I)x$, encontrando o autovetor correspondente.

Caso a matriz A seja simétrica, os autovetores serão mutuamente ortogonais.

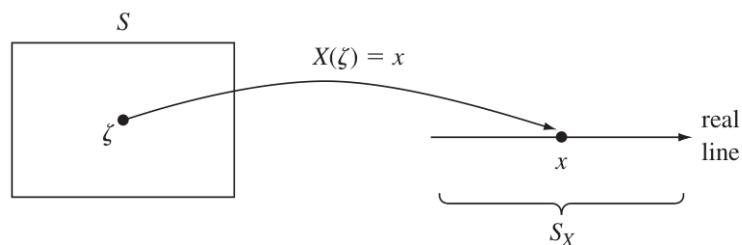
Estes vetores e escalares especiais estão relacionados à solução de equações diferenciais ordinárias e à resposta natural de sistemas dinâmicos, além de serem úteis na diagonalização de matrizes simétricas.

1.1.6 Sinais aleatórios e covariância

Algumas das técnicas usadas neste trabalho são possíveis pela modelagem de sinais de áudio discretos como vetores de variáveis aleatórias.

Uma variável aleatória é uma função que associa eventos aleatórios a valores (Figura 1.9).

Figura 1.9: Ilustração de variável aleatória como uma função X . Fonte: [8].



Logo, podemos modelar um sinal de áudio s como

$$s = [X_1, X_2, \dots, X_N]^T \quad (5)$$

, onde X_n é a enésima variável aleatória que representa a enésima amostra do sinal. A transposta T , é aplicada pela convenção de expressar vetores de variáveis aleatórias como vetores coluna.

Mesmo que o sinal seja conhecido *a priori*, como por exemplo uma música tocada várias vezes numa estação de rádio, ainda faz sentido modelar o sinal desse áudio gravado por um microfone como variável aleatória pois ele estará sempre sujeito ao efeito de outras variáveis aleatórias, como ruídos eletromagnéticos.

Uma vez interpretado como vetor de variáveis aleatórias, o sinal pode ser analisado por poderosas ferramentas probabilísticas que servem para caracterizá-lo.

A covariância, por exemplo, é uma forma de medir a correlação entre duas variáveis aleatórias X e Y . E é definida como

$$COV(X, Y) = E[(X - E[X])(Y - E[Y])] \quad (6)$$

, sendo $E[X]$ o valor esperado da variável aleatória X [8].

A covariância entre duas variáveis aleatórias é igual a 0 quando elas são descorrelacionadas.

É possível usar essa medida para caracterizar também vetores de variáveis aleatórias, através da matriz de covariância:

$$K_X = \begin{bmatrix} E[(X_1 - E[X_1])^2] & E[(X_1 - E[X_1])(X_2 - E[X_2])] & \dots & E[(X_1 - E[X_1])(X_n - E[X_n])] \\ E[(X_2 - E[X_2])^2] & E[(X_2 - E[X_2])^2] & \dots & E[(X_2 - E[X_2])(X_n - E[X_n])] \\ \vdots & \vdots & \dots & \vdots \\ E[(X_n - E[X_n])^2] & E[(X_n - E[X_n])(X_2 - E[X_2])] & \dots & E[(X_n - E[X_n])^2] \end{bmatrix} \quad (7)$$

, um matriz simétrica de valores reais, cuja diagonal é composta pelas variâncias de X_k , definidas como

$$VAR(X_k) = E[(X_k - E[X_k])^2] \quad (8)$$

Caso os elementos do vetor aleatório sejam descorrelacionados, a matriz covariância será uma matriz diagonal.

1.1.7 Análise de Componentes Principais (PCA)

Uma vez construída a matriz de covariância de um conjunto de vetores aleatórios, é possível realizar algumas operações para caracterizar ainda melhor esses dados.

A expansão de Karhunen-Loève (expansão *KL*) surge a partir da diagonalização da matriz de covariância K_x [8]. Dados os k autovetores e_k da matriz de covariância (e seus respectivos autovalores λ_k), é demonstrável que a seguinte transformação linear leva à matriz diagonal K_y

$$K_y = P^T K_x P = \Lambda \quad (9)$$

, onde

P : Matriz de colunas e_k ;

Λ : Matriz diagonal de autovalores λ_k .

A matriz resultante K_y é também uma matriz de covariância do vetor coluna Y de variáveis aleatórias descorrelacionadas Y_k . Além disso, é possível expressar o vetor X de variáveis aleatórias X_k original como uma soma ponderada dos autovetores e_k , pela equação

$$X = PY = [e_1, e_2, \dots, e_k] \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_k \end{bmatrix} \quad (10)$$

Ou seja, essa expansão pode ser considerada uma forma de extrair características (autovetores e_k) que descrevem o conjunto de dados inicial. Uma das vantagens dessa forma é que os autovetores, tendo sido obtidos de uma matriz simétrica, são mutuamente ortogonais, evitando assim a redundância de informações, já que nenhum deles pode ser expresso como combinação linear dos demais.

Além disso, caso algum dos autovalores λ_k seja igual a 0, isso faz com que $VAR[Y_k] = 0$ e então $Y_k = 0$. De modo que o correspondente e_k pode ser desprezado, reduzindo a dimensionalidade do problema.

Pode também ser demonstrado que a magnitude do autovalor pode ser considerada uma medida da importância do seu autovetor correspondente na representação dos dados. Esse fato vem da minimização de erros quadrados, quando aproximamos o vetor de dados por sua projeção num subespaço gerado por autovetores.

Sendo assim, é possível escolher alguns autovetores mais importantes como base do subespaço que melhor representa um conjunto de dados. Esses "componentes principais"[\[12\]](#), da Análise de Componentes Principais é, portanto, fundamentada pela expansão KL .

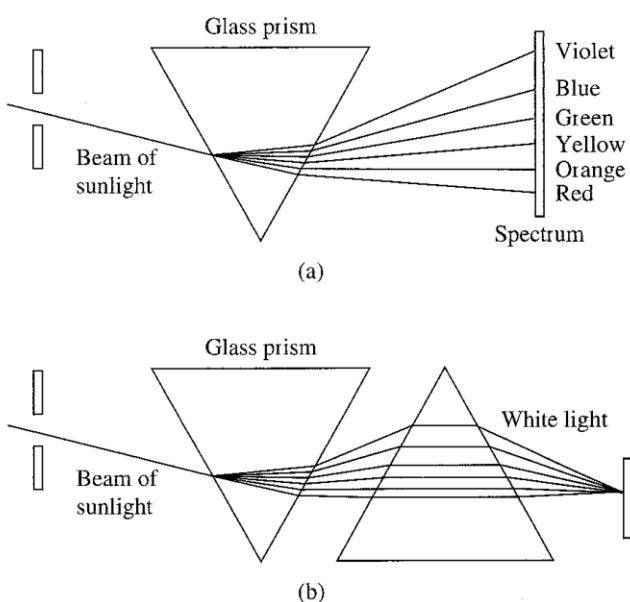
1.1.8 Transformada de Fourier

Outra ferramenta útil na extração de características é a transformada de Fourier. Esta transformação linear serve para decompor sinais em exponenciais complexas.

Estas exponenciais complexas, por sua vez, representam sinais oscilantes, como senóides. Isso faz sentido físico, uma vez que muitas vezes um sinal obtido no mundo real resulta de diferentes fontes com comportamento dinâmico oscilatório.

Assim a luz branca pode ser decomposta em luzes de frequências diferentes (cores diferentes) ao passar por um prisma (Figura 1.10), o sinal de áudio de uma orquestra, por exemplo, pode ser visto como a composição de oscilações do ar em diversas frequências (notas musicais) provocadas por vários instrumentos diferentes.

Figura 1.10: Exemplo de interpretação física da transformada de Fourier. A decomposição da luz branca em ondas eletromagnéticas de diferentes frequências. Fonte: [\[10\]](#).



Para um sinal contínuo descrito pela função $x(t)$, a transformada de Fourier é dada por

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft} dt \quad (11)$$

, onde f é a nova variável representando as frequências das exponenciais complexas.

De modo geral, a variável da função original é o tempo. Ao realizar a transformada de Fourier, é dito que a função está sendo levada do domínio do tempo ao domínio da frequência.

Neste trabalho, a base da dados é constituída de sinais discretos no domínio do tempo, portanto é preciso utilizar a versão discreta da transformada de Fourier, a *DFT*, para chegar ao domínio da frequência.

1.1.9 Transformada de Fourier discreta (*DFT*)

A Transformada Discreta de Fourier, do inglês, *Discrete Fourier Transform (DFT)*, é uma ferramenta que, juntamente com seu par, a *Inverse Discrete Fourier Transform (IDFT)*, permite a análise e síntese de sinais discretos (neste caso, digitais) em exponenciais complexas componentes.

Discorreremos apenas sobre a DFT, uma vez que não é de interesse particular a síntese do sinal, ou seja, a IDFT. Pois para este trabalho e as técnicas nele utilizadas, o intuito é apenas de extrair características do sinal no domínio da frequência e usá-las para classificação. As amostras estão dispostas no domínio do tempo *a priori*.

A construção dessa ferramenta parte da amostragem do espectro de um sinal, gerado pela Transformada de Fourier[10]. Após manipulação de equações, é provado ser possível reconstruir o sinal e seu espectro originais a partir dessas amostras, além de serem estabelecidas condições para tal.

A *DFT* é, então, a equação que permite transformar um sinal discreto $x(n)$, com número L de pontos em um espectro de frequências $X(k)$, também discreto. Ela pode ser escrita como

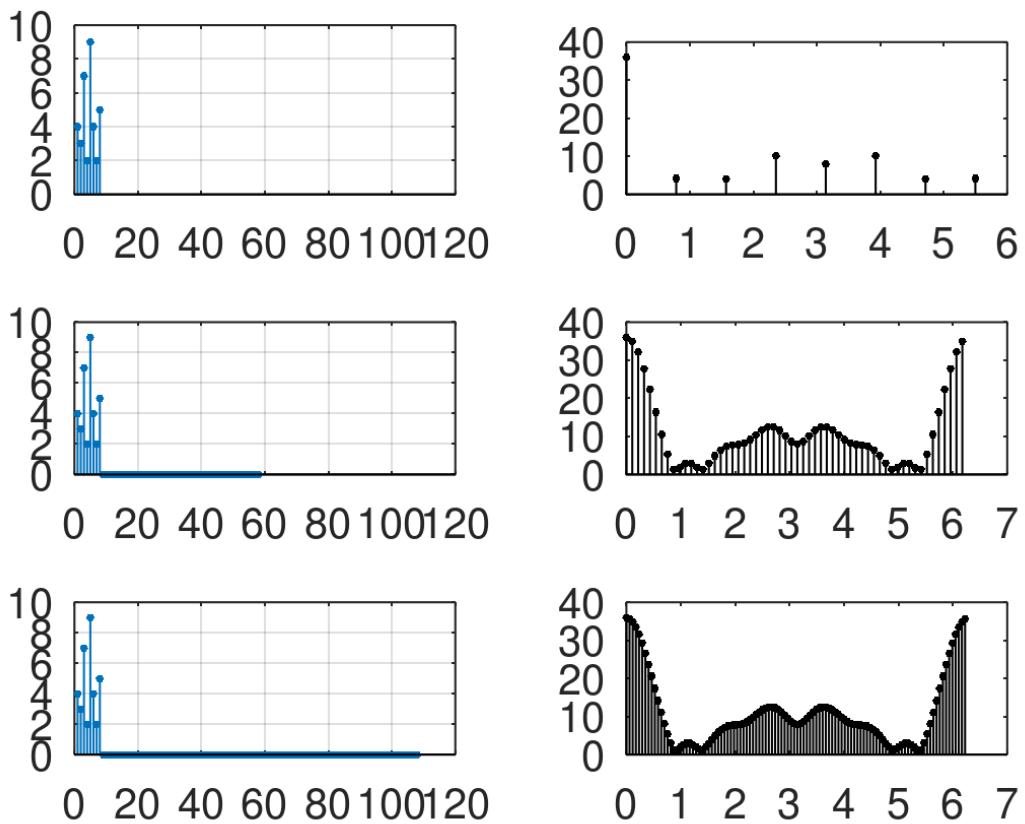
$$DFT : X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \quad (12)$$

, sendo N o número de pontos de $X(k)$.

Para sinais finitos -e se tratando de memórias de computador, sempre teremos sinais finitos- a transformada retorna uma representação única do sinal e basta apenas que façamos $N \geq L$ para que o fenômeno de *aliasing* não se manifeste no sinal reconstruído³.

Para que N possa ser maior que L , é preciso adicionar zeros ao final de x , uma manobra chamada *zero padding*, ou preenchimento com zeros (Figura 1.11). Esse preenchimento é usado quando se deseja melhorar a resolução do espectro do sinal e em contrapartida apenas torna a sua computação mais difícil.

Figura 1.11: Exemplo do efeito do preenchimento com zeros na construção do espectro do sinal. Fonte: o próprio autor.



Para uma notação mais simples e programação mais eficiente, é de interesse introduzir a interpretação matricial da *DFT*, encontrada na próxima seção.

³O efeito de *aliasing* é uma distorção intrínseca à DFT para sinais infinitos, mas ocorre em sinais finitos apenas quando $N < L$

1.1.10 Interpretação matricial da DFT

A transformada de Fourier discreta pode ser vista como uma transformação linear. Podemos então definir uma matriz, que multiplicada pelo sinal no domínio do tempo, leva ao espectro no domínio da frequência.

Definindo $m_N = e^{\frac{-j2\pi}{N}}$, a matriz *DFT* de tamanho N pode ser construída como

$$M_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & m_N & m_N^2 & \dots & m_N^{N-1} \\ 1 & m_N^2 & m_N^4 & \dots & m_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & m_N^{N-1} & m_N^{2(N-1)} & \dots & m_N^{(N-1)(N-1)} \end{bmatrix} \quad (13)$$

De modo que dado um vetor

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \quad (14)$$

, sua transformada X pode ser escrita sucintamente como

$$X = M_N x \quad (15)$$

1.2 Sistemas embarcados

Sistemas embarcados são sistemas computadores embutidos em algum outro sistema, sem necessidade de interação direta com o usuário. Geralmente, esses sistemas são relacionados a microcontroladores, pequenos computadores programáveis munidos de microprocessadores.

Através de estruturas *ciber-físicas*, ou seja combinando poder computacional e interfaces físicas, esses sistemas interagem com o ambiente para desempenhar alguma função.

Existem algumas definições de requerimentos para o projeto de um sistema embarcado, como confiabilidade, segurança, disponibilidade e outros, mas neste trabalho o foco reside na

implementação do algoritmo de reconhecimento de padrões, de modo que basta a definição mais básica de embarcados.

Para implementar o referido sistema, foram utilizadas, principalmente, técnicas de comunicação. Estas serão introduzidas nesta seção.

1.2.1 Internet e protocolos de comunicação

Uma das decisões de projeto desde trabalho foi a de implementar o sistema de coleta de amostras de áudio num sistema embarcado e realizar o processamento dos dados num computador pessoal. Para o envio de amostras em tempo real, foi utilizada comunicação via *Internet*, a rede de comunicação interligada mundialmente.

Para que dois dispositivos quaisquer se comuniquem nessa e em outras redes, é necessário atender a uma série de protocolos de comunicação.

Esses protocolos de comunicação não são particularidades da *Internet*, sempre que dispositivos se comunicam, há algum entendimento mútuo de quais passos essa comunicação deve seguir, ou não haveria como garantir a troca de informações corretas.

Assim como duas pessoas ao se comunicar pessoalmente precisam compartilhar protocolos: estarem próximas uma da outra, saber a hora de cada um falar e utilizar palavras ou linguagem não verbal que ambos entendam. Computadores e outros dispositivos precisam de protocolos de comunicação.

No caso da comunicação entre dispositivos eletrônicos, é preciso saber: quais serão os canais de transmissão (fios, ar), quando cada dispositivo recebe ou envia sinais elétricos, como as mensagens estão codificadas (como traduzir eletricidade em informação), como saber que não houve erro no envio, como proteger mensagens sigilosas, como o usuário vai interagir com as mensagens e muitos outros.

No caso da *Internet*, uma rede interconectada de milhares de outras redes, o número e complexidade de protocolos é grande. Uma estrutura organizacional que ajuda a desenvolver e entender esses protocolos é o conjunto de protocolos *TCP/IP* (Figura 1.12), que divide a comunicação em camadas:

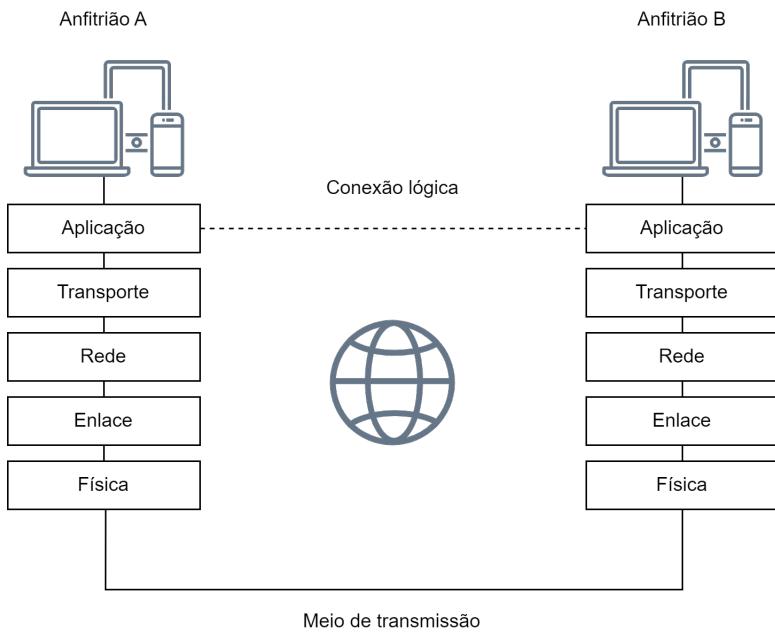


Figura 1.12: Ilustração da estrutura *TCP/IP*. Fonte: o próprio autor.

