UNIVERSIDADE FEDERAL DE PERNAMBUCO

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Eletrônica e Sistemas

Graduação em Engenharia Eletrônica

Beatrice Azoubel Michalewicz

SOS Connect: Sistema de detecção sonora de pedidos de socorro

Recife
2023

Beatrice Azoubel Michalewicz

SOS Connect: Sistema de detecção sonora de pedidos de socorro

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica, do Departamento de Eletrônica e Sistemas, da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Guilherme Nunes Melo, D.Sc.

Recife
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Michalewicz, Beatrice Azoubel.
SOS Connect: Sistema de detecção sonora de pedidos de socorro / Beatrice Azoubel Michalewicz. - Recife, 2023.
125 : il., tab.

Orientador(a): Guilherme Nunes Melo
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Eletrônica - Bacharelado, 2023.
Inclui referências, apêndices.

1. Reconhecimento de fala. 2. Detecção de palavras-chave. 3. Aprendizagem de máquina. 4. Sistemas embarcados. I. Melo, Guilherme Nunes. (Orientação). II. Título.

620 CDD (22.ed.)

Beatrice Azoubel Michalewicz

SOS Connect: Sistema de detecção sonora de pedidos de socorro

Trabalho de Conclusão apresentado ao Curso de Graduação em Engenharia Eletrônica, do Departamento de Eletrônica e Sistemas, da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Aprovado em: 13/04/2023

Banca Examinadora

gov.br
Documento assinado digitalmente
GUILHERME NUNES MELO
Data: 13/05/2023 14:52:01-0300
Verifique em https://validar.iti.gov.br

Prof. Guilherme Nunes Melo, D.Sc.
Universidade Federal de Pernambuco

gov.br
Documento assinado digitalmente
HERMANO ANDRADE CABRAL
Data: 12/05/2023 15:06:26-0300
Verifique em https://validar.iti.gov.br

Prof. Hermano Andrade Cabral, Ph.D.
Universidade Federal de Pernambuco

Agradecimentos

Gostaria de agradecer a minha família, que sempre me incentivou a seguir os meus sonhos, e me apoiou incondicionalmente em todas as minhas decisões. Agradeço especialmente aos meus pais, Suzana e Michal, por me mostrarem, desde cedo, o valor da educação.

Agradeço ao meu namorado, André, que esteve ao meu lado em todos os momentos. Obrigada por tornar a jornada mais leve e por me ajudar a enfrentar os obstáculos até aqui.

Agradeço a todos os professores que tive ao longo da graduação, por me proporcionarem a base que levarei para a minha vida profissional. Cada ensinamento foi importante para a minha formação.

Agradeço ao professor Guilherme, meu orientador, por compartilhar seu conhecimento e experiência, me oferecendo todo o direcionamento necessário para o desenvolvimento deste trabalho. Agradeço a confiança depositada no meu projeto, e sua disponibilidade em todo o processo.

Speech is not just the future of Windows, but the future of computing itself.

Bill Gates

Resumo do Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica(Eng.)

SOS Connect: Sistema de detecção sonora de pedidos de socorro

Beatrice Azoubel Michalewicz

A queda é o mais frequente acidente doméstico entre idosos, e a principal causa de morte acidental em pessoas acima de 65 anos. De acordo com a Organização Mundial de Saúde (OMS), de 28 a 35% das pessoas com mais de 65 anos sofrem quedas a cada ano, aumentando para 32 a 42% entre os maiores de 70 anos. O SOS Connect surgiu da necessidade de realizar o monitoramento e detecção automática de situações de risco para a população idosa, com foco no acidente doméstico mais frequente a este grupo: as quedas. Deseja-se garantir que os pedidos de socorros sejam detectados imediatamente, informando os responsáveis e tornando o processo de atendimento, onde cada minuto importa, mais rápido e eficiente. Neste trabalho, será desenvolvido o SOS Connect, um sistema de detecção sonora de pedidos de socorro, utilizando Aprendizagem de Máquina. Através da interface gráfica do computador, será possível conectar-se remotamente a Raspberry Pi, como cliente, através da comunicação TCP-IP, e solicitar o início do monitoramento. A Raspberry Pi, ligada a um microfone e programada com um algoritmo treinado para reconhecer as palavras “socorro” e “ajuda”, inicia a detecção sonora e informa o resultado de volta ao computador.

Palavras-chave: Reconhecimento de fala; detecção de palavras-chave; python; sistemas embarcados; aprendizagem de máquina.

Abstract of Course Conclusion Work, presented to Departament of Eletronic and Systems, as a partial fulfillment of the requirements for the degree of Bachelor of Electronic Engineering(Eng.)

SOS Connect: Sound Detection System for Calls for Help

Beatrice Azoubel Michalewicz

Falling is the most frequent domestic accident among the elderly, and the main cause of accidental death in the population over 65. According to the World Health Organization (WHO), approximately 28-35% of people aged of 65 and over fall each year, increasing to 32-42% for those over 70 years of age. SOS Connect came up from the need to monitor and automatically detect risk situations among the elderly, focusing on the most frequent domestic accident for this group: falling. It aims to ensure the calls for help are detected immediately, contacting the relatives and making the rescue process, where every minute matters, faster and more efficient. This paper is about SOS Connect, a sound detection system for calls for help using Machine Learning. Through the computer's graphical interface, it will be possible to remotely connect to the Raspberry Pi, as a client, through TCP-IP communication, and request the start of the monitoring. The Raspberry Pi, connected to a microphone and programmed with an algorithm trained to recognize the words "socorro"(which means "help") and "ajuda"(which means "aide"), initiates sound detection and reports the result back to the computer.

Keywords: Speech recognition; keyword detection; python; embbebed systems; machine learning.