Sabendo que os dispositivos finais que se comunicam nesta rede são conhecidos como anfitriões, segue uma descrição breve das camadas do conjunto de protocolos [3]:

- **Aplicação:** é a camada com a aplicação final de destino e origem das mensagens, rodando no anfitrião que se comunica. Apesar do fato de que a comunicação passa por todas as camadas, é possível dizer que há uma conexão lógica entre as camadas de aplicação dos anfitriões comunicantes. É aqui que estão protocolos como: *HTTP (Hypertext Transfer Protocol)*, que serve para acessar dados a partir de texto estruturado, *SMTP (Simple Mail Transfer Protocol)*, um protocolo para usar *e-mails*, *DHCP*, que fornece o endereço de *IP* (um endereço único na rede inteira, que serve para identificar o anfitrião), entre outros.
- **Transporte:** serve para definir alguns detalhes do transporte das mensagens. Os principais protocolos dessa camada são o *TCP (Transmission Control Protocol)* e o *UDP (User Datagram protocol)*. Sucintamente, o primeiro garante um transporte mais confiável, enquanto o segundo troca essa confiabilidade por mais velocidade.
- **Rede:** nesta camada encontra-se o protocolo *IP (Internet Protocol)*, talvez o protocolo mais importante para a *Internet*. Além de envolver os dados com diversas informações úteis à comunicação na rede (criação do datagrama), similar à como se envolve uma carta no envelope, com remetente e destinatário, este protocolo fornece vários serviços,

roteamento, como execução de lógicas para escolher as melhores rotas dentro da rede ou controle de erros e congestionamentos.

- **Enlace:** é a camada onde encontram-se as chamadas *LAN (Local Area Network)*, que são redes que conectam anfitriões localmente e *WAN (Wide Area Network)*, redes que conectam roteadores (conectores de redes), *modems* (conexões que transformam os dados para transmissão através de algum meio) ou *switches* (conectores de dispositivos), podendo se estender globalmente. Não há protocolos específicos, empresas podem criar seus protocolos ou usar padrões para pegar o datagrama da camada de rede e entregá-la à camada física, podendo oferecer serviços como detecção de erros.
- **Física:** transforma a informação digital em sinais físicos e vice-versa. Esses sinais devem ser próprios para a transmissão através do meio de transmissão, como sinais eletromagnéticos de alta frequência através do ar ou vácuo.

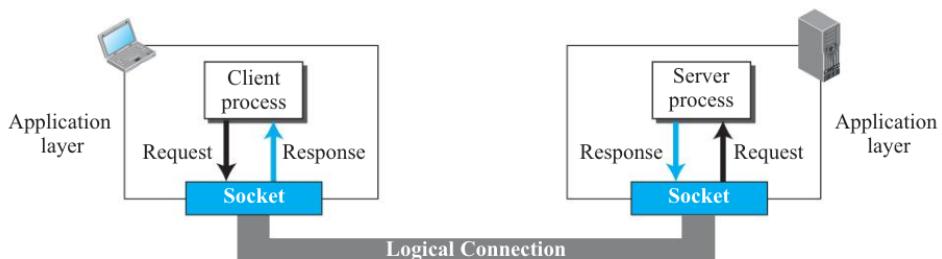
Essa divisão em camadas e padronização da comunicação permite que entreguemos a execução da maioria dos protocolos a processos automatizados. De modo que neste trabalho, o desenvolvimento é realizado com foco na camada de aplicação, onde podemos assumir a conexão lógica na transmissão de dados de um anfitrião ao outro.

1.2.2 Interface *socket*

Interface de soquetes ou *sockets* é uma interface de programação de aplicação. Ou seja, dois anfitriões precisam se comunicar através da internet e adotam uma maneira de interfacear as informações na camada de aplicação.

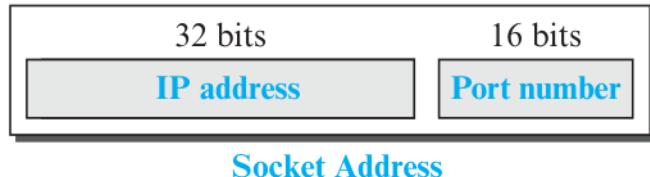
Os soquetes são uma forma de estruturar os dados, na qual é possível escrever ou ler dados nos soquetes, onde a leitura é condicionada ao envio de dados pelo outro anfitrião (Figura 1.13).

Figura 1.13: Funcionamento básico da interface *socket*. Fonte: [3].



Para que esses soquetes sejam acessados pelos anfitriões, é preciso definir endereços soquete. Cada anfitrião deve possuir seu endereço soquete, onde dados são "depositados". Este é composto pelo *IP* do anfitrião e um número de porta (Figura 1.14) para diferenciá-lo de outras interfaces e aplicações.

Figura 1.14: Ilustração da formatação do endereço *socket*. Fonte: [3]



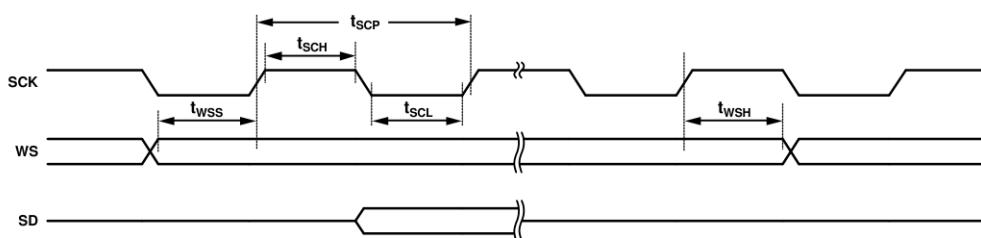
O sistema classificador implementado utiliza as linguagens de programação *Python* e *C* para realizar protocolos de comunicação e essa interface. Para mais detalhes sobre o uso da interface na prática, ler a Seção 2.3.2.

1.2.3 Protocolo I^2S

O protocolo I^2S é uma interface de comunicação serial. O microfone utilizado neste trabalho fornece as amostras codificadas em bits, quando estimulado por um sinal de clock (Figura 1.15).

Basicamente, esse protocolo divide os bits de dados em canais esquerdo e direito, para permitir a transmissão de áudio no formato estéreo ou mono.

Figura 1.15: Ilustração da transmissão de dados no protocolo I^2S . Fonte: [6]



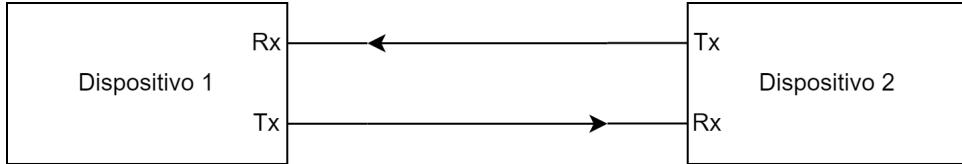
A seleção de canais é realizada pelo sinal *WS* ou *word select*. Para mais detalhes, consultar a Seção 2.3.1.

1.2.4 Protocolo *UART*

O protocolo de comunicação Transmissor/Receptor Assíncrono Universal ou *Universal Asynchronous Receiver/Transmitter (UART)* permite a comunicação serial bidirecional entre dois

dispositivos mediante apenas dois pontos de conexão (Rx e Tx)⁴ e de maneira assíncrona, ou seja, sem a necessidade de um sinal de *clock* (Figura 1.16).

Figura 1.16: Ilustração da comunicação *UART*. Fonte: o próprio autor.



Do ponto de vista de um dispositivo, a informação é recebida pelo seu terminal Rx e enviada pelo seu terminal Tx.

As liberdades e caráter universal desse protocolo fazem necessária, no entanto, que algumas variáveis sejam conhecidas pelos dois dispositivos, como por exemplo a taxa de transferência de *bits* e a largura de *bits* das palavras.

1.2.5 Conversão AD

É o processo de converter grandezas analógicas em números binários através de dispositivos conhecidos como conversores analógico para digital ou *analog to digital converters (ADC)* [1].

Quanto mais dígitos (*bits*) tiver esse número binário, maior será o número de valores possíveis para o sinal convertido, aumentando a resolução da conversão. Para n dígitos, a conversão possui resolução de n *bits*.

A taxa de conversão de um conversor AD, é definida como a maior taxa de realização de conversões ao longo do tempo que pode ser atingida. Neste trabalho, essa taxa está diretamente associada à taxa de amostragem dos áudios. Uma vez que cada conversão representa uma amostra do sinal, um conversor que realiza 20.000 conversões por segundo leva a uma taxa de amostragem de 20kHz, dado que não haja nenhum gargalo ao transmitir essas amostras entre dispositivos.

⁴é possível utilizar mais conexões para garantir funcionalidades adicionais

2 Metodologia

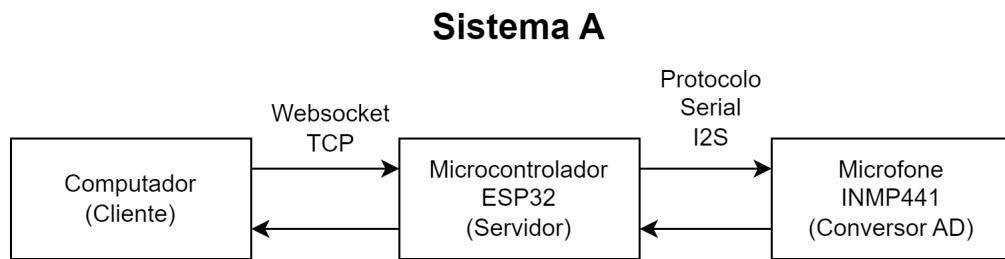
2.1 Visão geral

Ao longo do desenvolvimento do trabalho, são construídos dois sistemas embarcados para realizar a mesma tarefa de classificação. São esses os sistemas A e B.

O sistema classificador A ([2.1](#)) é constituído de três componentes principais:

- Computador (Cliente)
- Microcontrolador ESP32 (Servidor)
- Microfone INMP441 (Transdutor e Conversor AD)

Figura 2.1: Diagrama do funcionamento do sistema classificador A. Fonte: o próprio autor.



O funcionamento é controlado pelo usuário através do cliente implementado em Python no computador.

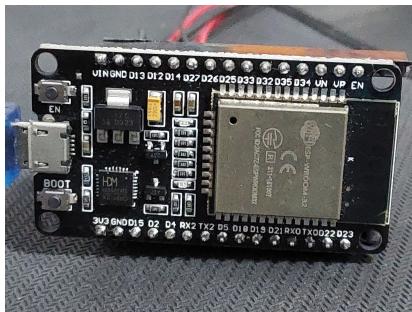
Essa foi uma decisão de projeto que pretendia manter o sistema embarcado no microcontrolador o mais simples possível, enquanto utilizava as vantagens de poder de processamento e memória de um computador pessoal.

O processamento dos dados é todo realizado no computador, mas nada impede o desenvolvimento de uma versão em que o processamento seja realizado no microcontrolador, para respostas em tempo real.

Na versão aqui apresentada, o usuário escolhe o número de amostras a serem coletadas e o microcontrolador ([Figura 2.2a](#)) se encarrega de coletá-las através do microfone e conversor AD ([Figura 2.2b](#)) e enviá-las utilizando o protocolo websocket TCP.

Figura 2.2: Principais componentes utilizados no classificador.

(a) Microcontrolador ESP32. Fonte: o próprio autor.



(b) Microfone INMP441. Fonte: o próprio autor.



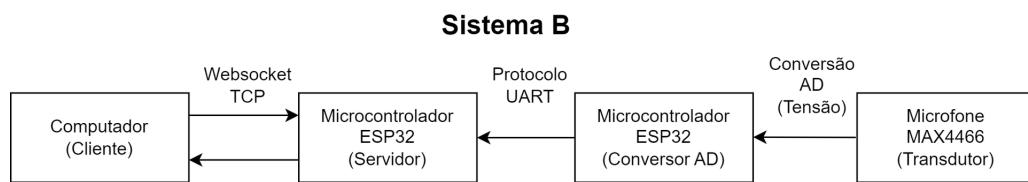
O processamento de áudio e classificação são então realizados no mesmo algoritmo em *python* no computador.

Apesar de que ainda pode ser útil em trabalhos futuros, este sistema se revelou inadequado para a técnica de classificação utilizada, por conta da distribuição espectral do ruído em suas amostras (Seção 3.2.3).

Por conta desse problema, foi projetado o sistema B, que realizou a amostragem de maneira mais satisfatória.

O sistema B funciona de forma análoga, com a diferença de que o trabalho de conversão AD é realizado também por um microcontrolador *ESP32*, a partir de valores de tensão produzidos pelo microfone *MAX4466* (Figura 2.5) e esses dados são enviados ao servidor soquete por comunicação *UART* (Figura 2.3).

Figura 2.3: Diagrama do funcionamento do sistema classificador B. Fonte: o próprio autor.



Apesar de ser incapaz de atingir a mesma frequência de amostragem do sistema A, o sistema B apresenta melhores resultados de classificação para a mesma técnica utilizada (Seção 3.3.3).

Figura 2.4: Microfone utilizado no sistema B. Fonte: o próprio autor.



Figura 2.5: Microfone utilizado no sistema B. Fonte: o próprio autor.

2.2 Estrutura de programação

Em ambos os sistemas, para programar o microcontrolador ESP32 foi utilizada a estrutura de desenvolvimento de internet das coisas da *Espressif* ou *Espressif IoT Development Framework* (ESP-IDF), oficial do fabricante *Espressif* (Figura 2.6a). Essa estrutura contém vários módulos responsáveis por controlar o funcionamento do microcontrolador atendendo a diversas aplicações, como conversão analógico digital, comunicação serial e conexão *wi-fi*.

Figura 2.6: Imagens relacionadas à programação do ESP32.

(a) Interface do guia de programação online da *ESP-IDF*.

Fonte: [2]

(b) Logomarca do *FreeRTOS*. Fonte: [4]



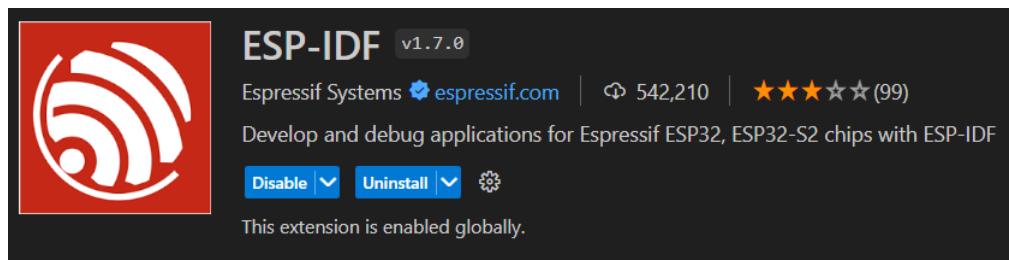
A *ESP-IDF* é baseada no sistema operacional *FreeRTOS* (Figura 2.6b), um sistema operacional em tempo real com licença de código aberto *MIT*, portanto ela conta com diversas bibliotecas auxiliares que lidam, por exemplo, com interrupções ou tarefas e com gerenciamento de filas (*buffers*).

Através dessa estrutura de desenvolvimento, é possível configurar o funcionamento do microcontrolador (por exemplo, qual frequência do processador utilizar), compilar o código, car-

regar o programa no dispositivo e monitorar o funcionamento em tempo real através de um monitor serial. Tudo isso pode ser feito através das instruções dadas no guia de programação da *ESP-IDF*.

Neste trabalho, foi utilizada a extensão oficial da *Espressif* para o editor *Visual Studio Code* ou *VS Code* (Figura 2.7). Ela facilita a criação de novos projetos, utilização de códigos exemplo, configuração, compilação e tudo mais necessário neste caso.

Figura 2.7: Extensão oficial da *ESP-IDF* para *VS Code*. Fonte: o próprio autor.



2.3 Programação do Sistema A

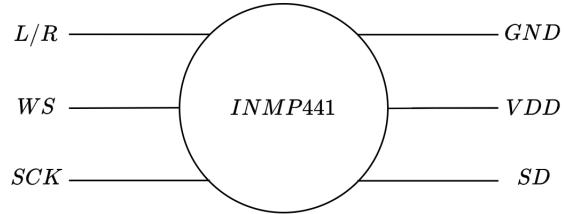
Os procedimentos para programar o sistema A são apresentados nesta seção.

2.3.1 Coleta de amostras (Protocolo I^2S)

O *INMP441* utiliza um protocolo de comunicação serial conhecido como *Inter-IC Sound*, ou *I²S*. Quando estimulado por um sinal de *clock*, o dispositivo envia bits correspondentes às amostras de maneira síncrona.

Na placa de circuitos contendo o microfone que foi utilizada, é dado o acesso a 6 pinos (Figura 2.8). Os *bits* de dados são enviados através do pino *SD* e o sinal *clock* é recebido pelo pino *SCK*. Alimentação de 3,3V é conectada aos pinos *GND* e *VDD*. *WS* é um sinal que auxilia a comunicação síncrona entre os dispositivos e será gerado pelo *ESP32*. E o pino *L/R* é utilizado principalmente quando são conectados dois microfones *INMP441*, para escolher qual é o esquerdo (*L*) e qual é o direito (*R*).

Figura 2.8: Diagrama de pinos do INMP441. Fonte: o próprio autor.



A Figura 2.9 ajuda a entender melhor o protocolo I²S padrão mono que será utilizado. A comutação do sinal WS indica o início de uma nova amostra e o sinal de *clock* determina o envio dos *bits* de dados. Duas amostras consecutivas (cada uma com 24 bits) compõem um quadro de dados. Ou seja, cada quadro de dados contém uma amostra em seu lado esquerdo (canal esquerdo) e uma em seu lado direito (canal direito). Fazendo $L/R = 0$, aterrando seu pino, o microfone irá emitir amostras no canal esquerdo do quadro de dados.

Figura 2.9: Visualização do protocolo I²S em diferentes modalidades. Fonte: [6]

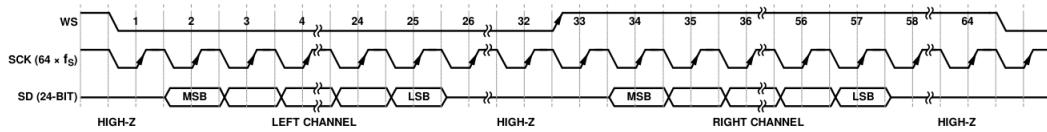


Figure 8. Stereo-Output I²S Format

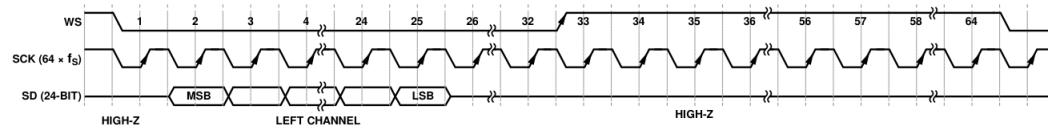


Figure 9. Mono-Output I²S Format Left Channel ($L/R = 0$)

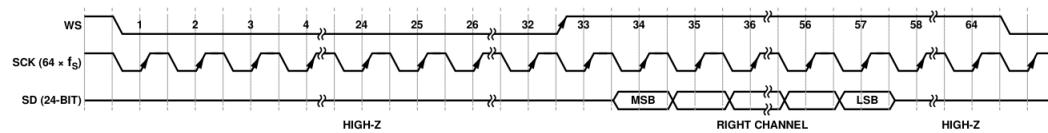


Figure 10. Mono-Output I²S Format Right Channel ($L/R = 1$)

O papel do microcontrolador é de gerar sinais de *clock* e WS necessários para que a frequência de amostragem seja de 44,1kHz, além de realizar a leitura dos dados recebidos através de SD.

Isso é feito através dos periféricos de I²S do ESP32. A estrutura *ESP-IDF* possui diferentes modos do protocolo I²S, mas o que coincide com o do INMP441 é o modo padrão ou *standard* (*std*). Após incluir as bibliotecas necessárias, em especial `driver/i2s_std`, a configuração de um periférico como receptor é realizada:

```
284 static void iniciar_i2s_std_simplex(void)
285 {
```

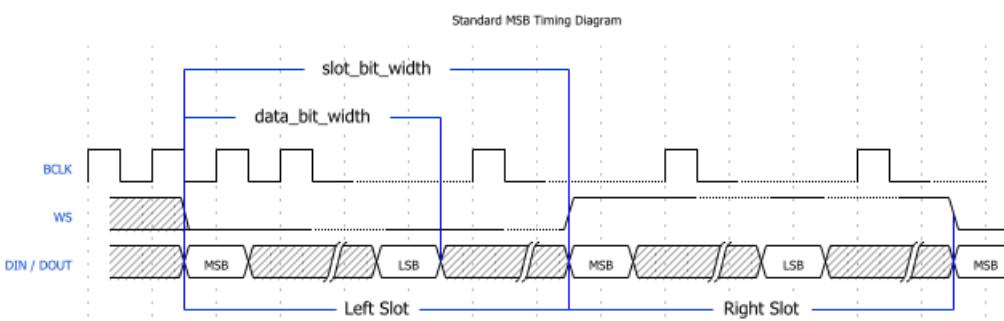
```

286 // Criando canal de recepção como mestre, ou seja, o microcontrolador vai gerar o bit clock (BCLK) e word select (WS):
287 i2s_chan_config_t rx_chan_cfg = I2S_CHANNEL_DEFAULT_CONFIG(I2S_NUM_AUTO, I2S_ROLE_MASTER);
288 ESP_ERROR_CHECK(i2s_new_channel(&rx_chan_cfg, NULL, &rx_chan));
289
290 // Configurações para usar canal no modo padrão (standard ou std)
291 i2s_std_config_t rx_std_cfg = {
292     // Configuração de clock:
293     .clk_cfg = {
294         .sample_rate_hz = FREQ_AMOST, // Frequência de amostragem em Hz
295         .clk_src = I2S_CLK_SRC_PLL, // Fonte de clock de alta frequência recomendada para áudio. O valor padrão é
296         I2S_CLK_SRC_PLL_160M
297         .mclk_multiple = I2S_MCLK_MULTIPLE_256, // Múltiplo do mclock, não usado, mas quando não configurado dá erro no
298         // programa
299     },
300     .slot_cfg = I2S_STD_MSB_SLOT_DEFAULT_CONFIG(I2S_DATA_BIT_WIDTH_16BIT, I2S_SLOT_MODE_MONO), // Formato MSB mono com
301     // largura de bits de dados de 16 bits
302     .gpio_cfg = {
303         .mclk = I2S_GPIO_UNUSED, // Sinal de mclock não usado
304         .bclk = BCLK_IO, // Porta do bit clock
305         .ws = WS_IO, // Porta do word select
306         .dout = I2S_GPIO_UNUSED, // Saída de dados não usada
307         .din = DIN_IO, // Porta do data in ou SD
308         .invert_flags = { // Não inverte nenhum sinal
309             .mclk_inv = false,
310             .bclk_inv = false,
311             .ws_inv = false,
312         },
313     },
314 };
315 rx_std_cfg.slot_cfg.slot_mask = I2S_STD_SLOT_LEFT; // Leitura do I2S feita no canal esquerdo
316 ESP_ERROR_CHECK(i2s_channel_init_std_mode(rx_chan, &rx_std_cfg)); // Inicializa o canal I2S
317 }

```

Onde é selecionada a frequência de amostragem desejada, o lado do canal, a fonte do clock, sendo "I2S_CLK_SRC_PLL" o clock *PLL* de áudio e as portas do periférico. Também é definida a formatação dos dados. Ao utilizar a função `I2S_STD_MSB_SLOT_DEFAULT_CONFIG`, é escolhido o formato *MSB* (Figura 2.10).

Figura 2.10: Ilustração da comunicação I2S no formato *MSB*. Fonte: [2]



Aqui cabe notar uma particularidade. O periférico está sendo configurado com largura de *bits* dos dados (ou seja, a largura de *bits* de uma amostra em um canal) igual a 16 *bits*. Embora a folha de dados do microfone tenha indicado uma largura de 24 *bits*, essa foi a única configuração que pareceu funcionar.

Outro ponto que pode ser discutido, é que a *ESP-IDF* faz distinção entre largura de *bits* de dados (`data_bit_width`) e largura de *bits* de fresta ou *slot* (`slot_bit_width`). Mas por padrão, a largura de *bits* fresta é definida como sendo igual à largura de *bits* de dados.

A comunicação com o microfone é feita criando uma tarefa que confere leituras bem sucedidas usando `if (i2s_channel_read(rx_chan, r_buf, TAM_BUFFER_I2S, &r_bytes, portMAX_DELAY) == ESP_OK):`

```

334 // Tarefa de leitura I2S
335 xTaskCreate(tarefa_leitura_i2s, "tarefa_leitura_i2s", 4096, NULL, 5, &handle_i2s);

246 static void tarefa_leitura_i2s(void *args)
247 {
248     // Variaveis de amostragem
249     int leituras = 0; // Numero de leituras feitas no canal I2S
250
251     // Alocacao de buffer para leituras I2S:
252     int8_t *r_buf = (int8_t *)calloc(1, TAM_BUFFER_I2S);
253     assert(r_buf); // Confere se alocacao foi feita com sucesso
254     size_t r_bytes = 0;
255
256     ESP_ERROR_CHECK(i2s_channel_enable(rx_chan)); // Habilita canal de recepcao
257
258     vTaskDelay(2000 / portTICK_PERIOD_MS); // Descartando amostras ruins
259     while(1)
260     {
261         while(flag_comecar == 0) // Aguardando o cliente estar pronto
262         {
263         }
264         vTaskSuspend(handle_blink); // Para de piscar led
265         gpio_set_level(LED, 1); // Deixa led aceso
266         while (leituras < nleituras)
267         {
268             if (i2s_channel_read(rx_chan, r_buf, TAM_BUFFER_I2S, &r_bytes, portMAX_DELAY) == ESP_OK) // Leitura do canal de
269             recepcao I2S
270             {
271                 xQueueSend(queue, (void *)r_buf, pdMS_TO_TICKS(100)); // Buffer com bits lidos entra na fila
272                 leituras++;
273             } else {
274                 printf("Read Task: i2s read failed\n");
275             }
276             flag_comecar = 0;
277             leituras = 0;
278             gpio_set_level(LED, 0);
279         }
280         free(r_buf);
281         vTaskDelete(NULL);
282     }
}

```

Disponibilizadas pelo *Free RTOS*, tarefas são rotinas independentes que são executadas sempre que for possível processá-las e quando forem escolhidas pelo *RTOS scheduler*. Essas rotinas são importantes em aplicações em tempo real, quando é preciso devotar poder de processamento para diversas demandas diferentes, paralelamente.

Neste caso, as tarefas são usadas principalmente para realizar o protocolo *I2S* e a comunicação via *websocket*.

Para deixar o *buffer* de leitura do canal livre para uma próxima leitura e transmitir os dados para a tarefa do servidor TCP, é utilizado uma fila do tipo primeiro a entrar primeiro a sair ou *first in first out (FIFO)*:

```
324 queue = xQueueCreate(5, TAM_BUFFER_I2S); // cria queue com 5 elementos com mesmo tamanho do buffer
```

```
270 xQueueSend(queue, (void *)r_buf, pdMS_TO_TICKS(100)); // Buffer com bits lidos entra na fila
```

Essa fila também ajuda a conferir se há perda de *buffers* de leitura de canal, uma vez que ela tem tamanho máximo igual a 5. Se um buffer for colocado na fila quando ela está cheia, haverá perda de dados e isso será notado pelas contagens de leituras e envios, definidas no programa.

2.3.2 Servidor websocket TCP

O sistema desenvolvido é capaz de enviar as amostras de áudio através da internet e isso é feito com comunicação pelo protocolo *websocket TCP*. Nessa lógica, é preciso estabelecer um servidor e um cliente. Neste problema, não faz muita diferença qual máquina será servidor ou cliente e o *ESP32* foi escolhido para ser o servidor.

Isso significa que o microcontrolador cria um endereço de websocket definido por seu endereço *IP* e um número de porta. Este será o endereço acessado pelo cliente, quando a comunicação iniciar.