Sumário

1 Introdução **17**

1.1 Justificativa 17

1.2 Estado da Arte 18

1.3 Objetivo Geral 21

1.3.1 Objetivos Específicos 21

1.4 Organização do TCC 21

2 Fundamentação Teórica **23**

2.1 Sinal de Áudio 23

2.1.1 Qualidades do Som 24

2.1.2 Captação e Digitalização 26

2.1.3 Análise Sonora 29

2.2 Protocolo TCP/IP 31

2.2.1 Modelo de referência OSI e TCP/IP 31

2.3 Aprendizagem de Máquina 37

2.3.1 Ramos de Aprendizagem de Máquina 38

2.3.2 Etapas de Aprendizagem de Máquina 40

2.3.3 Algoritmos de Aprendizagem de Máquina 43

3 Metodologia **44**

3.1 Etapas de Execução do Projeto 44

3.2 *Hardware* 45

3.2.1 Microcontroladores e Microprocessadores 46

3.2.2 Microfone 49
3.2.3 Configuração Final 50
3.3 *Software* 51
3.3.1 Diagrama de Funcionamento 52
3.3.2 Bibliotecas 53
3.3.3 Aprendizagem de Máquina 55
3.3.4 Comunicação 58
3.3.5 Interface Gráfica do Usuário 60

4 Resultados **64**
4.1 Treinamento 64
4.2 Validações 65
4.2.1 Discussão dos Resultados 69
4.3 Funcionamento da Interface Gráfica 70

5 Considerações Finais **73**
5.1 Conclusão 73
5.2 Dificuldades Encontradas 74
5.3 Trabalhos Futuros 74

Referências **76**

A Apêndice **80**
A.1 Código 'Main' do SOS Server. Fonte: Autoria própria 80
A.2 Código 'Main' do SOS Client. Fonte: Autoria própria 89
A.3 Empacotamento de Variáveis. Fonte: Autoria própria 119

Lista de Ilustrações

2.1 (a) Visão instantânea das regiões de compressão e rarefação de uma onda sonora (b) Representação gráfica da variação de pressão no espaço da onda sonora. Fonte: Adaptado de (Everest, 2001) 24
2.2 Faixa de frequências e intensidades sonoras audíveis. Fonte: Adaptado de (Rossing et al., 2014) 25
2.3 Frequências das teclas de um piano. Fonte: Adaptado de (Everest, 2001) 26
2.4 Formas de onda de associadas à nota "lá". Fonte: Adaptado de (Nussenzveig, 2014) 26
2.5 Amostragem de um sinal analógico. Fonte: (Zuben, 2004) 27
2.6 Quantização do sinal amostrado. Fonte: Adaptado de (Lathi e Green, 2018) 28
2.7 Codificação do sinal quantizado. Fonte: Adaptado de (Lathi e Green, 2018) 28
2.8 Oscilograma e Espectrograma sonoros de uma frase falada, obtidos a partir do programa Audacity. Fonte: Autoria própria 29
2.9 Análise do espectro de frequência da palavra "Socorro", obtido a partir do programa Audacity. Fonte: Autoria própria 30
2.10 Espectrograma sonoro de uma frase falada. Fonte: Adaptado de (Rossing et al., 2014) 30
2.11 Modelos de referência OSI e TCP/IP. Fonte: Adaptado de (Tanenbaum, 2011) 32

2.12 Rota percorrida pelo pacote, através dos roteadores, até chegar ao seu destino. Fonte: Adaptado de (Caviglione et al., 2012) 33
2.13 Estabelecimento do *handshake*. Fonte: (Wetherall, 2011) 36
2.14 Mudança de paradigma na programação. Fonte: Adaptado de (Chollet, 2018) 38
2.15 Classificação de Algoritmos de Aprendizagem de Máquina. Fonte: Autoria própria, com base em (Raghav Bali e Lesmeister, 2016) . . . 39
2.16 Processo de Aprendizagem de Máquina. Fonte: Autoria própria . . . 40
2.17 Processo de Treinamento de um Algorítmo de Aprendizagem de Máquina. Fonte: Autoria própria 42
2.18 Visualização dos métodos de Aprendizagem de Máquina posicionados conforme interpretabilidade e performance. Fonte: Adaptado de (Badillo et al., 2020) 43

3.1 Visão superior da placa *Raspberry Pi 4 Model B*. Fonte: (Halfacree, 2019) 49
3.2 *Raspberry Pi 4 Model B*. (a) *System on Chip*. (b) Memória RAM. Fonte: (Halfacree, 2019) 49
3.3 Microfone *GXT 232 Mantis*. Fonte: (Trust, 2023) 50
3.4 Configuração final do sistema. Fonte: Autoria própria 51
3.5 Diagrama esquemático do sistema. Fonte: Autoria própria 52
3.6 Visualização das características de áudio das amostras gravadas. Fonte: Autoria própria 56
3.7 Conversão do dataset da extensão .M4A para a extensão .WAV. Fonte: Autoria própria 57
3.8 Conversão do *dataset* da taxa de amostragem 48kHz para 16k Hz. Fonte: Autoria própria 57
3.9 Ambiente de desenvolvimento do QtDesigner. Fonte: Autoria própria 61
3.10 Tela de carregamento do programa SOS Connect. Fonte: Autoria própria 61

3.11 Menu Principal do programa SOS Connect. Fonte: Autoria própria . 62
3.12 Menu de Configurações do programa SOS Connect. Fonte: Autoria própria 63

4.1 Gráfico de acurácia do sistema em função do número de *epochs*. Fonte: Autoria própria 65
4.2 Inicialização do SOS Client pelo Prompt de Comando. Fonte: Autoria própria 70
4.3 Menu Principal do programa SOS Connect conectado ao servidor. Fonte: Autoria própria 71
4.4 Possíveis resultados da detecção. Fonte: Autoria própria (a) 'Socorro' identificado. (b) 'Ajuda' identificado. (c) Não identificado. 71

Lista de Tabelas

3.1 Tabela Comparativa entre um Microcontrolador e Microprocessadores. Fonte: Autoria própria 47
3.2 Especificações do Microfone *GXT 232 Mantis*. Fonte: (Trust, 2023) . 50