A tarefa que cuida desse protocolo desempenha tal função:

```
332 xTaskCreate(tarefa_servidor_tcp, "tcp_server", 4096, (void *)AF_INET, 5, &thandle_tcp);
```

```
117
118 // Tarefa que cuida da comunicacao do servidor websocket TCP
119 static void tarefa_servidor_tcp(void *pvParameters)
120 {
121     // Variaveis de comunicacao websocket
122     int sock; // Variavel que guarda o socket
123     static const char *TAG = "TCP_SOCKET"; // Variavel para visualizacao serial
124
125     int k, k2, k3; // Inteiros auxiliares
126     char strbuff[CONFIG_BUFF_SIZE + 1]; // buffer para receber mensagens via websocket
127     int namost; // Variavel auxiliar que recebe numero de amostras desejado
128     int envios = 0; // Numero de amostras enviadas por websocket TCP
129
130     // Buffer de 8 bits para recebimento via I2S:
131     int8_t *bufi2s = (int8_t *)calloc(1, TAM_BUFFER_I2S);
132     assert(bufi2s);
133
134     // Variaveis padrao do exemplo websocket TCP:
135     char addr_str[128];
136     int keepAlive = 1;
137     int keepIdle = 5;
138     int keepInterval = 5;
139     int keepCount = 3;
140     struct sockaddr_storage dest_addr;
141
142     struct sockaddr_in *dest_addr_ip4 = (struct sockaddr_in *)&dest_addr;
```

```

143 dest_addr_ip4->sin_addr.s_addr = htonl(INADDR_ANY);
144 dest_addr_ip4->sin_family = AF_INET;
145 dest_addr_ip4->sin_port = htons(PORT);
146
147 // Criar socket:
148 int listen_sock = socket(AF_INET, SOCK_STREAM, 0); // 0 para protocolo TCP
149 int opt = 1;
150 setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
151
152 ESP_LOGI(TAG, "Socket created");
153
154 bind(listen_sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
155 ESP_LOGI(TAG, "Socket bound, port %d", PORT);
156
157 // Escutar socket:
158 listen(listen_sock, 1);
159
160 while (1)
161 {
162     ESP_LOGI(TAG, "Socket listening");
163
164     struct sockaddr_storage source_addr; // salvar endereco ipv4 ou ipv6
165     socklen_t addr_len = sizeof(source_addr);
166
167     // Aceitar socket
168     sock = accept(listen_sock, (struct sockaddr *)&source_addr, &addr_len);
169     if (sock < 0)
170     {
171         ESP_LOGE(TAG, "Unable to accept connection: errno %d", errno);
172         break;
173     }
174
175     // Configurar opcao keepalive tcp
176     setsockopt(sock, SOL_SOCKET, SO_KEEPALIVE, &keepAlive, sizeof(int));
177     setsockopt(sock, IPPROTO_TCP, TCP_KEEPIDLE, &keepIdle, sizeof(int));
178     setsockopt(sock, IPPROTO_TCP, TCP_KEEPINTVL, &keepInterval, sizeof(int));
179     setsockopt(sock, IPPROTO_TCP, TCP_KEEPCNT, &keepCount, sizeof(int));
180
181     // Converter endereco ip para string
182     if (source_addr.ss_family == PF_INET)
183     {
184         inet_ntoa_r(((struct sockaddr_in *)&source_addr)->sin_addr, addr_str, sizeof(addr_str) - 1);
185     }
186
187     ESP_LOGI(TAG, "Socket accepted ip address: %s", addr_str);
188
189     // Recebendo configuracoes do cliente Socket:
190     recv(sock, &strbuff, CONFIG_BUFF_SIZE, 0); // Recebendo mensagem do cliente
191
192     // Lendo string de configuracoes (escalonavel para poder receber outras configuracoes):
193     // formato das configuracoes: Aconf1Aconf2Aconf3...X
194     // sendo conf1 = configuracao 1, conf2 = configuracao 2....
195     k = 0;
196     namost = 0;
197     while (strbuff[k] != 'X')
198     {
199         if (strbuff[k] == 'A')
200         {
201             k++;
202             k2 = k;
203             while(strbuff[k] != 'X' && strbuff[k] != 'A')
204             {
205                 k++;
206             }
207             k2 = k - k2;
208             k3 = 0;
209             for (k3 = 0; k3 < k2; k3++)
210             {
211                 strbuff[k3] = strbuff[k];
212             }
213             strbuff[k] = '\0';
214         }
215     }
216
217     if (namost > 0)
218     {
219         if (namost == 1)
220         {
221             if (strbuff[0] == 'A')
222             {
223                 strbuff[0] = '\0';
224             }
225         }
226         else
227         {
228             if (strbuff[0] == 'A')
229             {
230                 strbuff[0] = '\0';
231             }
232             if (strbuff[1] == 'A')
233             {
234                 strbuff[1] = '\0';
235             }
236         }
237     }
238
239     if (namost > 1)
240     {
241         if (namost == 2)
242         {
243             if (strbuff[0] == 'A')
244             {
245                 strbuff[0] = '\0';
246             }
247             if (strbuff[1] == 'A')
248             {
249                 strbuff[1] = '\0';
250             }
251         }
252         else
253         {
254             if (strbuff[0] == 'A')
255             {
256                 strbuff[0] = '\0';
257             }
258             if (strbuff[1] == 'A')
259             {
260                 strbuff[1] = '\0';
261             }
262             if (strbuff[2] == 'A')
263             {
264                 strbuff[2] = '\0';
265             }
266         }
267     }
268
269     if (namost > 2)
270     {
271         if (namost == 3)
272         {
273             if (strbuff[0] == 'A')
274             {
275                 strbuff[0] = '\0';
276             }
277             if (strbuff[1] == 'A')
278             {
279                 strbuff[1] = '\0';
280             }
281             if (strbuff[2] == 'A')
282             {
283                 strbuff[2] = '\0';
284             }
285         }
286         else
287         {
288             if (strbuff[0] == 'A')
289             {
290                 strbuff[0] = '\0';
291             }
292             if (strbuff[1] == 'A')
293             {
294                 strbuff[1] = '\0';
295             }
296             if (strbuff[2] == 'A')
297             {
298                 strbuff[2] = '\0';
299             }
300             if (strbuff[3] == 'A')
301             {
302                 strbuff[3] = '\0';
303             }
304         }
305     }
306
307     if (namost > 3)
308     {
309         if (namost == 4)
310         {
311             if (strbuff[0] == 'A')
312             {
313                 strbuff[0] = '\0';
314             }
315             if (strbuff[1] == 'A')
316             {
317                 strbuff[1] = '\0';
318             }
319             if (strbuff[2] == 'A')
320             {
321                 strbuff[2] = '\0';
322             }
323             if (strbuff[3] == 'A')
324             {
325                 strbuff[3] = '\0';
326             }
327         }
328         else
329         {
330             if (strbuff[0] == 'A')
331             {
332                 strbuff[0] = '\0';
333             }
334             if (strbuff[1] == 'A')
335             {
336                 strbuff[1] = '\0';
337             }
338             if (strbuff[2] == 'A')
339             {
340                 strbuff[2] = '\0';
341             }
342             if (strbuff[3] == 'A')
343             {
344                 strbuff[3] = '\0';
345             }
346             if (strbuff[4] == 'A')
347             {
348                 strbuff[4] = '\0';
349             }
350         }
351     }
352
353     if (namost > 4)
354     {
355         if (namost == 5)
356         {
357             if (strbuff[0] == 'A')
358             {
359                 strbuff[0] = '\0';
360             }
361             if (strbuff[1] == 'A')
362             {
363                 strbuff[1] = '\0';
364             }
365             if (strbuff[2] == 'A')
366             {
367                 strbuff[2] = '\0';
368             }
369             if (strbuff[3] == 'A')
370             {
371                 strbuff[3] = '\0';
372             }
373             if (strbuff[4] == 'A')
374             {
375                 strbuff[4] = '\0';
376             }
377         }
378         else
379         {
380             if (strbuff[0] == 'A')
381             {
382                 strbuff[0] = '\0';
383             }
384             if (strbuff[1] == 'A')
385             {
386                 strbuff[1] = '\0';
387             }
388             if (strbuff[2] == 'A')
389             {
390                 strbuff[2] = '\0';
391             }
392             if (strbuff[3] == 'A')
393             {
394                 strbuff[3] = '\0';
395             }
396             if (strbuff[4] == 'A')
397             {
398                 strbuff[4] = '\0';
399             }
400             if (strbuff[5] == 'A')
401             {
402                 strbuff[5] = '\0';
403             }
404         }
405     }
406
407     if (namost > 5)
408     {
409         if (namost == 6)
410         {
411             if (strbuff[0] == 'A')
412             {
413                 strbuff[0] = '\0';
414             }
415             if (strbuff[1] == 'A')
416             {
417                 strbuff[1] = '\0';
418             }
419             if (strbuff[2] == 'A')
420             {
421                 strbuff[2] = '\0';
422             }
423             if (strbuff[3] == 'A')
424             {
425                 strbuff[3] = '\0';
426             }
427             if (strbuff[4] == 'A')
428             {
429                 strbuff[4] = '\0';
430             }
431             if (strbuff[5] == 'A')
432             {
433                 strbuff[5] = '\0';
434             }
435         }
436         else
437         {
438             if (strbuff[0] == 'A')
439             {
440                 strbuff[0] = '\0';
441             }
442             if (strbuff[1] == 'A')
443             {
444                 strbuff[1] = '\0';
445             }
446             if (strbuff[2] == 'A')
447             {
448                 strbuff[2] = '\0';
449             }
450             if (strbuff[3] == 'A')
451             {
452                 strbuff[3] = '\0';
453             }
454             if (strbuff[4] == 'A')
455             {
456                 strbuff[4] = '\0';
457             }
458             if (strbuff[5] == 'A')
459             {
460                 strbuff[5] = '\0';
461             }
462             if (strbuff[6] == 'A')
463             {
464                 strbuff[6] = '\0';
465             }
466         }
467     }
468
469     if (namost > 6)
470     {
471         if (namost == 7)
472         {
473             if (strbuff[0] == 'A')
474             {
475                 strbuff[0] = '\0';
476             }
477             if (strbuff[1] == 'A')
478             {
479                 strbuff[1] = '\0';
480             }
481             if (strbuff[2] == 'A')
482             {
483                 strbuff[2] = '\0';
484             }
485             if (strbuff[3] == 'A')
486             {
487                 strbuff[3] = '\0';
488             }
489             if (strbuff[4] == 'A')
490             {
491                 strbuff[4] = '\0';
492             }
493             if (strbuff[5] == 'A')
494             {
495                 strbuff[5] = '\0';
496             }
497             if (strbuff[6] == 'A')
498             {
499                 strbuff[6] = '\0';
500             }
501         }
502         else
503         {
504             if (strbuff[0] == 'A')
505             {
506                 strbuff[0] = '\0';
507             }
508             if (strbuff[1] == 'A')
509             {
510                 strbuff[1] = '\0';
511             }
512             if (strbuff[2] == 'A')
513             {
514                 strbuff[2] = '\0';
515             }
516             if (strbuff[3] == 'A')
517             {
518                 strbuff[3] = '\0';
519             }
520             if (strbuff[4] == 'A')
521             {
522                 strbuff[4] = '\0';
523             }
524             if (strbuff[5] == 'A')
525             {
526                 strbuff[5] = '\0';
527             }
528             if (strbuff[6] == 'A')
529             {
530                 strbuff[6] = '\0';
531             }
532             if (strbuff[7] == 'A')
533             {
534                 strbuff[7] = '\0';
535             }
536         }
537     }
538
539     if (namost > 7)
540     {
541         if (namost == 8)
542         {
543             if (strbuff[0] == 'A')
544             {
545                 strbuff[0] = '\0';
546             }
547             if (strbuff[1] == 'A')
548             {
549                 strbuff[1] = '\0';
550             }
551             if (strbuff[2] == 'A')
552             {
553                 strbuff[2] = '\0';
554             }
555             if (strbuff[3] == 'A')
556             {
557                 strbuff[3] = '\0';
558             }
559             if (strbuff[4] == 'A')
560             {
561                 strbuff[4] = '\0';
562             }
563             if (strbuff[5] == 'A')
564             {
565                 strbuff[5] = '\0';
566             }
567             if (strbuff[6] == 'A')
568             {
569                 strbuff[6] = '\0';
570             }
571             if (strbuff[7] == 'A')
572             {
573                 strbuff[7] = '\0';
574             }
575         }
576         else
577         {
578             if (strbuff[0] == 'A')
579             {
580                 strbuff[0] = '\0';
581             }
582             if (strbuff[1] == 'A')
583             {
584                 strbuff[1] = '\0';
585             }
586             if (strbuff[2] == 'A')
587             {
588                 strbuff[2] = '\0';
589             }
590             if (strbuff[3] == 'A')
591             {
592                 strbuff[3] = '\0';
593             }
594             if (strbuff[4] == 'A')
595             {
596                 strbuff[4] = '\0';
597             }
598             if (strbuff[5] == 'A')
599             {
600                 strbuff[5] = '\0';
601             }
602             if (strbuff[6] == 'A')
603             {
604                 strbuff[6] = '\0';
605             }
606             if (strbuff[7] == 'A')
607             {
608                 strbuff[7] = '\0';
609             }
610             if (strbuff[8] == 'A')
611             {
612                 strbuff[8] = '\0';
613             }
614         }
615     }
616
617     if (namost > 8)
618     {
619         if (namost == 9)
620         {
621             if (strbuff[0] == 'A')
622             {
623                 strbuff[0] = '\0';
624             }
625             if (strbuff[1] == 'A')
626             {
627                 strbuff[1] = '\0';
628             }
629             if (strbuff[2] == 'A')
630             {
631                 strbuff[2] = '\0';
632             }
633             if (strbuff[3] == 'A')
634             {
635                 strbuff[3] = '\0';
636             }
637             if (strbuff[4] == 'A')
638             {
639                 strbuff[4] = '\0';
640             }
641             if (strbuff[5] == 'A')
642             {
643                 strbuff[5] = '\0';
644             }
645             if (strbuff[6] == 'A')
646             {
647                 strbuff[6] = '\0';
648             }
649             if (strbuff[7] == 'A')
650             {
651                 strbuff[7] = '\0';
652             }
653             if (strbuff[8] == 'A')
654             {
655                 strbuff[8] = '\0';
656             }
657         }
658         else
659         {
660             if (strbuff[0] == 'A')
661             {
662                 strbuff[0] = '\0';
663             }
664             if (strbuff[1] == 'A')
665             {
666                 strbuff[1] = '\0';
667             }
668             if (strbuff[2] == 'A')
669             {
670                 strbuff[2] = '\0';
671             }
672             if (strbuff[3] == 'A')
673             {
674                 strbuff[3] = '\0';
675             }
676             if (strbuff[4] == 'A')
677             {
678                 strbuff[4] = '\0';
679             }
680             if (strbuff[5] == 'A')
681             {
682                 strbuff[5] = '\0';
683             }
684             if (strbuff[6] == 'A')
685             {
686                 strbuff[6] = '\0';
687             }
688             if (strbuff[7] == 'A')
689             {
690                 strbuff[7] = '\0';
691             }
692             if (strbuff[8] == 'A')
693             {
694                 strbuff[8] = '\0';
695             }
696             if (strbuff[9] == 'A')
697             {
698                 strbuff[9] = '\0';
699             }
700         }
701     }
702
703     if (namost > 9)
704     {
705         if (namost == 10)
706         {
707             if (strbuff[0] == 'A')
708             {
709                 strbuff[0] = '\0';
710             }
711             if (strbuff[1] == 'A')
712             {
713                 strbuff[1] = '\0';
714             }
715             if (strbuff[2] == 'A')
716             {
717                 strbuff[2] = '\0';
718             }
719             if (strbuff[3] == 'A')
720             {
721                 strbuff[3] = '\0';
722             }
723             if (strbuff[4] == 'A')
724             {
725                 strbuff[4] = '\0';
726             }
727             if (strbuff[5] == 'A')
728             {
729                 strbuff[5] = '\0';
730             }
731             if (strbuff[6] == 'A')
732             {
733                 strbuff[6] = '\0';
734             }
735             if (strbuff[7] == 'A')
736             {
737                 strbuff[7] = '\0';
738             }
739             if (strbuff[8] == 'A')
740             {
741                 strbuff[8] = '\0';
742             }
743             if (strbuff[9] == 'A')
744             {
745                 strbuff[9] = '\0';
746             }
747         }
748         else
749         {
750             if (strbuff[0] == 'A')
751             {
752                 strbuff[0] = '\0';
753             }
754             if (strbuff[1] == 'A')
755             {
756                 strbuff[1] = '\0';
757             }
758             if (strbuff[2] == 'A')
759             {
760                 strbuff[2] = '\0';
761             }
762             if (strbuff[3] == 'A')
763             {
764                 strbuff[3] = '\0';
765             }
766             if (strbuff[4] == 'A')
767             {
768                 strbuff[4] = '\0';
769             }
770             if (strbuff[5] == 'A')
771             {
772                 strbuff[5] = '\0';
773             }
774             if (strbuff[6] == 'A')
775             {
776                 strbuff[6] = '\0';
777             }
778             if (strbuff[7] == 'A')
779             {
780                 strbuff[7] = '\0';
781             }
782             if (strbuff[8] == 'A')
783             {
784                 strbuff[8] = '\0';
785             }
786             if (strbuff[9] == 'A')
787             {
788                 strbuff[9] = '\0';
789             }
790             if (strbuff[10] == 'A')
791             {
792                 strbuff[10] = '\0';
793             }
794         }
795     }
796
797     if (namost > 10)
798     {
799         if (namost == 11)
800         {
801             if (strbuff[0] == 'A')
802             {
803                 strbuff[0] = '\0';
804             }
805             if (strbuff[1] == 'A')
806             {
807                 strbuff[1] = '\0';
808             }
809             if (strbuff[2] == 'A')
810             {
811                 strbuff[2] = '\0';
812             }
813             if (strbuff[3] == 'A')
814             {
815                 strbuff[3] = '\0';
816             }
817             if (strbuff[4] == 'A')
818             {
819                 strbuff[4] = '\0';
820             }
821             if (strbuff[5] == 'A')
822             {
823                 strbuff[5] = '\0';
824             }
825             if (strbuff[6] == 'A')
826             {
827                 strbuff[6] = '\0';
828             }
829             if (strbuff[7] == 'A')
830             {
831                 strbuff[7] = '\0';
832             }
833             if (strbuff[8] == 'A')
834             {
835                 strbuff[8] = '\0';
836             }
837             if (strbuff[9] == 'A')
838             {
839                 strbuff[9] = '\0';
840             }
841             if (strbuff[10] == 'A')
842             {
843                 strbuff[10] = '\0';
844             }
845         }
846         else
847         {
848             if (strbuff[0] == 'A')
849             {
850                 strbuff[0] = '\0';
851             }
852             if (strbuff[1] == 'A')
853             {
854                 strbuff[1] = '\0';
855             }
856             if (strbuff[2] == 'A')
857             {
858                 strbuff[2] = '\0';
859             }
860             if (strbuff[3] == 'A')
861             {
862                 strbuff[3] = '\0';
863             }
864             if (strbuff[4] == 'A')
865             {
866                 strbuff[4] = '\0';
867             }
868             if (strbuff[5] == 'A')
869             {
870                 strbuff[5] = '\0';
871             }
872             if (strbuff[6] == 'A')
873             {
874                 strbuff[6] = '\0';
875             }
876             if (strbuff[7] == 'A')
877             {
878                 strbuff[7] = '\0';
879             }
880             if (strbuff[8] == 'A')
881             {
882                 strbuff[8] = '\0';
883             }
884             if (strbuff[9] == 'A')
885             {
886                 strbuff[9] = '\0';
887             }
888             if (strbuff[10] == 'A')
889             {
890                 strbuff[10] = '\0';
891             }
892             if (strbuff[11] == 'A')
893             {
894                 strbuff[11] = '\0';
895             }
896         }
897     }
898
899     if (namost > 11)
900     {
901         if (namost == 12)
902         {
903             if (strbuff[0] == 'A')
904             {
905                 strbuff[0] = '\0';
906             }
907             if (strbuff[1] == 'A')
908             {
909                 strbuff[1] = '\0';
910             }
911             if (strbuff[2] == 'A')
912             {
913                 strbuff[2] = '\0';
914             }
915             if (strbuff[3] == 'A')
916             {
917                 strbuff[3] = '\0';
918             }
919             if (strbuff[4] == 'A')
920             {
921                 strbuff[4] = '\0';
922             }
923             if (strbuff[5] == 'A')
924             {
925                 strbuff[5] = '\0';
926             }
927             if (strbuff[6] == 'A')
928             {
929                 strbuff[6] = '\0';
930             }
931             if (strbuff[7] == 'A')
932             {
933                 strbuff[7] = '\0';
934             }
935             if (strbuff[8] == 'A')
936             {
937                 strbuff[8] = '\0';
938             }
939             if (strbuff[9] == 'A')
940             {
941                 strbuff[9] = '\0';
942             }
943             if (strbuff[10] == 'A')
944             {
945                 strbuff[10] = '\0';
946             }
947             if (strbuff[11] == 'A')
948             {
949                 strbuff[11] = '\0';
950             }
951         }
952         else
953         {
954             if (strbuff[0] == 'A')
955             {
956                 strbuff[0] = '\0';
957             }
958             if (strbuff[1] == 'A')
959             {
960                 strbuff[1] = '\0';
961             }
962             if (strbuff[2] == 'A')
963             {
964                 strbuff[2] = '\0';
965             }
966             if (strbuff[3] == 'A')
967             {
968                 strbuff[3] = '\0';
969             }
970             if (strbuff[4] == 'A')
971             {
972                 strbuff[4] = '\0';
973             }
974             if (strbuff[5] == 'A')
975             {
976                 strbuff[5] = '\0';
977             }
978             if (strbuff[6] == 'A')
979             {
980                 strbuff[6] = '\0';
981             }
982             if (strbuff[7] == 'A')
983             {
984                 strbuff[7] = '\0';
985             }
986             if (strbuff[8] == 'A')
987             {
988                 strbuff[8] = '\0';
989             }
990             if (strbuff[9] == 'A')
991             {
992                 strbuff[9] = '\0';
993             }
994             if (strbuff[10] == 'A')
995             {
996                 strbuff[10] = '\0';
997             }
998             if (strbuff[11] == 'A')
999             {
1000                strbuff[11] = '\0';
1001            }
1002        }
1003    }
1004
1005    if (namost > 12)
1006    {
1007        if (namost == 13)
1008        {
1009            if (strbuff[0] == 'A')
1010            {
1011                strbuff[0] = '\0';
1012            }
1013            if (strbuff[1] == 'A')
1014            {
1015                strbuff[1] = '\0';
1016            }
1017            if (strbuff[2] == 'A')
1018            {
1019                strbuff[2] = '\0';
1020            }
1021            if (strbuff[3] == 'A')
1022            {
1023                strbuff[3] = '\0';
1024            }
1025            if (strbuff[4] == 'A')
1026            {
1027                strbuff[4] = '\0';
1028            }
1029            if (strbuff[5] == 'A')
1030            {
1031                strbuff[5] = '\0';
1032            }
1033            if (strbuff[6] == 'A')
1034            {
1035                strbuff[6] = '\0';
1036            }
1037            if (strbuff[7] == 'A')
1038            {
1039                strbuff[7] = '\0';
1040            }
1041            if (strbuff[8] == 'A')
1042            {
1043                strbuff[8] = '\0';
1044            }
1045            if (strbuff[9] == 'A')
1046            {
1047                strbuff[9] = '\0';
1048            }
1049            if (strbuff[10] == 'A')
1050            {
1051                strbuff[10] = '\0';
1052            }
1053            if (strbuff[11] == 'A')
1054            {
1055                strbuff[11] = '\0';
1056            }
1057        }
1058        else
1059        {
1060            if (strbuff[0] == 'A')
1061            {
1062                strbuff[0] = '\0';
1063            }
1064            if (strbuff[1] == 'A')
1065            {
1066                strbuff[1] = '\0';
1067            }
1068            if (strbuff[2] == 'A')
1069            {
1070                strbuff[2] = '\0';
1071            }
1072            if (strbuff[3] == 'A')
1073            {
1074                strbuff[3] = '\0';
1075            }
1076            if (strbuff[4] == 'A')
1077            {
1078                strbuff[4] = '\0';
1079            }
1080            if (strbuff[5] == 'A')
1081            {
1082                strbuff[5] = '\0';
1083            }
1084            if (strbuff[6] == 'A')
1085            {
1086                strbuff[6] = '\0';
1087            }
1088            if (strbuff[7] == 'A')
1089            {
1090                strbuff[7] = '\0';
1091            }
1092            if (strbuff[8] == 'A')
1093            {
1094                strbuff[8] = '\0';
1095            }
1096            if (strbuff[9] == 'A')
1097            {
1098                strbuff[9] = '\0';
1099            }
1100            if (strbuff[10] == 'A')
1101            {
1102                strbuff[10] = '\0';
1103            }
1104            if (strbuff[11] == 'A')
1105            {
1106                strbuff[11] = '\0';
1107            }
1108        }
1109    }
1110
1111    if (namost > 13)
1112    {
1113        if (namost == 14)
1114        {
1115            if (strbuff[0] == 'A')
1116            {
1117                strbuff[0] = '\0';
1118            }
1119            if (strbuff[1] == 'A')
1120            {
1121                strbuff[1] = '\0';
1122            }
1123            if (strbuff[2] == 'A')
1124            {
1125                strbuff[2] = '\0';
1126            }
1127            if (strbuff[3] == 'A')
1128            {
1129                strbuff[3] = '\0';
1130            }
1131            if (strbuff[4] == 'A')
1132            {
1133                strbuff[4] = '\0';
1134            }
1135            if (strbuff[5] == 'A')
1136            {
1137                strbuff[5] = '\0';
1138            }
1139            if (strbuff[6] == 'A')
1140            {
1141                strbuff[6] = '\0';
1142            }
1143            if (strbuff[7] == 'A')
1144            {
1145                strbuff[7] = '\0';
1146            }
1147            if (strbuff[8] == 'A')
1148            {
1149                strbuff[8] = '\0';
1150            }
1151            if (strbuff[9] == 'A')
1152            {
1153                strbuff[9] = '\0';
1154            }
1155            if (strbuff[10] == 'A')
1156            {
1157                strbuff[10] = '\0';
1158            }
1159            if (strbuff[11] == 'A')
1160            {
1161                strbuff[11] = '\0';
1162            }
1163        }
1164        else
1165        {
1166            if (strbuff[0] == 'A')
1167            {
1168                strbuff[0] = '\0';
1169            }
1170            if (strbuff[1] == 'A')
1171            {
1172                strbuff[1] = '\0';
1173            }
1174            if (strbuff[2] == 'A')
1175            {
1176                strbuff[2] = '\0';
1177            }
1178            if (strbuff[3] == 'A')
1179            {
1180                strbuff[3] = '\0';
1181            }
1182            if (strbuff[4] == 'A')
1183            {
1184                strbuff[4] = '\0';
1185            }
1186            if (strbuff[5] == 'A')
1187            {
1188                strbuff[5] = '\0';
1189            }
1190            if (strbuff[6] == 'A')
1191            {
1192                strbuff[6] = '\0';
1193            }
1194            if (strbuff[7] == 'A')
1195            {
1196                strbuff[7] = '\0';
1197            }
1198            if (strbuff[8] == 'A')
1199            {
1200                strbuff[8] = '\0';
1201            }
1202            if (strbuff[9] == 'A')
1203            {
1204                strbuff[9] = '\0';
1205            }
1206            if (strbuff[10] == 'A')
1207            {
1208                strbuff[10] = '\0';
1209            }
1210            if (strbuff[11] == 'A')
1211            {
1212                strbuff[11] = '\0';
1213            }
1214        }
1215    }
1216
1217    if (namost > 14)
1218    {
1219        if (namost == 15)
1220        {
1221            if (strbuff[0] == 'A')
1222            {
1223                strbuff[0] = '\0';
1224            }
1225            if (strbuff[1] == 'A')
1226            {
1227                strbuff[1] = '\0';
1228            }
1229            if (strbuff[2] == 'A')
1230            {
1231                strbuff[2] = '\0';
1232            }
1233            if (strbuff[3] == 'A')
1234            {
1235                strbuff[3] = '\0';
1236            }
1237            if (strbuff[4] == 'A')
1238            {
1239                strbuff[4] = '\0';
1240            }
1241            if (strbuff[5] == 'A')
1242            {
1243                strbuff[5] = '\0';
1244            }
1245            if (strbuff[6] == 'A')
1246            {
1247                strbuff[6] = '\0';
1248            }
1249            if (strbuff[7] == 'A')
1250            {
1251                strbuff[7] = '\0';
1252            }
1253            if (strbuff[8] == 'A')
1254            {
1255                strbuff[8] = '\0';
1256            }
1257            if (strbuff[9] == 'A')
1258            {
1259                strbuff[9] = '\0';
1260            }
1261            if (strbuff[10] == 'A')
1262            {
1263                strbuff[10] = '\0';
1264            }
1265            if (strbuff[11] == 'A')
1266            {
1267                strbuff[11] = '\0';
1268            }
1269        }
1270        else
1271        {
1272            if (strbuff[0] == 'A')
1273            {
1274                strbuff[0] = '\0';
1275            }
1276            if (strbuff[1] == 'A')
1277            {
1278                strbuff[1] = '\0';
1279            }
1280            if (strbuff[2] == 'A')
1281            {
1282                strbuff[2] = '\0';
1283            }
1284            if (strbuff[3] == 'A')
1285            {
1286                strbuff[3] = '\0';
1287            }
1288            if (strbuff[4] == 'A')
1289            {
1290                strbuff[4] = '\0';
1291            }
1292            if (strbuff[5] == 'A')
1293            {
1294                strbuff[5] = '\0';
1295            }
1296            if (strbuff[6] == 'A')
1297            {
1298                strbuff[6] = '\0';
1299            }
1300            if (strbuff[7] == 'A')
1301            {
1302                strbuff[7] = '\0';
13
```

```

209     {
210         namost = namost + (strbuff[k - k2 + k3] - 48) * pow(10, k2 - 1 - k3);
211     }
212 }
213 }
214
215 // Configuracao 1: Numero de amostras desejado:
216 if(namost %4 != 0 || namost > 30 * FREQ_AMOST || namost <0)
217 {}
218 else
219 {
220     nleituras = namost * RAZAO_BUFFER_AMOST / TAM_BUFFER_I2S;
221 }
222
223
224 flag_comecar = 1; // Flag para que o modulo I2S espere o cliente estar pronto
225 envios = 0; // Usada para garantir que o numero de buffers enviados via socket sera igual ao numero de leituras I2S,
ou seja, nenhum buffer foi perdido na queue FIFO
226 while(envios < nleituras)
227 {
228     if(xQueueReceive(queue, bufi2s, pdMS_TO_TICKS(100))) // Confere chegou algum buffer na queue FIFO
229     {
230         send(sock, bufi2s, TAM_BUFFER_I2S, 0); // Envia o buffer obtido na leitura I2S para o cliente via websocket
231         envios++;
232     }
233 }
234 // Fechando o socket:
235 shutdown(sock, 0);
236 close(sock);
237 vTaskDelay(500 / portTICK_PERIOD_MS);
238
239 // Reiniciando o ESP32:
240 esp_restart();
241 }
242 close(listen_sock);
243 vTaskDelete(NULL);
244 }

```

Além disso, esta tarefa trata de criar *buffers* para receber e enviar os dados. É importante definir a ordem na qual um sistema recebe os dados e o outro envia. Assim que o servidor aceita a conexão do cliente com sucesso e registra seu endereço *websocket*, o cliente envia uma string de configurações, aqui foi usada apenas uma configuração, o número de amostras que o cliente deseja receber:

```
189     recv(sock, &strbuff, CONFIG_BUFF_SIZE, 0); // Recebendo mensagem do cliente
```

Assim, o microcontrolador está pronto para enviar ao cliente as amostras coletadas pelo periférico *I2S*. Basta que a tarefa do servidor acesse a variável global da fila *FIFO* e verifique se chegou algum *buffer* novo e no caso afirmativo, envie-o:

```

228     if(xQueueReceive(queue, bufi2s, pdMS_TO_TICKS(100))) // Confere chegou algum buffer na queue FIFO
229     {
230         send(sock, bufi2s, TAM_BUFFER_I2S, 0); // Envia o buffer obtido na leitura I2S para o cliente via websocket
231         envios++;
232     }

```

2.3.3 Cliente *websocket TCP*

Para implementar a aplicação do cliente *TCP*, escolhida a linguagem de programação *Python*, por sua facilidade de programação e acessibilidade.

Assim como no caso do servidor, é preciso implementar um endereço *websocket*. Isso é feito através da biblioteca `socket`, que realiza o processo automaticamente, utilizando o endereço de *IP* da máquina. Ao realizar esse procedimento e conectar-se ao endereço *websocket* do servidor, a troca de informações pode ser iniciada:

```
1 import socket # Modulo para comandos websocket  
15 # Conexao websocket (AF_INET = ipv4, SOCK_STREAM = TCP):  
16 client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Criando endereco websocket  
17 client.connect((Host, Port)) # Conectando ao servidor do ESP32
```

O fluxo de informações é simples. O cliente envia uma string com instruções, que conta apenas com o número de amostras desejado e então começa a receber os *buffers* correspondentes às leituras do canal *I2S* do microcontrolador:

```
27 # Enviando configuracoes ao servidor:  
28 client.send(("A" + str(namost) + "X").encode("utf-8")) # Enviando configuracoes (escalonavel para incluir varias instrucoes  
no formato: Aconf1Aconf2Aconf3...X, sendo conf1 igual a configuracao 1 e assim por diante)  
  
30 # Comeca a receber buffers:  
31 inicio = time.time()  
32 for k in tqdm(range(ngravacoes)):  
33     jsinal.append(client.recv(buffsize))  
34 fim = time.time()  
35 print("Duracao: " + str(fim - inicio))
```

Esses *buffers* são então convertidos em amostras, considerando que a cada dois elementos de 8 *bits*, tem-se uma amostra de 16 *bits*:

```
37 # Traduz os buffers em amostras (a cada dois elementos de 8 bits, uma amostra):  
38 sinal = np.zeros(namost) # Sinal de audio com zeros  
39 for l in range(len(jsinal)):  
40     for c in range(0, buffsize, razao_buffer_amost):  
41         sinal[int(l*buffsize/razao_buffer_amost) + int(c/razao_buffer_amost)] = int.from_bytes(jsinal[l][c : c + razao_buffer_amost], byteorder='big', signed=True)
```

É possível então processar essas amostras de áudio, gravá-las como arquivos *.wav* ou usá-las no classificador de veículos.

2.4 Programação do Sistema B

Os principais procedimentos para programar o sistema B se encontram nesta seção. Será dada ênfase àqueles elementos da programação que não foram utilizados no sistema A (Seção 2.3).

2.4.1 Coleta de amostras (Conversor AD)

A conversão AD da tensão gerada pelo microfone *MAX4466* é feita pelo *driver* de conversão AD contínua do *ESP32*. Essa abstração permite que seja programada a taxa de conversão que produza uma frequência de amostragem desejada (*FREQ_AMOST*) além da resolução de conversão (*LARGURA_BITS_ADC*):

```

32 // Biblioteca conversao AD continua
33 #include "esp_adc/adc_continuous.h"

93 // Inicia modulo de conversao AD continua
94 static void inicializa_adc_continuo(adc_channel_t *channel, uint8_t channel_num, adc_continuous_handle_t *out_handle)
95 {
96     adc_continuous_handle_t handle = NULL;
97
98     adc_continuous_handle_cfg_t adc_config = {
99         .max_store_buf_size = 1024,
100        .conv_frame_size = TAM_BUFF_ADC,
101    };
102    ESP_ERROR_CHECK(adc_continuous_new_handle(&adc_config, &handle));
103
104    adc_continuous_config_t dig_cfg = {
105        .sample_freq_hz = FREQ_AMOST, // Configura frequencia de amostragem
106        .conv_mode = MODO_CONVERSAO_ADC, // Escolhe quais unidades ADC sao utilizadas
107        .format = TIPO_SAIDA_ADC,
108    };
109
110    // Faz configuracao escalonavel caso sejam utilizados multiplos canais
111    adc_digi_pattern_config_t adc_pattern[SOC_ADC_PATT_LEN_MAX] = {0};
112    dig_cfg.pattern_num = channel_num;
113    for (int i = 0; i < channel_num; i++) {
114        adc_pattern[i].atten = ATENUACAO_ADC;
115        adc_pattern[i].channel = channel[i] & 0x7;
116        adc_pattern[i].unit = UNIDADE_ADC;
117        adc_pattern[i].bit_width = LARGURA_BITS_ADC;
118    }
119
120    dig_cfg.adc_pattern = adc_pattern;
121    ESP_ERROR_CHECK(adc_continuous_config(handle, &dig_cfg));
122
123    *out_handle = handle;
124 }
```

Vale notar que quando configurada como $22,05\text{kHz}$, por algum motivo, a frequência de amostragem real obtida por esse sistema é de aproximadamente $18,2\text{kHz}$.

Uma vez configurada a conversão contínua, o *driver* precisa ser iniciado e as conversões podem ser acessadas pela função *adc_continuous_read()*:

```

136     adc_continuous_handle_t handle = NULL;
137     inicializa_adc_continuo(channel, 1, &handle);
138
139     ESP_ERROR_CHECK(adc_continuous_start(handle));

142     ret = adc_continuous_read(handle, result, TAM_BUFF_ADC, &ret_num, 0); // Leitura ADC
143     if (ret == ESP_OK) { // Se a leitura nao retornou erros
144         xQueueSend(queue, (void *)result, pdMS_TO_TICKS(100)); // Buffer com bits lidos entra na fila
145     }
```

As leituras são então encaminhadas ao servidor soquete via protocolo *UART*.

2.4.2 Comunicação *UART*

Os dois microcontroladores tem seu *driver UART* configurado por rotinas iguais para garantir o sucesso da comunicação. Nestas rotinas são configurados o *baud rate* (taxa de *bits* por segundo), o tamanho das palavras binárias transmitidas e quais pinos utilizar:

```
62 // Funcao para configurar comunicacao UART
63 void iniciar_uart(void)
64 {
65     const uart_config_t uart_config = {
66         .baud_rate = 115200 * 8, // Baud rate suficiente para garantir a frequencia de amostragem de 18.2kHz com 4 bits de
67         // parada
68         .data_bits = UART_DATA_8_BITS, // Palavras enviadas por comunicacao UART tem 8 bits
69         .parity = UART_PARITY_EVEN, // Checagem de erros usando paridade
70         .stop_bits = UART_STOP_BITS_MAX, //UART_STOP_BITS_1, // 1 bit de parada
71         .flow_ctrl = UART_HW_FLOWCTRL_DISABLE, // Nao ha controle de fluxo. Foi testado e nao melhorava a comunicacao.
72         .source_clk = UART_SCLK_DEFAULT, // Clock padrao APB
73     };
74     uart_driver_install(UART_NUM_2, TAM_BUFF_RX * 2, 0, 0, NULL, 0); // Instala driver
75     uart_param_config(UART_NUM_2, &uart_config); // Configura parametros
76     uart_set_pin(UART_NUM_2, TXD_PIN, RXD_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE); // Configura pinos
77 }
```

O *ESP32* responsável pela conversão AD envia dados continuamente (*uart_write_bytes()*):

```
88     uart_write_bytes(UART_NUM_2, result, TAM_BUFF_ADC); // Envia buffer por UART
```

E o servidor passa a recebê-los quando necessário (*uart_read_bytes()*):

```
291     const int rxBytes = uart_read_bytes(UART_NUM_2, data, TAM_BUFF_ADC, 1000 / portTICK_PERIOD_MS);
292     if (rxBytes > 0) {
293         xQueueSend(queue, (void *)data, pdMS_TO_TICKS(100)); // Buffer com bits lidos entra na fila
294         leituras++;
295     }
```

2.4.3 Servidor e Cliente

O procedimento de programação do servidor e do cliente é análogo àquele realizado no sistema A (Seção 2.3).

2.5 Processamento de áudio

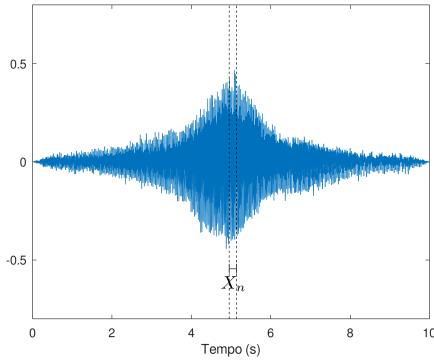
2.5.1 Pré Processamento

Após coletado um conjunto de amostras sonoras, foi preciso dividi-lo em janelas menores (Figura 2.11). Isso porque, na abordagem utilizada neste trabalho, foi preciso fazer a análise espectral sobre trechos estacionários dos sinais.

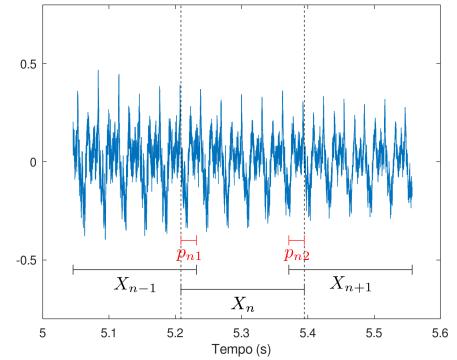
Segundo[14], para um veículo viajando a $48\text{km}/\text{h}$, uma janela de 0,186 segundos de áudio pode ser considerada estacionária, sendo o efeito de Doppler máximo possível de $\pm 4,2\%$ num componente de frequência.

Figura 2.11: Ilustração de construção de recortes estacionários a partir da assinatura sonora

(a) Assinatura sonora de uma moto com 10 segundos a $44,1\text{kHz}$ e um recorte X_n de 0,186s em pontilhado. Fonte: o próprio autor.



(b) Recortes X_n de 0,186s compartilhando vetores de amostras p_{n1} e p_{n2} com recortes vizinhos. Fonte: o próprio autor.



No caso de áudio amostrado a $44,1\text{kHz}$, por exemplo, 0,186s correspondem a aproximadamente 8203 amostras, e este foi o número N adotado. Também foi preciso permitir uma sobreposição de P amostras entre cada recorte. Isto serve para evitar a caracterização de transições bruscas entre os recortes, suavizando os resultados.

Adotando uma sobreposição de 12,5%, cada recorte deve compartilhar 1025 amostras com cada um de seus vizinhos ($P = 1025$).

Esse processo de recortar o sinal original em janelas equivale, matematicamente, a multiplicá-lo por janelas unitárias no domínio do tempo. Essa operação leva a uma convolução no domínio da frequência que suaviza a transformada do sinal recortado.

Uma prática comum, que também foi adotada, é recortar o sinal utilizando janelas de Hamming [12]:

$$h[n] = \begin{cases} 0,54 - 0,46\cos\left(\frac{2\pi n}{N-1}\right) & , 0 \leq n \leq N-1 \\ 0 & , \text{em todo o resto} \end{cases} \quad (16)$$

Para um recorte utilizando janela unitária X' , com elementos x'_n , o recorte utilizando janela de Hamming X pode ser escrito como

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \quad x_n = h[n]x'_n \quad (17)$$

2.5.2 Análise espectral e extração de características

O cálculo do espectro de frequências é feito sobre cada uma das janelas estacionárias, para que seja possível, posteriormente, extrair as características que melhor informam sobre a natureza do sinal.

Como foi visto no na seção 1.1.10, podemos construir a transformação das L janelas estacionárias de tamanho N de forma elegante através de um único produto matricial

$$\bar{\Phi}''' = M_N X \quad (18)$$

, onde

M_N : Matriz DTF de tamanho N ;

X : Matriz $N \times L$ contendo os recortes em colunas.

O resultado $\bar{\Phi}'''$ é uma matriz $N \times L$ que contém, então, as transformadas de cada recorte em cada uma de suas colunas.

Naturalmente, $\bar{\Phi}'''$ apresenta valores complexos e aqui utilizaremos as transformadas em módulo

$$\bar{\Phi}'' = |\bar{\Phi}'''| = \begin{bmatrix} \bar{\phi}_{11}'' & \bar{\phi}_{12}'' & \dots & \bar{\phi}_{1L}'' \\ \bar{\phi}_{21}'' & \ddots & & \bar{\phi}_{2L}'' \\ \vdots & & \ddots & \vdots \\ \bar{\phi}_{N1}'' & \dots & \dots & \bar{\phi}_{NL}'' \end{bmatrix} \quad (19)$$

Para realizar uma avaliação prévia das características dessas transformadas, é possível traçar um gráfico da sua média

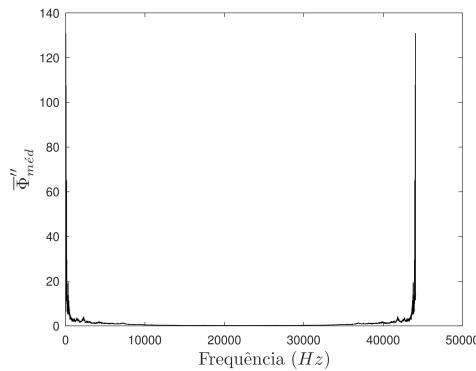
$$\bar{\Phi}_{med}'' = \begin{bmatrix} \bar{\phi}_{med_1}'' \\ \bar{\phi}_{med_2}'' \\ \vdots \\ \bar{\phi}_{med_N}'' \end{bmatrix}, \bar{\phi}_{med_n}'' = \frac{\sum_{l=0}^{L-1} \bar{\phi}_{nl}''}{L} \quad (20)$$

, como na Figura 2.12, que exibe o espectro do áudio de uma moto.