4.1 Testes da palavra Socorro com voz utilizada para treinar o algoritmo. Fonte: Autoria própria 66
4.2 Testes da palavra Socorro com voz não utilizada para treinar o algoritmo. Fonte: Autoria própria 67
4.3 Testes da palavra Ajuda com voz utilizada para treinar o algoritmo. Fonte: Autoria própria 68
4.4 Testes da palavra Ajuda com voz não utilizada para treinar o algoritmo. Fonte: Autoria própria 69

Lista de Símbolos

A	....Ampère
dB	... Decibéis
GHz	.. GigaHertz
Hz	.....Hertz
KB	... Kilobyte
m^2	Metro quadrado
mA	..miliAmpère
MB	...Megabyte
MHz	.. MegaHertz
Ω	.....Ohm
V	.....Volts
W	.....Watts

Lista de Abreviações

ACK	Acknowledge
AD	Analógico-para-Digital
API	Application Programming Interface
bps	bits por segundo
CI	Circuito Integrado
CPU	Central Processing Unit
CSI	Camera Serial Interface
DA	Digital-para-Analógico
DSI	Display Serial Interface
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
GMM	Gaussian Mixture Models
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GPU	Graphics Processing Unit
GSM	Global System for Mobiles
HDMI	High-Definition Multimedia Interface
HTTP	HyperText Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
ISP	Internet Service Provider
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbors
LPC	Linear Predictive Coding
LPDDR4	Low-Power Double-Data-Rate 4

MAC ..... Media Access Control

MTU ..... Maximum Transmission Unit

OMS ..... Organização Mundial de Saúde

OSI ..... Open Systems Interconnection

PVM ..... Parallel Virtual Machine

RAM ..... Random-Access Memory

RNN ..... Recurrent Neural Network

SO ..... Sistema Operacional

SoC ..... System On Chip

SONET ..... Synchronous Optical Network

SVM ..... Support Vector Machines

SYN ..... Synchronize

TCP ..... Transmission Control Protocol

USB ..... Universal Serial Bus

WHO ..... World Health Organization

WWW ..... World Wide Web

Capítulo 1

Introdução

Segundo a Organização Mundial de Saúde (OMS), de 28 a 35% das pessoas com mais de 65 anos sofrem quedas a cada ano, aumentando para 32 a 42% entre os maiores de 70 anos (Organization, 2007). A queda é o mais frequente acidente doméstico entre idosos e a principal causa de morte acidental em pessoas acima de 65 anos (Buksman et al., 2008).

Em muitos casos, os idosos não conseguem se levantar sem ajuda após a queda. Um estudo realizado pela Universidade de Yale, que acompanhou 1103 idosos com mais de 72 anos por 21 meses, observou que 47% dos idosos que tiveram quedas sem lesões não conseguiram se levantar sem assistência (Tinetti et al., 1993). Longos períodos no chão podem agravar quaisquer lesões causadas pela queda e causar outras consequências graves. Períodos de permanência no chão superiores a 12 horas estão associados à desidratação, úlceras de pressão, rabdomiólise (ruptura do tecido muscular que libera mioglobina no sangue, podendo afetar os rins), hipotermia e pneumonia (Todd e Skelton, 2004).

1.1 Justificativa

Nesse contexto, além de medidas de prevenção às quedas, é imprescindível a implementação de medidas de monitorização e detecção de quedas, de forma a garantir socorro imediato à vítima.

Na busca de reduzir o alto índice de ocorrências fatais em acidentes e incidentes quaisquer, medidas tecnológicas passaram a ser empregadas para contactar rapidamente socorro. Muitos pesquisadores tentaram desenvolver sistemas *wearables* baseados em sensores, e sistemas com visão computacional, que monitoram o comportamento das vítimas e detectam possíveis eventos de emergência. No entanto, as abordagens existentes enfrentam desafios no monitoramento, devido a algumas desvantagens técnicas; por exemplo, se a vítima não estiver usando o dispositivo eletrônico, ou se o sinal estiver obstruído por outros objetos, não é possível realizar a monitorização adequada da vítima. Além destes fatores, os sistemas baseados em processamento de imagem não são bem aceitos devido à exposição da imagem do monitorado, gerando preocupações de falta de privacidade.

De forma a tornar mais ágil os pedidos de urgência, praticamente todos os celulares da atualidade utilizam, em seus sistemas operacionais, a funcionalidade de Emergência SOS, que permite realizar ligações automáticas de emergência ou enviar a localização atual para contatos previamente armazenados como emergenciais. Além de celulares, os atuais dispositivos *wearables* comerciais também realizam algumas destas funções, enviando dados quando possíveis quedas ou anomalias na monitorização corporal forem detectadas (Apple, 2022) (Google, 2022).

Neste contexto, desenvolveu-se o SOS Connect, um sistema de detecção sonora que utiliza Aprendizagem de Máquina para detectar palavras-chave, utilizando um sistema embarcado com microfone acoplado e enviando as informações aos clientes conectados.

1.2 Estado da Arte

Alguns trabalhos importantes na área foram coletados e servirão de fomento aos estudos relacionados a este trabalho. Serão apresentadas pesquisas relevantes, que terão relação com dispositivos de segurança comerciais, aquisição de áudio, aprendizagem de máquina e diversos tipos de monitoramento.

Uma gama de trabalhos utiliza o conceito de identificação sonora de palavras

como base. O artigo de Akif Naeem, por exemplo, utiliza uma cadeira de rodas inteligente, que responde a comandos de voz para se movimentar em diversas direções. Há uma detecção automática de obstáculos utilizando um sensor ultrassônico, que ajuda o paciente a frear a cadeira e informa-o a distância ao obstáculo. O projeto utiliza Google API (*Speech to text*) para o reconhecimento dos comandos voz (Naeem et al., 2014).

O trabalho de Felipe Negri também utiliza reconhecimento de comandos de voz para ajudar na acessibilidade, com foco em automatizar tarefas domésticas, como o controle de aparelhos eletrônicos, lâmpadas e portas. A solução também utilizou o *Google Speech Recognition* como *software* de reconhecimento de voz, com as devidas adaptações (Negri e Ledel, 2016).

O trabalho de Suellem Queiroz objetiva auxiliar pessoas na solicitação de socorro, através de um sistema que funciona como um botão de pânico, mas com a agregação de reconhecimento de atividades, sensores móveis com sensibilidade ao contexto [1], e agentes para disparo de solicitações de socorro e reconhecimento de possíveis acidentes. O sistema é desenvolvido para utilização em Sistema Operacional (SO) Android e possui também uma interface de monitoramento e recebimento de alertas *web* (Queiroz, 2016).

Entre os trabalhos pesquisados voltados para acidentes domésticos, pôde-se observar um grande foco em idosos, particularmente na detecção de quedas, com variadas abordagens. O trabalho de Zulfiqar Khan usa um acelerômetro triaxial para detectar a queda, assim como um módulo GPS para identificar a localização da queda e um módulo GSM para enviar estas informações por mensagem de texto a um responsável (Khan et al., 2018).

O artigo de Diana Yacchirema também utiliza um acelerômetro triaxial para a detecção de quedas, particularmente embutido em um dispositivo *wearable*. Ela propõe um sistema que utiliza Internet das Coisas, *Big Data* e *Cloud Computing*,

[1] Sensibilidade ao contexto se refere a computadores com comportamento adaptado de acordo com o ambiente, com consciência da localização e situação a que estão inseridos (Da Costa et al., 2008)

para coletar os movimentos de idosos em ambientes fechados em tempo real, detectando possíveis quedas, e armazenar estes dados na nuvem. Caso seja detectada uma queda, um alerta é ativado e envia-se notificações aos responsáveis (Yacchiremaa et al., 2018).

Shaikh Abdul Waheed apresenta, em seu artigo, um sistema de detecção de queda através de processamento das imagens obtidas por uma câmera Pi. O Sistema utiliza Internet das Coisas e algorítmos como *Motion History Image* - em português, Imagem da História do Movimento - e C-Motion para realizar o processamento de imagem. Caso detectado um movimento inesperado, considera-se que ocorreu uma queda. Em caso de alarme falso, o usuário tem 5 segundos para suspender o alarme e, caso contrário, o sistema irá confirmar a queda e enviar mensagens de alerta via Wi-fi para os cuidadores da vítima (Waheed e Khader, 2017).

O trabalho de J. Sree Madhubala utiliza um sensor Microsoft Kinect para monitorar o comportamento de idosos, e as imagens obtidas são processadas por um Raspberry Pi. A técnica de extração de características com reconhecimento de contexto identifica o formato da pessoa e a classificação distingue uma queda de outras atividades usuais. O sistema consegue identificar as ações de sentar, ficar em pé e cair e, em caso de queda, um alerta é enviado através de um modem GSM (Madhubala e Umamakeswari, 2015).

O trabalho de Fadi Muheidat utiliza um *pad sensor*, isto é, um tapete sensorizado, colocado embaixo do tapete convencional, capaz de identificar passos e quedas. O sistema realiza um monitoramento automatizado de pessoas idosas com armazenamento de atividades na nuvem e, caso uma queda seja detectada, um alerta é emitido para os profissionais de saúde (Muheidat et al., 2018).

Por fim, vários trabalhos adotam a identificação sonora como forma de monitoramento e identificação de quedas. É o caso do artigo de Mohamed A. Sehili , que realiza a análise de som e de voz em tempo real, utilizando a detecção do sinal, identificação do som/voz, classificação do som e reconhecimento de voz. Os sons foram classificados em 18 categorias, incluindo escovar os dentes, tosse, palmas, as-

pirador de pó, telefone tocando, entre outros. O trabalho utiliza uma combinação de método para realizar a classificação, como GMM (*Gaussian Mixture Models*) e SVM (*Support Vector Machines*) (Sehili et al., 2012).

Martin Frešer, em seu artigo, realiza a classificação sonora e faz a distinção entre 6 classes: dormir, fazer exercícios, trabalhar, comer, tarefas domésticas e descanso. A classificação é realizada baseado em 19 tipos de características sonoras, como o centro espectral, cruzamento do sinal no zero (*zero crossing*), codificação preditiva linear (LPC), entre outros. Foram testados 4 tipos de algoritmos de aprendizagem de máquina supervisionada, dentre os quais a "Máquina de Vetores de Suporte" (*Support Vector Machines*) obteve maior acurácia (Frešer et al., 2019).

1.3 Objetivo Geral

Objetiva-se desenvolver um sistema capaz de detectar de forma sonora pedidos de socorro e enviar alertas por *e-mail* ou Telegram.

1.3.1 Objetivos Específicos

- Desenvolver um programa do tipo cliente, com implementação de interface gráfica para enviar comandos ao servidor;
- Desenvolver um programa do tipo servidor, implementando um algorítmo de detecção das palavras-chave "socorro" e "ajuda";
- Implementar a comunicação remota entre cliente e servidor;

1.4 Organização do TCC

O conteúdo deste TCC está dividido em cinco capítulos e um apêndice. As referências encontram-se nas páginas finais. A seguir, um resumo dos capítulos seguintes do TCC.

Capítulo 2. Capítulo dedicado à Fundamentação Teórica, no qual é apresentado a teoria que será usada como base para o desenvolvimento do projeto.

Capítulo 3. Capítulo relativo à Metodologia, onde é descrita a forma que o projeto será desenvolvido, incluindo os componentes utilizados e as etapas de execução do projeto.

Capítulo 4. Capítulo destinado aos Resultados, onde são apresentados os resultados obtidos, como o funcionamento do sistema e a análise de seu desempenho.