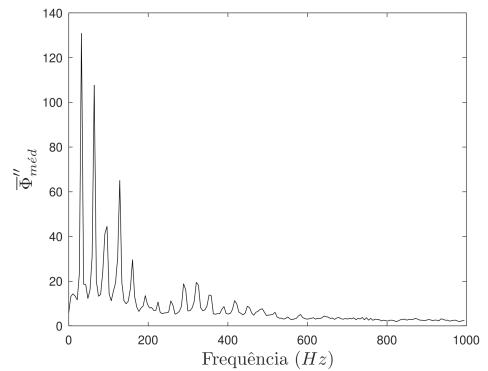
Um aspecto relevante a ser tratado é a concentração de energia do sinal em frequências até 4.000Hz (90%, para a maioria dos veículos).[14] E isso faz com que seja possível reduzir a quantidade de dados tratados.

Figura 2.12: Diferentes recortes do espectro de frequências médio da moto da Figura 2.11.

(a) Espectro de frequências médio completo obtido com uma DFT. Fonte: o próprio autor.



(b) Espectro de frequências médio reduzido à parte de interesse. Fonte: o próprio autor.



Sabe-se que cada linha da matriz DFT e da transformada obtida está associada a uma frequência de forma crescente. Logo, para obter os valores da transformada para uma determinada faixa de frequências, basta selecionar as linhas de interesse, sabendo que para a linha n , a frequência correspondente é

$$Freq_n = \frac{F_a n}{N} \quad (21)$$

, sendo

F_a : Frequência de amostragem do sinal no domínio do tempo.

Pode-se observar essa redução na Figura 2.12b.

Para mais clareza, neste texto, serão escolhidas todas as frequências entre $n = 0$ e $n = N_{lim}$. Logo, a matriz de transformadas reduzida em frequências de interesse é

$$\bar{\Phi}' = \begin{bmatrix} \bar{\phi}_{11}'' & \bar{\phi}_{12}'' & \dots & \bar{\phi}_{1L}'' \\ \bar{\phi}_{21}'' & \ddots & & \bar{\phi}_{2L}'' \\ \vdots & & \ddots & \vdots \\ \bar{\phi}_{N_{lim}1}'' & \dots & \dots & \bar{\phi}_{N_{lim}L}'' \end{bmatrix} \quad (22)$$

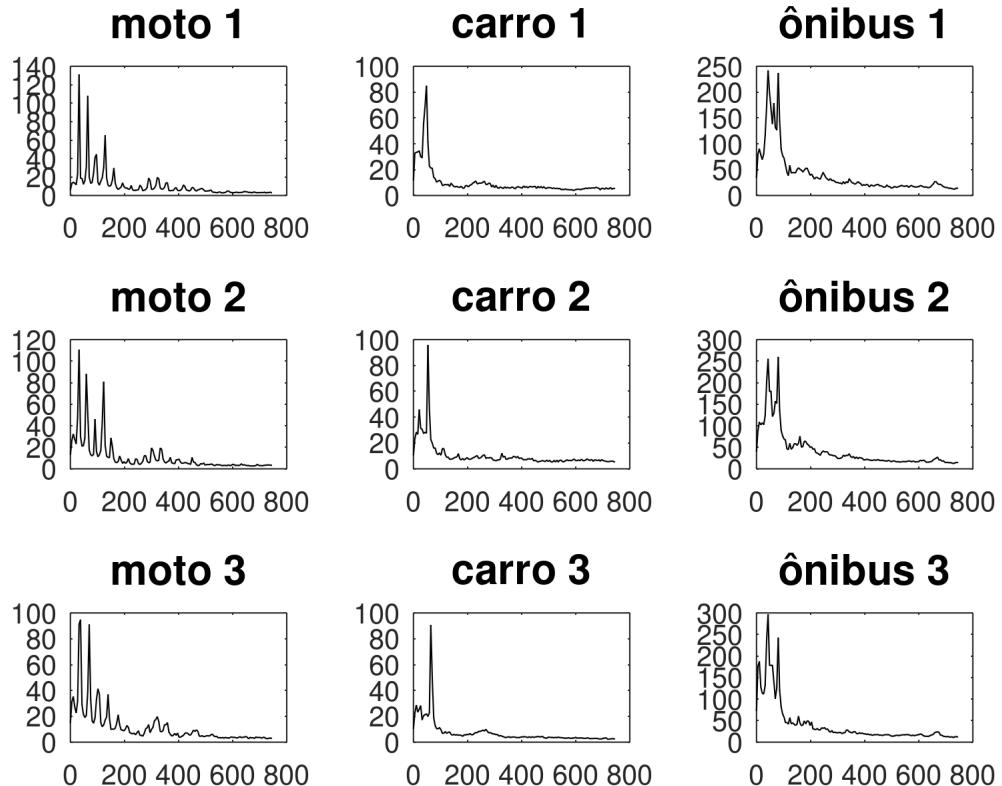
, sendo

N_{lim} : Linha correspondente à frequência máxima desejada.

Este é um dos passos mais importantes do método, pois estamos assumindo que esses vetores de espectro carregam informação suficiente a respeito do sinal para proporcionar sua classificação.

Na Figura 2.13 são apresentados os espectros de 9 sinais diferentes, selecionados entre as 3 classes em estudo (motos, carros, ônibus) e restringidos a $800Hz$ para melhor visualização.

Figura 2.13: Espectros médios de áudios pertencentes às três classes. Fonte: o próprio autor.



É possível imaginar que existem semelhanças no formato do espectro entre membros de uma mesma classe e diferenças entre classes. E nas próximas seções essa questão é explorada.

2.5.3 Ajustes e normalização

Algumas modificações adicionais são necessárias antes do treinamento do classificador.

Primeiramente, é feita a normalização para obter potência unitária sobre cada janela:

$$\bar{\Phi}_l = \begin{bmatrix} \bar{\phi}_{11} & \bar{\phi}_{12} & \dots & \bar{\phi}_{1L} \\ \bar{\phi}_{21} & \ddots & & \bar{\phi}_{2L} \\ \vdots & & \ddots & \vdots \\ \bar{\phi}_{N_{lim}1} & \dots & \dots & \bar{\phi}_{N_{lim}L} \end{bmatrix}, \quad \bar{\phi}_{nl} = \frac{\bar{\phi}'_{nl}}{\sum_{n=0}^{N_{nl}-1} \bar{\phi}'_{nl}} \quad (23)$$

Isso é feito por conta da dificuldade de controlar as condições de gravação que afetam essa variável, como por exemplo a proximidade entre veículo e microfone.

Adicionalmente, é preciso manipular os espectros em escala logarítmica, dando origem à matriz

$$\Phi = \begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1L} \\ \phi_{21} & \ddots & & \phi_{2L} \\ \vdots & & \ddots & \vdots \\ \phi_{N_{lim}1} & \dots & \dots & \phi_{N_{lim}L} \end{bmatrix}, \quad \phi_{nl} = C_2 \log_{10}(C_1 \bar{\phi}_{nl} + 1) \quad (24)$$

, onde

C_1, C_2 : Constantes escolhidas empiricamente.

Essa mudança tem por objetivo evitar que pequenas partes de variação do espectro sejam mais evidentes que a visão geral.

2.5.4 Classificação

O método de classificação deste trabalho é baseado na análise de componentes principais. Portanto, é necessária a construção de bases vetoriais a partir dos dados disponíveis.

Na abordagem utilizada, foi construída uma base para cada uma das classes. E essa construção é iniciada com o agrupamento ou concatenação das janelas espectrais contidas em matrizes Φ .

É possível escolher quaisquer janelas espectrais entre todas as instâncias de uma classe para representá-la. Então digamos que são escolhidas K janelas pertencentes à classe c , resultando na matriz

$$\Psi_c = \begin{bmatrix} \psi_{c11} & \psi_{c12} & \dots & \psi_{c1K} \\ \psi_{c21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \psi_{cN_{lim}1} & \dots & \dots & \psi_{cN_{lim}K} \end{bmatrix} \quad (25)$$

Dada a média

$$\bar{\Psi}_c = \begin{bmatrix} \bar{\psi}_{c1} \\ \bar{\psi}_{c2} \\ \vdots \\ \bar{\psi}_{cN_{lim}} \end{bmatrix}, \quad \bar{\psi}_{cn} = \frac{\sum_{k=0}^{K-1} \psi_{cnk}}{K} \quad (26)$$

e sendo Ψ_{ck} a coluna k de Ψ_c , temos a matriz de covariância

$$M = \frac{1}{K} \sum_{k=0}^{K-1} (\Psi_{ck} - \bar{\Psi}_c)(\Psi_{ck} - \bar{\Psi}_c)^T \quad (27)$$

, da qual são extraídos os autovetores θ_n e seus autovalores correspondentes λ_n , sendo $1 \leq n \leq N_{lim}$, pois uma matriz quadrada de tamanho N pode ter no máximo N autovetores.

Para tal, foi utilizado um algoritmo da biblioteca matemática consolidada *numpy*, da linguagem *python*.

Os autovetores constituem um subespaço vetorial que pode ser usado para representar a classe da qual foram extraídos. Quanto maior o valor de λ_n , mais predominante é o autovetor θ_n nessa representação.

Escolhendo os N_v autovetores mais importantes para representar a classe c , temos

$$\Theta_c = [\theta_1 \ \theta_2 \ \dots \ \theta_{N_v}] \quad , \quad \theta_n = \begin{bmatrix} \theta_{n1} \\ \theta_{n2} \\ \vdots \\ \theta_{nN_{lim}} \end{bmatrix} \quad (28)$$

e é possível comparar uma assinatura sonora desconhecida x com uma determinada classe c . Uma parte dessa comparação pode ser feita a partir da projeção das janelas espetrais no subespaço gerado por esses autovetores selecionados.

Seja Φ_{x_l} a janela espectral logarítmica l do som desconhecido x e seja $\Phi_{x_{l_{média}}}$ a média das L janelas:

$$\Phi_{x_l} = \begin{bmatrix} \phi_{x_{l_1}} \\ \phi_{x_{l_2}} \\ \vdots \\ \phi_{x_{l_{N_{lim}}}} \end{bmatrix}, \quad \Phi_{x_{média}} = \begin{bmatrix} \phi_{x_{média_1}} \\ \phi_{x_{média_2}} \\ \vdots \\ \phi_{x_{média_{N_{lim}}}} \end{bmatrix} \quad (29)$$

Dada a matriz de projeção

$$P_c = \Theta_c (\Theta_c^T \Theta_c)^{-1} \Theta_c^T \quad (30)$$

, pode-se escrever a projeção desejada como

$$\Gamma_{x_l} = P_c (\Phi_{x_l} - \Phi_{x_{média}}) = \begin{bmatrix} \gamma_{x_{l_1}} \\ \gamma_{x_{l_2}} \\ \vdots \\ \gamma_{x_{l_{N_{lim}}}} \end{bmatrix} \quad (31)$$

, onde foi feita a normalização com relação à média.

A métrica usada para classificação é um valor residual da diferença entre as janelas espetrais da assinatura sonora a ser classificada, sua projeção no subespaço de autovetores e a média das janelas usadas na construção da matriz de covariância:

$$E_{x_l} = \Phi_{x_l} - \bar{\Psi}_c - \Gamma_{x_l} = \begin{bmatrix} e_{x_{l_1}} \\ e_{x_{l_2}} \\ \vdots \\ e_{x_{l_{N_{lim}}}} \end{bmatrix} \quad (32)$$

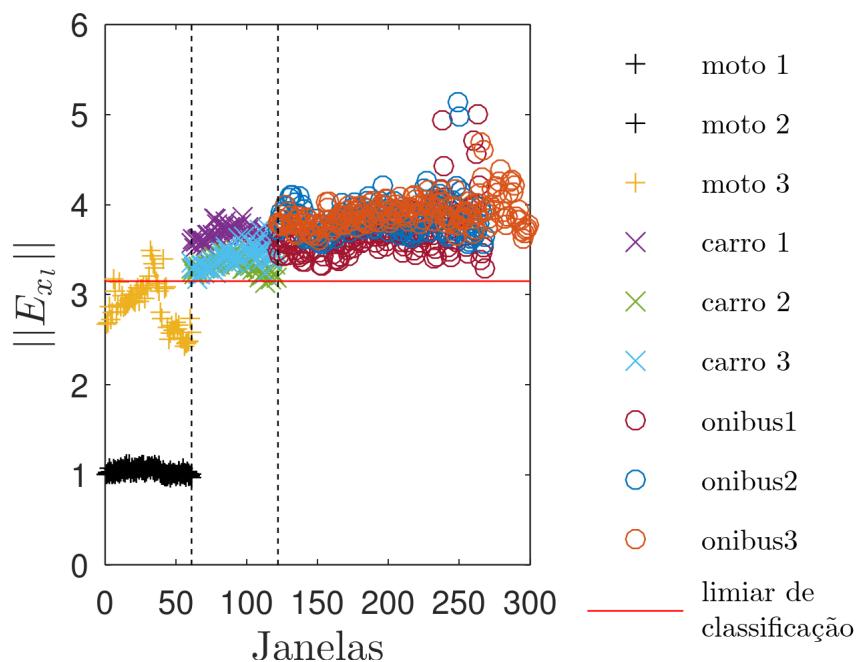
, para a qual

E_{x_l} : Vetor de valores residuais.

As normas desses vetores, $\|E_{x_l}\|$, podem então ser utilizadas como medida da proximidade entre a janela desconhecida e a classe c . É com estes valores que é feita a classificação do veículo desconhecido.

Observemos a Figura 2.14. Foi realizada a *PCA* e escolhidos os 4 autovetores mais importantes para representar a classe das motos. O valor residual foi então calculado para todas as janelas disponíveis.

Figura 2.14: Valores residuais de comparação com a classe de motos para cada janela espectral. Em preto, janelas que foram utilizadas para realizar a *PCA*. Fonte: o próprio autor.



Em preto, estão os recortes que foram utilizados na *PCA* da classe das motos, e portanto possuem a maior proximidade com suas projeções.

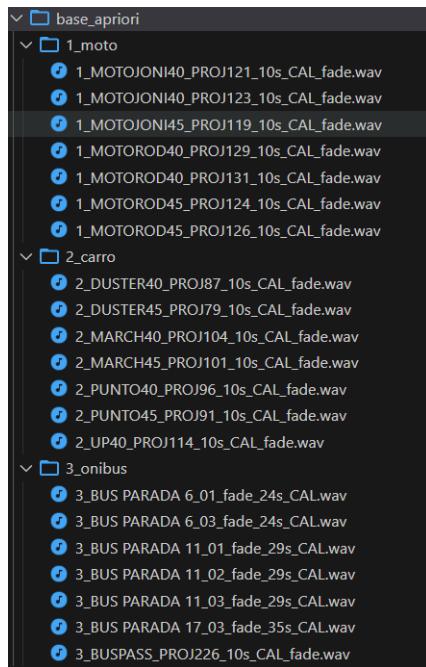
É notável que as janelas pertencentes ao som de uma moto que não foi utilizada na construção do subespaço possui valor residual menor que janelas de outras classes, de modo geral.

A partir disso, é possível utilizar um limiar de classificação para determinar quais janelas estacionárias pertencem a áudios de motos. As janelas abaixo do limiar são consideradas membros da mesma classe das janelas usadas na construção da matriz de covariância, nesse caso, motos.

3 Resultados

Para estabelecer um resultado esperado, primeiramente são apresentados os resultados do reconhecimento de padrões utilizando uma base de dados disponível *a priori*. Essa base contém 21 áudios de automóveis com até 35s de gravação cada, sendo 7 motos, 7 carros e 7 ônibus (Figura 3.1).

Figura 3.1: Organização dos arquivos da base de dados disponível *a priori*. Fonte: o próprio autor.



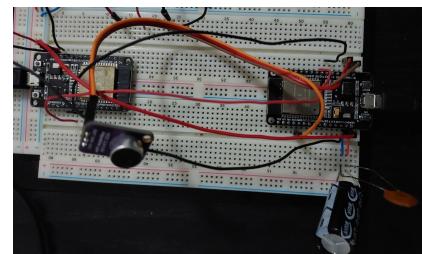
Subsequentemente são apresentados outros dois resultados utilizando essa mesma base de dados regravada pelos sistemas A e B (Figura 3.2).

Figura 3.2: Implementação dos sistemas A e B.

(a) Implementação do sistema A. Fonte: o próprio autor.



(b) Implementação do sistema B. Fonte: o próprio autor.



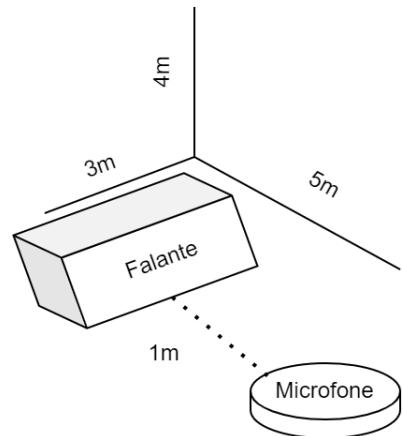
E as regravações foram feitas utilizando o falante da Figura 3.3a, numa sala $3 \times 4 \times 5m$, de acordo com o diagrama da Figura 3.3b.

Figura 3.3: Detalhes de regravação usando os sistemas A e B.

(a) Falante utilizado nas regravacoes. Fonte: o próprio autor.



(b) Diagrama do funcionamento das regravações. Fonte: o próprio autor.