Capítulo 5. Capítulo das Considerações Finais, onde descreve-se as conclusões obtidas a partir dos resultados, assim como sugestões de aprimoramento das funcionalidades e adição de melhorias que podem dar origem a trabalhos futuros.

Apêndice. Capítulo onde são mostrados os principais códigos do projeto, relativos ao código '*Main*' do SOS Server, ao código '*Main*' do SOS Client, e ao código que realiza o empacotamento de variáveis para a comunicação.

Capítulo 2

Fundamentação Teórica

Este capítulo é destinado à apresentação da teoria que será usada como base para o desenvolvimento do projeto. São explicados as características de um sinal de áudio, como funciona a Comunicação TCP-IP e noções básicas sobre Aprendizagem de Máquina.

2.1 Sinal de Áudio

O som é o resultado de vibrações de corpos elásticos, que se propagam longitudinalmente no meio, formando regiões de alta pressão (compressão) e de baixa pressão (rarefação) que se intercalam periodicamente (da Costa, 2003) (Everest, 2001). A Figura 2.1 ilustra as regiões de compressão e rarefação de uma onda sonora, assim como sua respectiva forma de onda.

A onda sonora necessita de um meio material para se propagar, o que a torna, por definição, uma onda mecânica. Também é classificada como onda longitudinal, cujas oscilações acontecem na direção de propagação da onda, em oposição às ondas transversais, cujas oscilações são perpendiculares à direção de propagação (Halliday e Resnick, 2014).

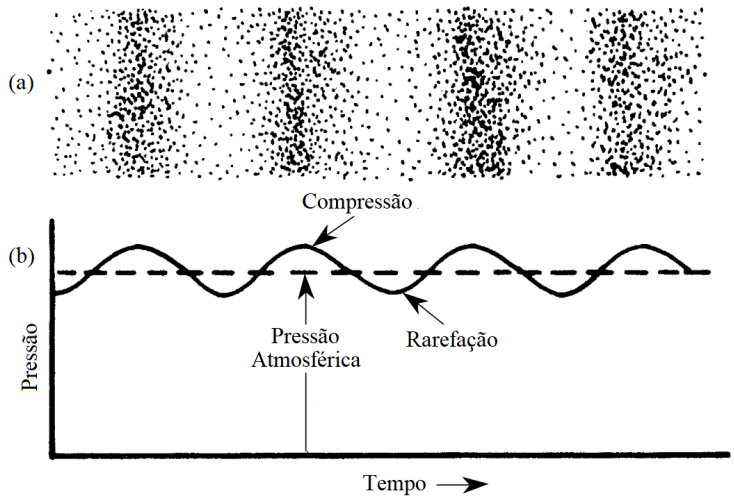


Figura 2.1: (a) Visão instantânea das regiões de compressão e rarefação de uma onda sonora (b) Representação gráfica da variação de pressão no espaço da onda sonora. Fonte: Adaptado de (Everest, 2001)

2.1.1 Qualidades do Som

É possível caracterizar uma onda sonora a partir da sensação auditiva ao som recebido, permitindo a distinção de qualidades como intensidade, altura e timbre.

Intensidade

A intensidade do som, normalmente chamada de "volume", é a qualidade associada a amplitude da onda sonora, que caracteriza a variação de pressão no meio (da Costa, 2003). A intensidade sonora é a energia média transmitida através de uma seção, por unidade de tempo e área. O limiar de audibilidade corresponde à intensidade da densidade de potência do som mais fraco que pode ser ouvido, que equivale a $I_0 \approx 10^{-12} W/m^2$ (Nussenzveig, 2014).

Na prática, costuma-se utilizar o nível de intensidade sonora, que é medido em uma escala logarítmica, de modo que incrementos iguais na escala correspondem a fatores iguais de aumento na intensidade. Adotou-se esta convenção devido ao grande alcance de intensidades audíveis, cobrindo muitas ordens de grandeza, e devido à lei empírica psicofísica de Weber e Fechner, segundo a qual a sensação auditiva produzida por um determinado estímulo é proporcional ao logaritmo da excitação sonora (Nussenzveig, 2014). Tomando o limiar de audibilidade I_0 como

intensidade de referência, define-se o nível de intensidade α por:

$$\alpha = 10log_{10}\left(\frac{I}{I_0}\right) db$$

A unidade de nível de intensidade é o decibel (db), unidade derivada do bel, nomeada em homenagem a Alexander Graham Bell. A Figura 2.2 ilustra as faixas de frequência e intensidades sonoras audíveis, assim como os limiares de audibilidade e dor.

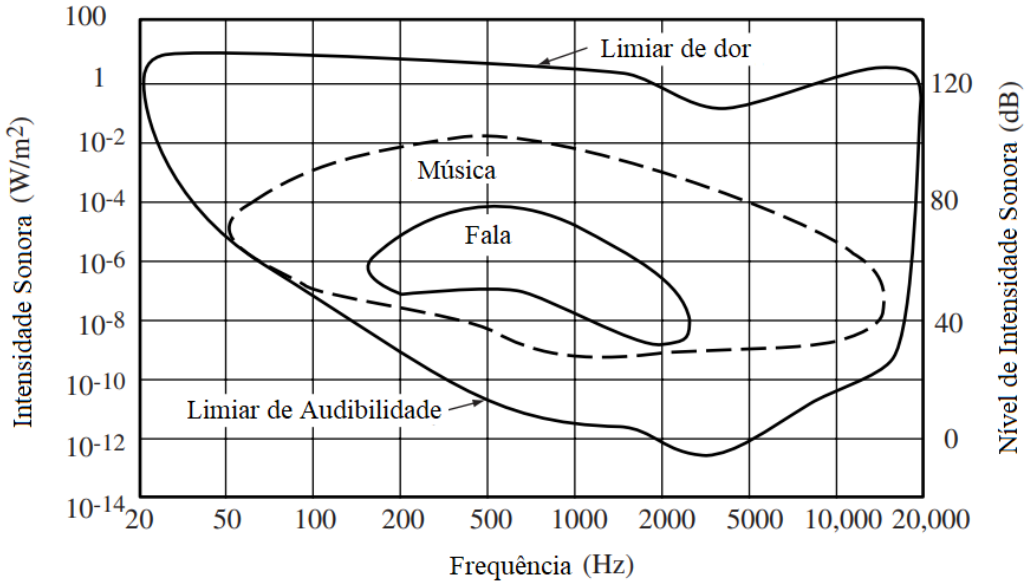


Figura 2.2: Faixa de frequências e intensidades sonoras audíveis. Fonte: Adaptado de (Rossing et al., 2014)