3.1 Resultados com base de dados disponibilizada

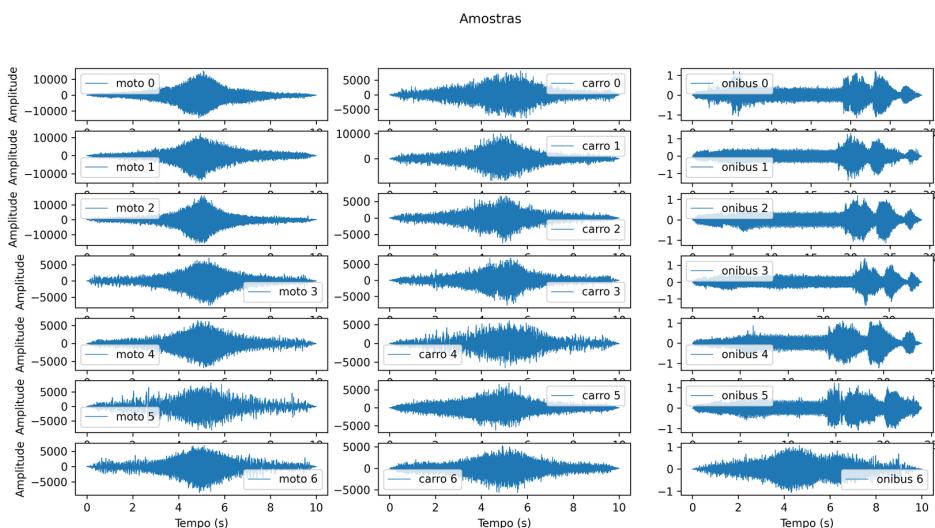
3.1.1 Amostragem

Os áudios disponíveis possuem as seguintes características:

- Frequência de amostragem de $44,1\text{kHz}$;
- Resolução de 16 bits .

e suas formas de onda podem ser visualizadas na Figura 3.4.

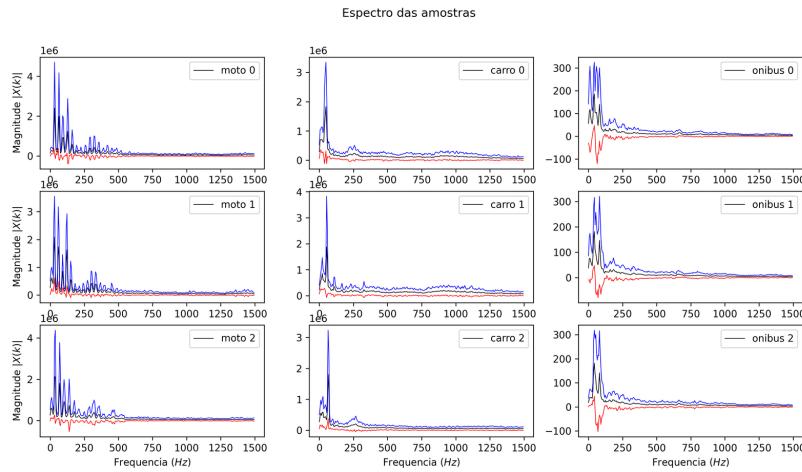
Figura 3.4: Formas de onda dos áudios disponíveis *a priori*. Fonte: o próprio autor.



3.1.2 Extração de Características

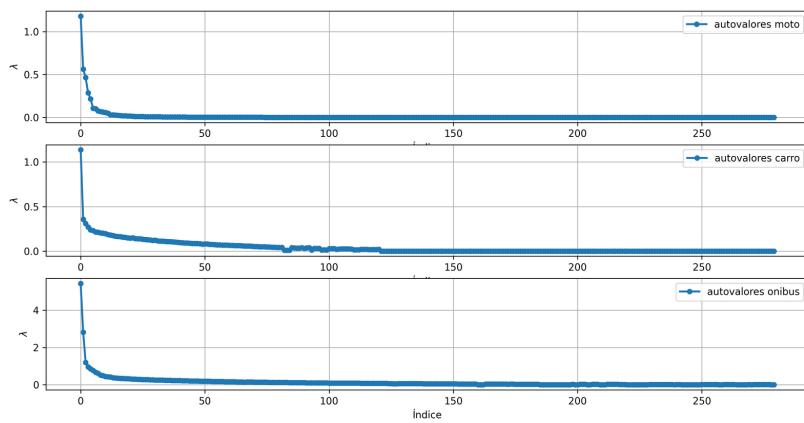
A média do módulo das transformadas discretas de Fourier até 1500Hz é apresentada na Figura 3.5, além do desvio padrão em azul e vermelho. Apenas as 3 primeiras assinaturas sonoras de cada classe estão na imagem para melhor visualização.

Figura 3.5: Módulo do espectro de frequências médio e desvio padrão para as assinaturas sonoras da base de dados disponível *a priori*. Fonte: o próprio autor.



Feita a análise espectral, são escolhidas as primeiras duas assinaturas sonoras de cada classe para representá-las através da análise de componentes principais. Ou seja, as motos 0 e 1 representam as motos, os carros 0 e 1 representam os carros e os ônibus 0 e 1 representam os ônibus. E os autovalores extraídos das matrizes de covariância estão na Figura 3.6.

Figura 3.6: Gráfico decrescente dos autovalores de cada classe obtidos a partir da base de dados disponível. Fonte: o próprio autor.



O rápido decaimento da magnitude dos autovalores até o valor zero indica uma dimen-

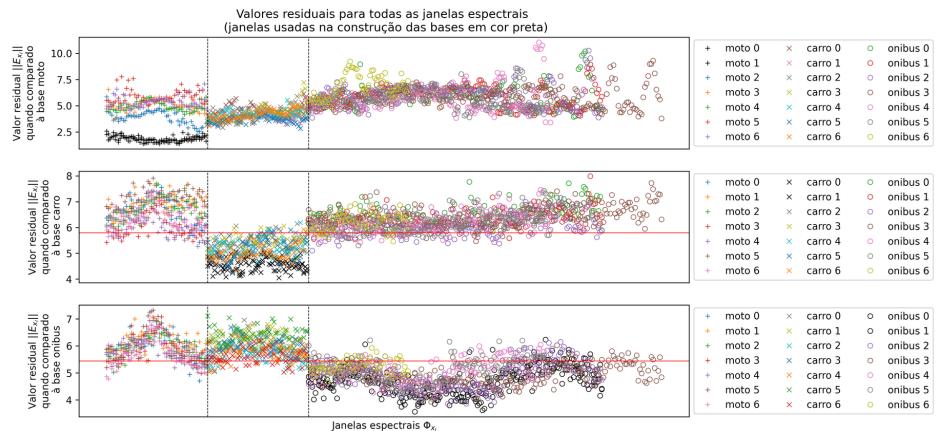
sionalidade menor para o problema. Foram escolhidos então os 4 maiores autovetores para representar cada classe, o que também torna o processamento dos dados mais eficiente.

3.1.3 Classificação

Após o cálculo dos valores residuais para cada uma das janelas espectrais estacionárias quando comparadas à cada uma das bases de autovetores, são definidos dois limiares de classificação manualmente (Figura 3.7). É destacado que os automóveis entre 2 e 6 de cada classe são aqui considerados desconhecidos.

É possível notar que o subespaço gerado pelas amostras representantes das motos não é muito útil para este problema de classificação. E esse é um problema que pode acontecer na análise de componentes principais, pois ela não maximiza critérios de separação das classes. Mas ainda é possível separar as classes a partir dos valores residuais em relação aos subespaços de carros e ônibus.

Figura 3.7: Valores residuais e limiares de classificação para a base original. Fonte: o próprio autor.



Uma parte das janelas desconhecidas é classificada como ônibus por um limiar (terceiro gráfico). Enquanto o restante é classificado como carro ou moto pelo outro limiar (segundo gráfico).

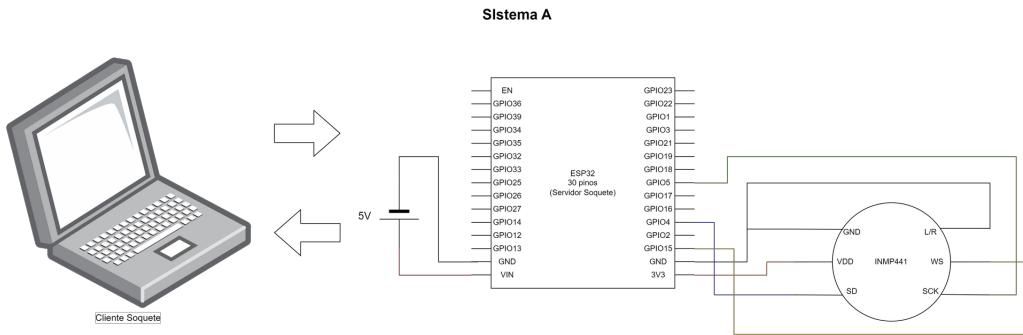
Essa abordagem levou aos seguintes resultados:

- 76,07% de acerto para janelas "desconhecidas" de motos
- 78,03% de acerto para janelas "desconhecidas" de carros
- 81,39% de acerto para janelas "desconhecidas" de ônibus

3.2 Resultados com sistema A

Nesta seção estão os resultados do sistema classificador A, cujo esquema elétrico é apresentado na Figura 3.8.

Figura 3.8: Esquema elétrico do sistema A. Fonte: o próprio autor.



3.2.1 Amostragem

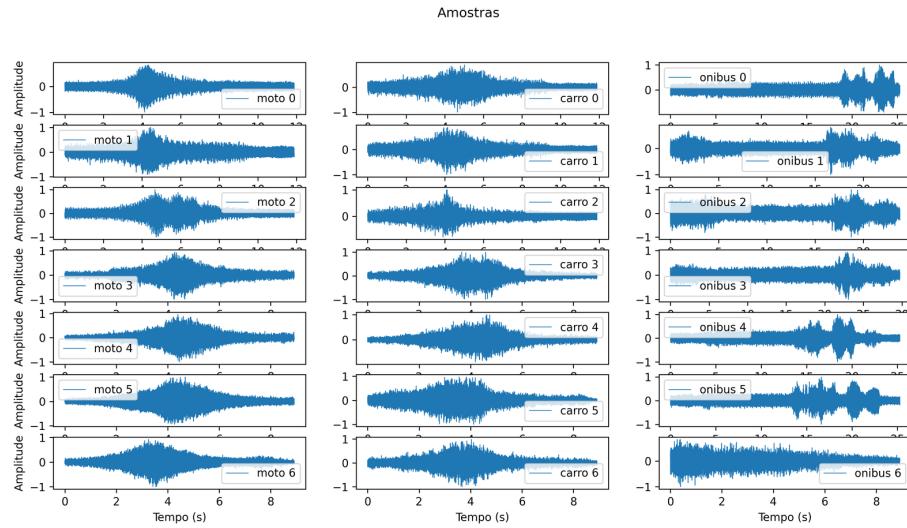
A amostragem de áudio do sistema A possui:

- Frequência de amostragem de $22,05\text{kHz}$;
- Resolução de 16 bits .

Apesar de configurar a amostragem como $44,1\text{kHz}$ no código do microcontrolador, esta é aplicada quando são usados os dois canais do quadro I^2S . Como foi usado apenas um canal, aterrando o pino L/R do microfone, a frequência real de amostragem é $22,05\text{kHz}$. E a resolução de 16 bits levanta uma problemática já discutida na Seção 2.3.

Reamostragem da base de dados na Figura 3.9.

Figura 3.9: Base de dados reamostrada pelo sistema A. Fonte: o próprio autor.

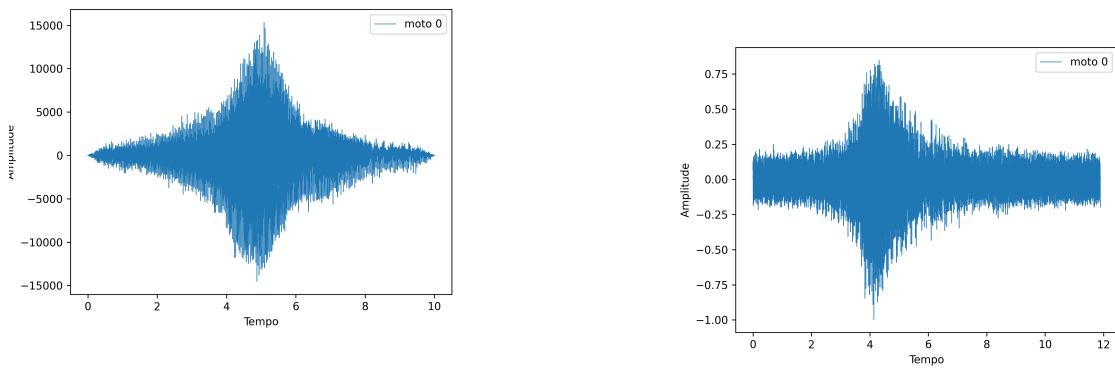


Vale notar que as amostras coletadas por esse sistema apresentam alto ruído, mesmo gravando em ambiente silencioso (Figura 3.10). E este ruído pode ser o que inviabiliza a análise de componentes principais sobre a transformada de Fourier (Seção 3.2.3).

Figura 3.10: Efeito da reamostragem do áudio da moto 0 pelo sistema A.

(a) Áudio da moto 0 disponibilizado. Fonte: o próprio autor.

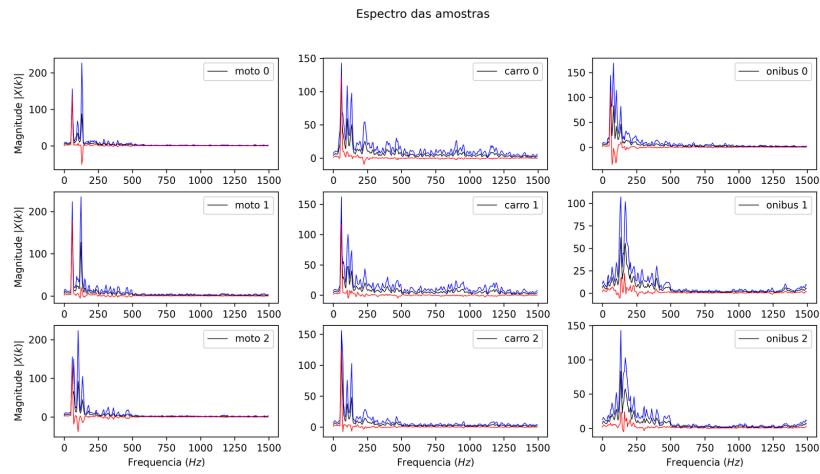
(b) Áudio da moto 0 regravado pelo sistema A. Fonte: o próprio autor.



3.2.2 Extração de Características

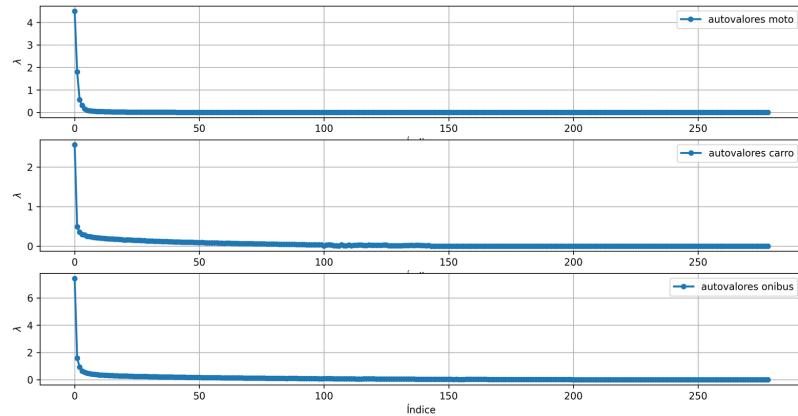
A análise espectral da Figura 3.11 indica uma grande diferença em relação ao esperado (Figura 3.5).

Figura 3.11: Módulo do espectro de frequências médio e desvio padrão para as assinaturas sonoras da base de dados reamostrada pelo sistema A. Fonte: o próprio autor.



E novamente foram escolhidos os 4 autovetores mais importantes da PCA (Figura 3.12) para gerar os subespaços.

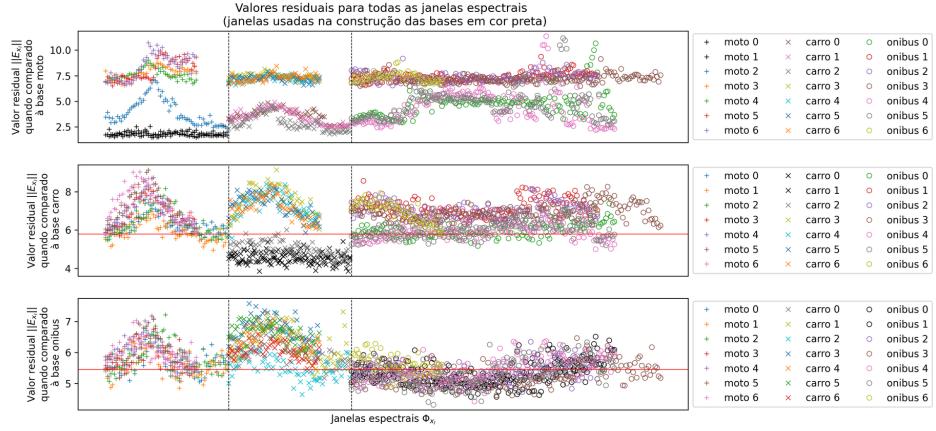
Figura 3.12: Gráfico decrescente dos autovalores de cada classe obtidos a partir da base de dados reamostrada pelo sistema A. Fonte: o próprio autor.



3.2.3 Classificação

A classificação utilizando limiares de valor residual não foi satisfatória (Figura 3.13).

Figura 3.13: Valores residuais e limiares de classificação para a base reamostrada pelo sistema A. Fonte: o próprio autor.

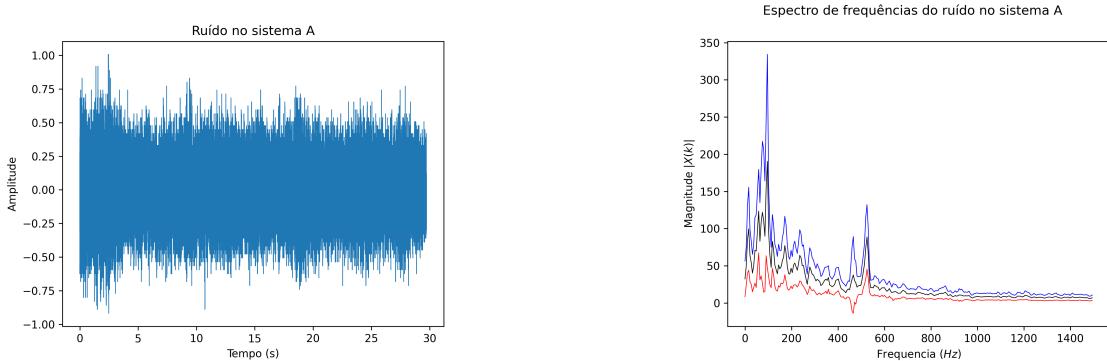


Talvez o espectro de frequências do ruído nas amostras desse sistema (Figura 3.14) esteja "embaralhando" as assinaturas sonoras.

Figura 3.14: Ruído no sistema A.

(a) Ruído no sistema A ao longo do tempo. Fonte: o próprio autor.

(b) Espectro de frequências do ruído no sistema A. Fonte: o próprio autor.

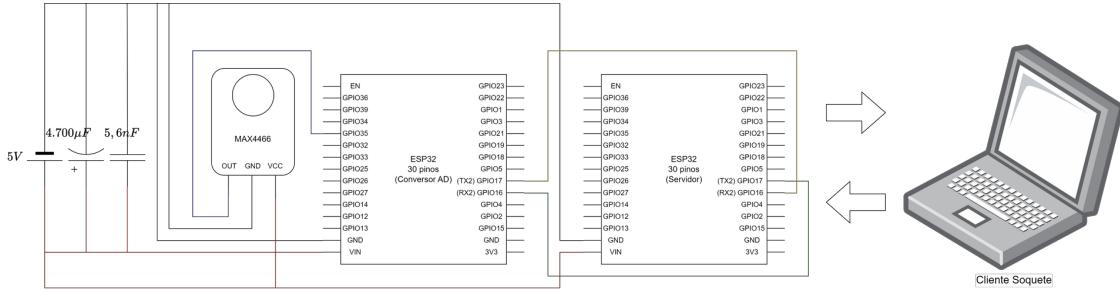


A transformada de *Fourier* do ruído no sistema A indica que suas frequências predominantes estão justamente na faixa de valores interessante à extração de características, entre 0 e 600Hz. O que pode explicar a dificuldade na classificação.

3.3 Resultados com sistema B

Nesta seção estão os resultados do sistema classificador B (Figura 3.15).

Figura 3.15: Esquema elétrico do sistema B. Fonte: o próprio autor.



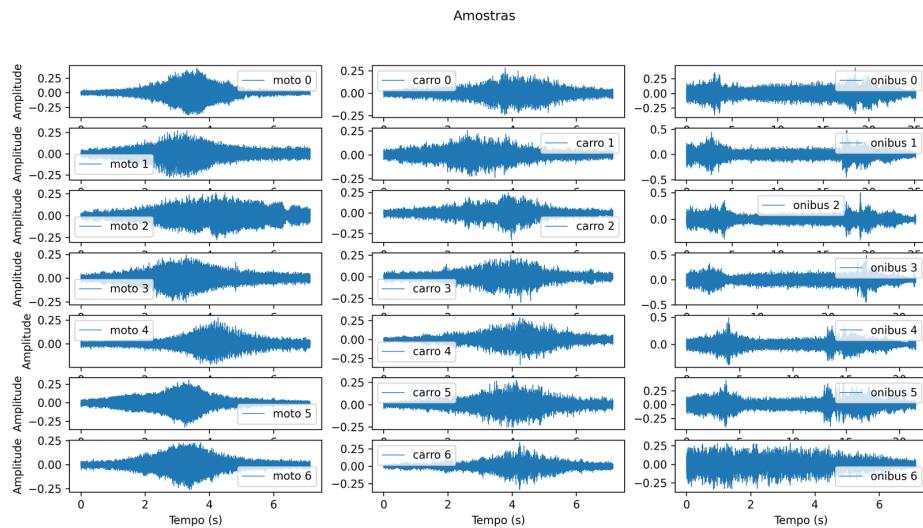
3.3.1 Amostragem

Áudio amostrado pelo sistema B possui:

- Frequência de amostragem de $18,2\text{kHz}$;
- Resolução de 12 bits .

E a reamostragem da base de dados é dada na Figura 3.16.

Figura 3.16: Base de dados reamostrada pelo sistema B. Fonte: o próprio autor.



Ao invés de realizar as operações em um único microcontrolador, a decisão de utilizar um *ESP32* exclusivamente para fazer a conversão AD surge pois o módulo de conversão AD do microcontrolador não funciona bem quando este está conectado a algum *wi-fi* (Figura 3.17).

Figura 3.17: Resultado da amostragem ao utilizar um *ESP32* para realizar a conversão AD e a comunicação soquete ao mesmo tempo.

(a) Amostragem defeituosa da primeira versão do sistema B.

Fonte: o próprio autor.



(b) Amostragem defeituosa da primeira versão do sistema B.

Fonte: o próprio autor.

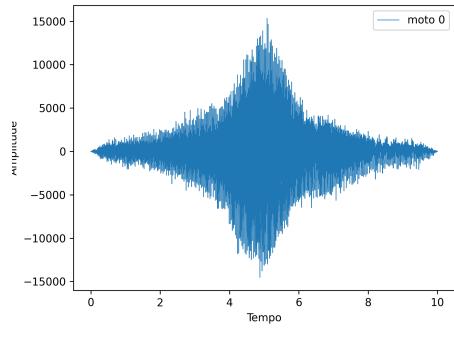


Além disso, o banco de capacitores paralelo à alimentação dos microcontroladores (Figura 3.15) foi adicionado por conta de alguns picos estranhos na conversão AD. Aparentemente o sistema estava gerando oscilações na própria alimentação, causando oscilações na conversão.

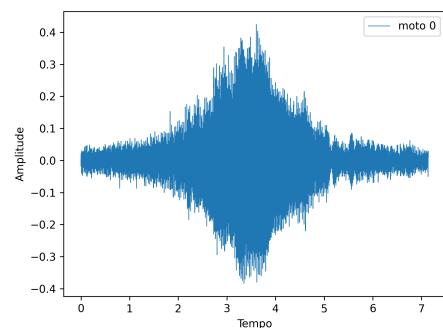
Estes problemas foram eliminados na versão final do sistema B (Figura 3.18).

Figura 3.18: Efeito da reamostragem do áudio da moto 0 pelo sistema B.

(a) Áudio da moto 0 disponibilizado. Fonte: o próprio autor.



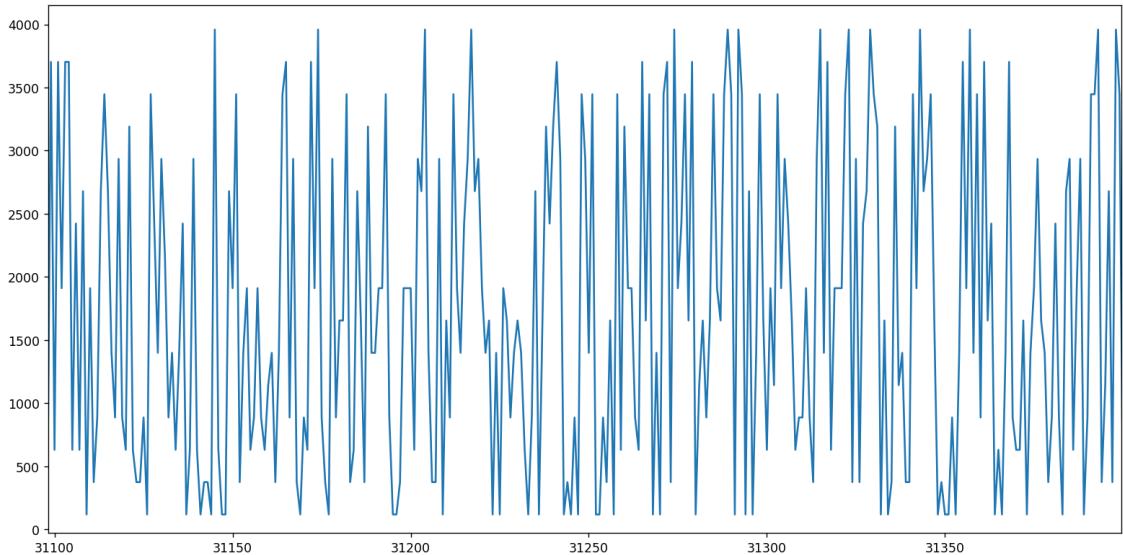
(b) Áudio da moto 0 regravado pelo sistema B. Fonte: o próprio autor.



Vale notar que o áudio gravado por este sistema soa um pouco estranho ao ouvido humano, como se houvesse alguma distorção não linear na conversão. Ainda assim, este sistema apresentou resultados muito melhores que o sistema A durante a classificação.

Por fim, algum problema faz com que comunicação UART não seja muito confiável, pois às vezes ela torna as amostras incompreensíveis (Figura 3.19).

Figura 3.19: Áudio amostrado quando há erro na comunicação *UART*. Fonte: o próprio autor.

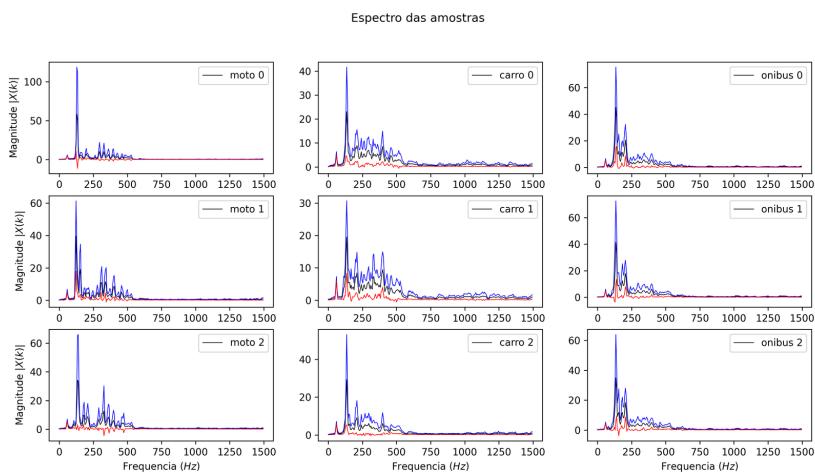


Ao *driver* da comunicação *UART*, foram adicionados checagem de erros por paridade e número maior de *bits* de parada e até foi testado controle de fluxo, mas este problema ainda não foi resolvido.

3.3.2 Extração de Características

A análise no domínio da frequência e os autovalores da *PCA* para o sistema B estão nas Figuras 3.20 e 3.21, respectivamente.

Figura 3.20: Módulo do espectro de frequências médio e desvio padrão para as assinaturas sonoras da base de dados reamostrada pelo sistema B. Fonte: o próprio autor.

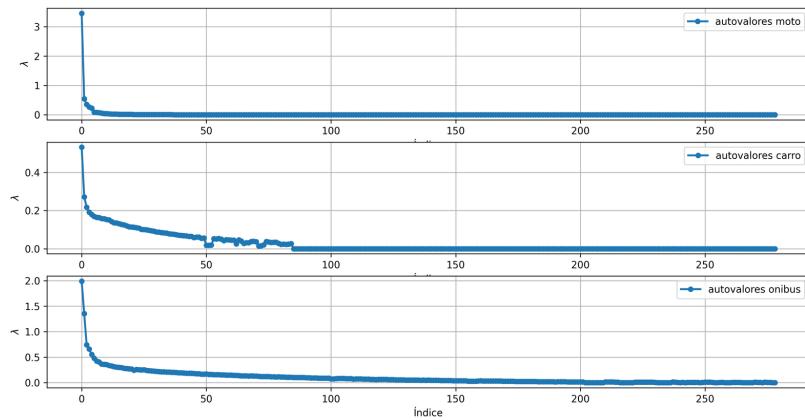


Novamente foram escolhidos os 4 autovetores mais importantes para manter condições pa-

recidas nos resultados de cada sistema.

Figura 3.21: Gráfico dos autovalores de cada classe obtidos a partir da base de dados reamostrada pelo sistema B.

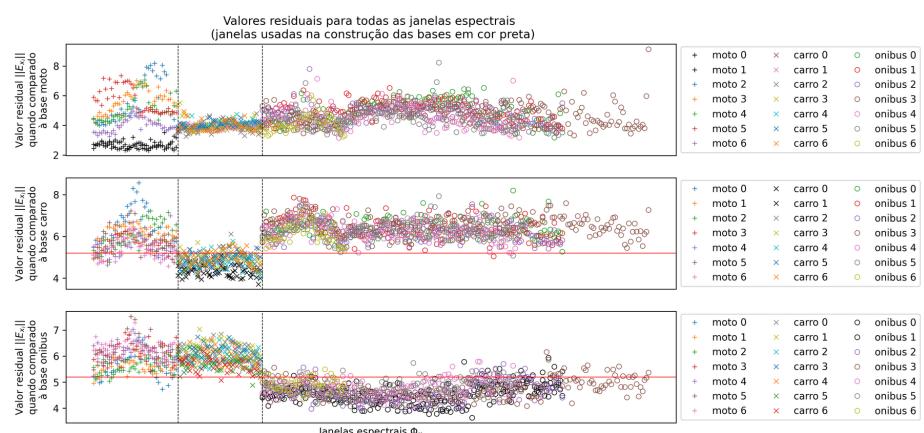
Fonte: o próprio autor.



3.3.3 Classificação

Os valores residuais calculados e limiares de classificação definidos usando a base regravada pelo sistema B se encontram na Figura 3.22.

Figura 3.22: Valores residuais e limiares de classificação para a base reamostrada pelo sistema B. Fonte: o próprio autor.



Este é um resultado muito mais próximo daquele obtido com a base de dados original.

Ao aplicar a mesma metodologia de classificação, ou seja, classificar parte das janelas estacionárias desconhecidas como ônibus com base num limiar de classificação (terceiro gráfico) e

então classificar o restante como moto ou carro, com base no outro limiar (segundo gráfico), é possível chegar às seguintes taxas de acerto:

- 77,21% de acerto para janelas "desconhecidas"de motos
- 80,47% de acerto para janelas "desconhecidas"de carros
- 87,56% de acerto para janelas "desconhecidas"de ônibus

4 Conclusão

A análise de componentes principais foi utilizada para classificar amostras de áudio pertencentes a um conjunto finito de assinaturas sonoras de motos, carros e ônibus, gravadas em condições semelhantes, com taxas de acerto maiores que 70%.

O sistema classificador A, que realiza a amostragem com um microfone *INMP441* não é interessante para técnicas envolvendo análise espectral em baixas frequências, por apresentar ruído de alta potência entre 0 e $600Hz$.

Utilizando um computador pessoal, dois microcontroladores *ESP32*, capacitores e um microfone *MAX4466*, foi possível construir um sistema classificador embarcado (sistema B) capaz de classificar áudios de automóveis desconhecidos como pertencentes às classes moto, carro ou ônibus, utilizando *PCA*, com taxas de acerto maiores que 70%.

5 Perspectiva de trabalhos futuros

5.1 Substituição do conversor AD no sistema B

Para diminuir o custo do sistema B e eliminar problemas de comunicação *UART*, o microcontrolador responsável pela conversão AD poderia ser substituído por um dispositivo conversor dedicado, como o *ADS1115* (Figura 5.1).

Figura 5.1: Conversor AD *ADS1115*. Fonte: o próprio autor.



5.2 Investigação do efeito das condições de gravação no desempenho da classificação

Tanto a base de dados original quanto as bases regravadas pelos sistemas A e B são compostas por áudios gravados em condições similares, como sugerido em [14].

É preciso testar o desempenho da classificação em condições adversas e possivelmente buscar soluções para reduzir o efeito dessas condições.

5.3 Integração de câmera ao sistema B

Existem trabalhos em classificação de automóveis que utilizam ao mesmo tempo processamento de áudio e de imagem [13]. A reprodução desses trabalhos pode ser um bom motivador para incorporar processamento de imagem aos sistemas A ou B.

Isso poderia ser feito utilizando o dispositivo *ESP32-CAM* (Figura 5.2) baseado em *ESP32*, reaproveitando a estrutura de programação já explorada neste trabalho.

Figura 5.2: *ESP32-CAM*. Fonte: o próprio autor.



Referências

- [1] Alexandre Balbinot e Valner João Brusamarello. *Instrumentação e Fundamentos de Medidas Volume 1*. 2^a ed.
- [2] *ESP-IDF Programming Guide*. URL: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/index.html>. (accessed: 28.02.2024).
- [3] Behrouz A. Forouzan e Firouz Mosharraf. *Computer Networks A Top-Down Approach*. Tata McGraw Hill Education Private Limited, 2012.
- [4] *FreeRTOS™ Real-time operating system for microcontrollers*. URL: <https://www.freertos.org/index.html>. (accessed: 28.02.2024).
- [5] Jobin George et al. “EXPLORING SOUND SIGNATURE FOR VEHICLE DETECTION AND CLASSIFICATION USING ANN”. Em: *IJSC* (2013).
- [6] *INMP441 Omnidirectional Microphone with Bottom Port and I2S Digital Output*. InvenSense.
- [7] B.P. Lathi. *Linear Systems and Signals*. 2^a ed. 198 Madison Avenue, New York: Oxford University Press, 2005.
- [8] Alberto Leon-Garcia. *Probability, Statistics, and Random Processes for Electrical Engineering*. 3^a ed. Pearson, 2008.
- [9] Luz Elena Y. Mimbela e Lawrence A. Klein. “SUMMARY OF VEHICLE DETECTION AND SURVEILLANCE TECHNOLOGIES USED IN INTELLIGENT TRANSPORTATION SYSTEMS”. Em: *FHWA* (2007).
- [10] John G. Proakis e Dimitris G. Manolakis. *Digital Signal Processing*. 4^a ed. Pearson, 2007.
- [11] Gilbert Strang. *Linear Algebra and Its Applications*.
- [12] Sergios Theodoridis e Konstantinos Koutroumbas. *Pattern Recognition*. 4^a ed. Academic Press, 2008.
- [13] Tao Wang, Zhigang Zhu e Riad Hammoud. “Audio-Visual Feature Fusion for Vehicles Classification in a Surveillance System”. Em: *IEEE* (2013), pp. 381–386.
- [14] Huadong Wu, Mel Siegel e Pradeep Khosla. “Vehicle Sound Signature Recognition by Frequency Vector Principal Component Analysis”. Em: *IEEE* (1998), pp. 429–434.