Altura

A altura é a qualidade associada a frequência da onda sonora, permitindo a diferenciação entre sons agudos, que estão na faixa de alta frequência, e graves, que estão na faixa de baixa frequência (da Costa, 2003).

De acordo com a altura, pode-se classificar vozes e notas musicais em escalas (da Costa, 2003). A nota mais grave do piano, por exemplo, é o La_0, de frequência 27.5 Hz, enquanto que sua nota mais aguda é o Do_8, de frequência 4186 Hz, conforme apresentado na Figura 2.3 (Everest, 2001).

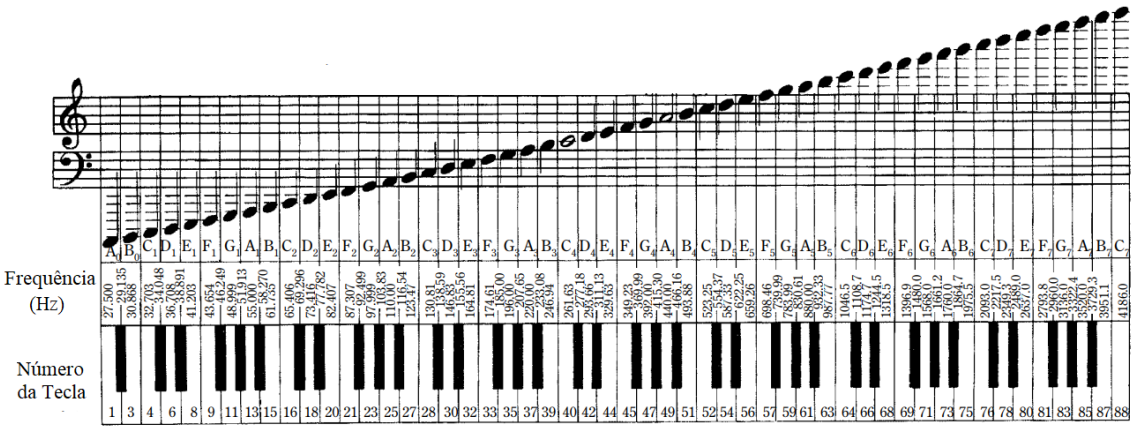


Figura 2.3: Frequências das teclas de um piano. Fonte: Adaptado de (Everest, 2001)

Timbre

O timbre é a qualidade associada à forma de onda, isto é, à composição harmônica da onda sonora. É o que permite diferenciar sons de mesma altura e intensidade, emitidos por fontes diferentes (da Costa, 2003) (Nussenzveig, 2014).

A Figura 2.4 ilustra graficamente as formas de onda associadas à nota ”Lá”, de frequência 440 Hz. Embora representem a mesma nota, o som da Figura 2.4(b) é mais rico em harmônicos de ordem elevada, indicando que possuem timbres diferentes.

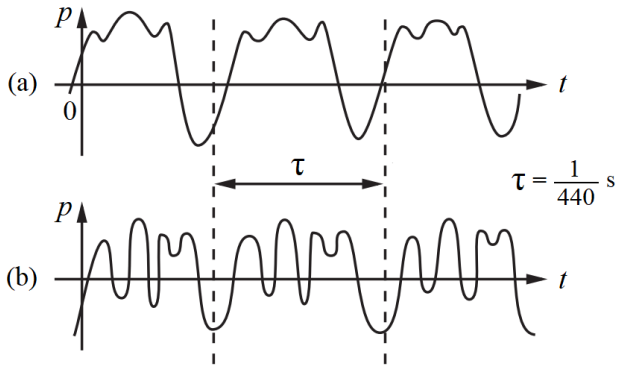


Figura 2.4: Formas de onda de associadas à nota ”lá”. Fonte: Adaptado de (Nussenzveig, 2014)

2.1.2 Captação e Digitalização

O som é um sinal analógico, isto é, sua amplitude pode assumir qualquer valor em uma faixa contínua, em oposição aos sinais digitais, cujas amplitudes só podem

assumir um número finito de valores (Lathi, 2008).

Entretanto, para que o sinal possa ser processado por meios digitais, como computadores e microcontroladores, é preciso converter o sinal analógico em sinal digital.

O processo de digitalização de um sinal analógico é constituído de três etapas: Amostragem, Quantização e Codificação. A Amostragem consiste em medir valores instantâneos de um sinal analógico em intervalos regulares. O sinal resultante da amostragem ainda é um sinal analógico, mas passa de um sinal contínuo para um sinal discreto no tempo, conforme pode-se observar na Figura 2.5 (Lathi e Green, 2018).

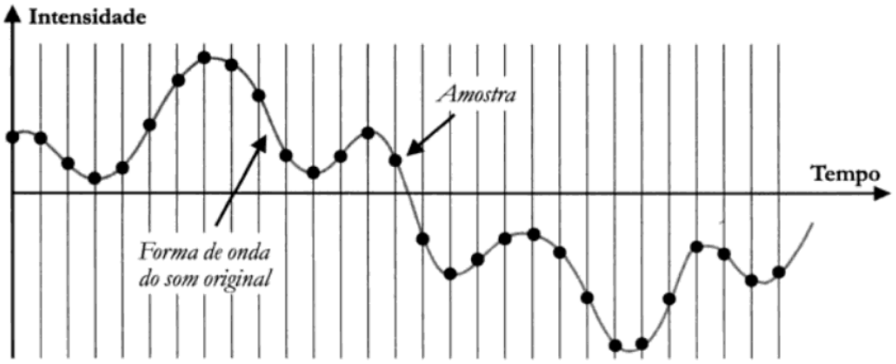


Figura 2.5: Amostragem de um sinal analógico. Fonte: (Zuben, 2004)

A Taxa de Amostragem é o número de amostras adquiridas de um sinal analógico em uma determinada unidade de tempo. Quanto maior a taxa de amostragem, melhor é a representação digital do som (Lathi e Green, 2018).

Na Quantização, o sinal amostrado é arredondado para o valor mais próximo dos níveis possíveis permitidos (ou níveis de quantização), resultando em um sinal digital. A resolução de um sinal digital é o número de bits usados para representar cada amostra. Quanto maior a resolução, mais precisa é a representação de cada amostra e menor a distorção do sinal. A Figura 2.6 ilustra a quantização do sinal amostrado (Lathi e Green, 2018).

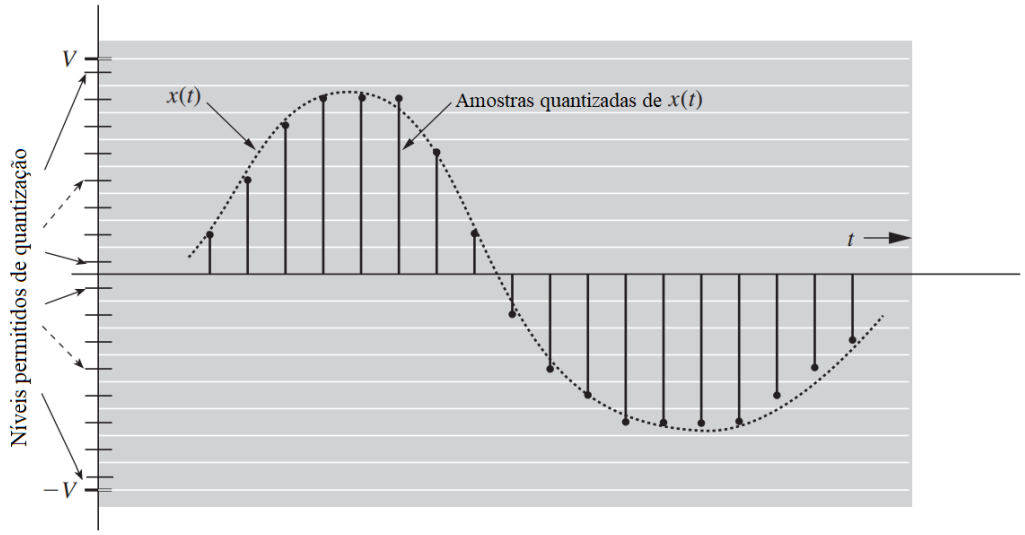


Figura 2.6: Quantização do sinal amostrado. Fonte: Adaptado de (Lathi e Green, 2018)

A Codificação é o processo pelo qual cada nível de quantização é transformado em uma sequência de "zeros" e "uns" (codificação binária) para ser processado pelo computador. A Figura 2.7 mostra a codificação do sinal quantizado (Lathi e Green, 2018).

Dígito	Equivalente binário	Forma de onda do pulso codificado
0	**0000**	
1	**0001**	
2	**0010**	
3	**0011**	
4	**0100**	
5	**0101**	
6	**0110**	
7	**0111**	
8	**1000**	
9	**1001**	
10	**1010**	
11	**1011**	
12	**1100**	
13	**1101**	
14	**1110**	
15	**1111**	

Figura 2.7: Codificação do sinal quantizado. Fonte: Adaptado de (Lathi e Green, 2018)

2.1.3 Análise Sonora

Existem várias formas de representar graficamente um sinal de áudio de forma a realizar sua análise, dentre os quais pode-se citar o oscilograma, análise do espectro de frequência e espectrograma.

Oscilograma

Oscilogramas são gráficos que representam a intensidade do sinal sonoro no tempo. Podem indicar os momentos de maior e menor intensidade, indicando quando alguém está falando, por exemplo. A imagem superior da Figura 2.8 ilustra um oscilograma (Ivo, 2004).

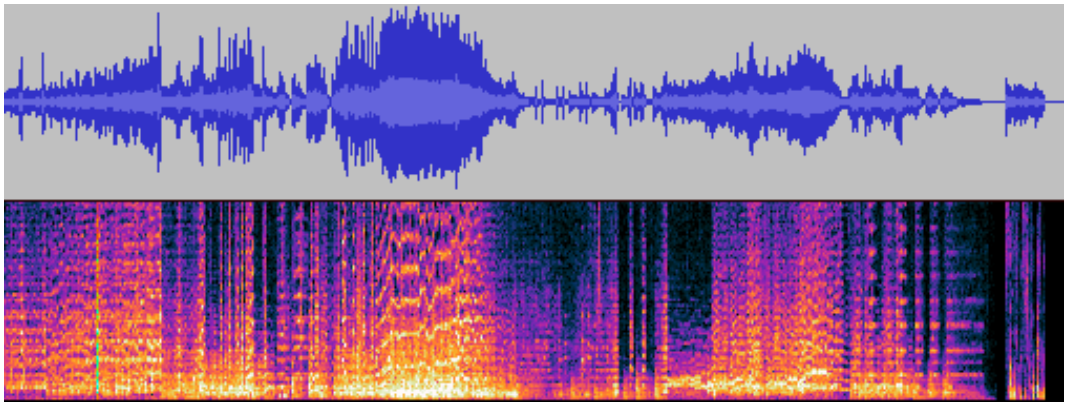


Figura 2.8: Oscilograma e Espectrograma sonoros de uma frase falada, obtidos a partir do programa Audacity. Fonte: Autoria própria

Análise do Espectro de Frequência

A representação espectral permite visualizar a composição de frequências de uma amostra, assim como a intensidade de cada frequência. A Figura 2.9 exibe a análise do espectro de frequência da palavra “Socorro” obtido a partir do programa Audacity (Ivo, 2004).

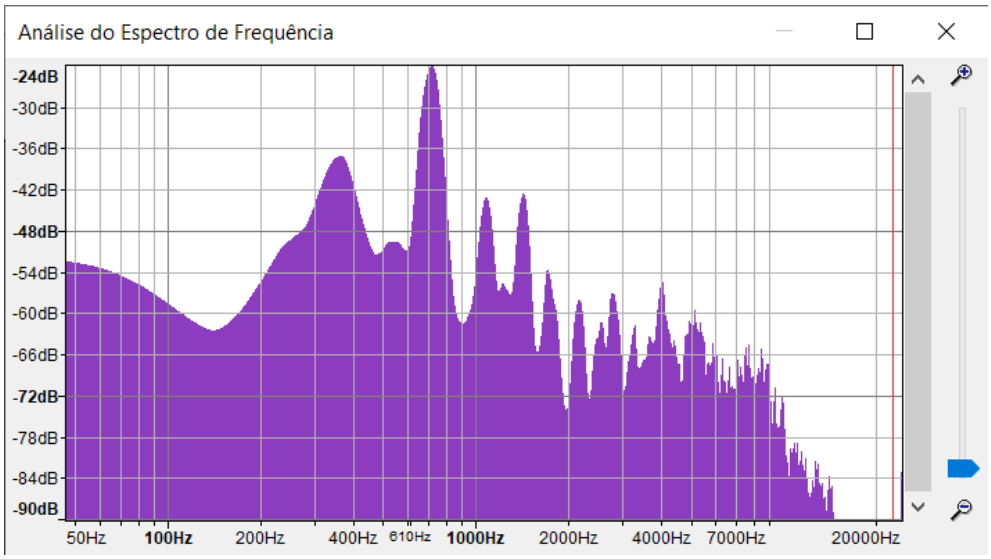


Figura 2.9: Análise do espectro de frequência da palavra "Socorro", obtido a partir do programa Audacity. Fonte: Autoria própria

Espectrograma

Espectrogramas são gráficos que mostram a densidade espectral de energia no plano tempo x frequência. Podem ser representados através de um gráfico de superfície, onde o valor da densidade espectral ocupa a terceira dimensão, mas são usualmente empregados de forma planar, onde são utilizadas diferentes cores para indicar a intensidade da densidade espectral de energia, variando do violeta ao vermelho do espectro visível, como visto na Figura 2.8. Também podem representar a variação de intensidade em tons de cinza, conforme mostra a Figura 2.10 (Ivo, 2004) (Valentim et al., 2010) (Rossing et al., 2014).

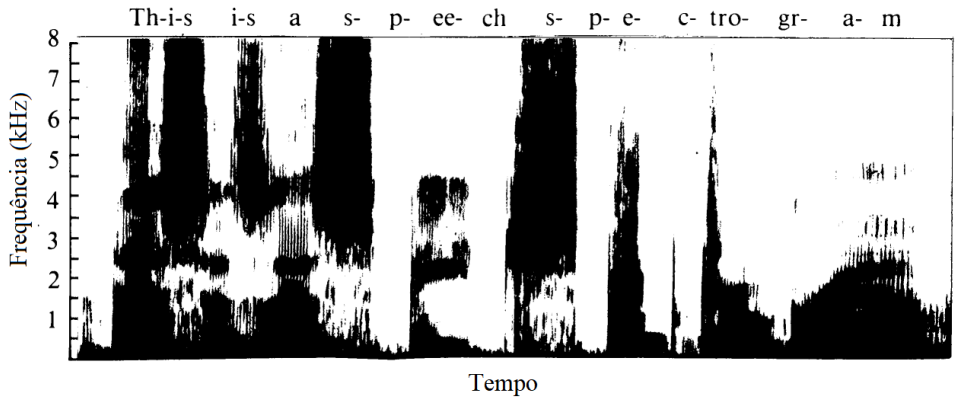


Figura 2.10: Espectrograma sonoro de uma frase falada. Fonte: Adaptado de (Rossing et al., 2014)

2.2 Protocolo TCP/IP

Ao planejar uma estrutura de rede para conectar dispositivos, os projetistas de rede organizam seus protocolos em camadas. O modelo em camadas reduz a complexidade, permitindo que um problema complexo seja dividido em problemas mais simples, além de ajudar que os detalhes da implementação sejam abstraídos nas várias camadas. Essa arquitetura padroniza as interfaces, facilitando mudanças em uma camada sem interferir nas demais. Nas seções a seguir, será apresentado um das camadas do modelo ISO, enfocado ao protocolo TCP/IP (Tanenbaum, 2011).

2.2.1 Modelo de referência OSI e TCP/IP

O modelo de camada OSI (*Open Systems Interconnection*) foi desenvolvido para simplificar o estudo de redes de computadores. Este modelo foi desenvolvido pela ISO (*International Standards Organization*) em 1983, como ponto de partida para a padronização internacional de protocolos. Também é conhecido como Modelo de Referência ISO OSI e trata da interconexão de sistemas abertos à comunicação com outros sistemas (Wetherall, 2011).

O modelo OSI tem sete camadas, implementadas tendo em mente as necessidades de abstração; além disso, as camadas devem desempenhar funções e ter limites bem definidos para minimizar o fluxo de informações entre as interfaces. A função de cada nível foi escolhida tendo em vista a definição de protocolos já padronizados internacionalmente e o número de níveis (que deve ser grande o suficiente para não ser necessário inserir diferentes funções na mesma camada e pequeno o suficiente para que a arquitetura não se torne altamente complexa para controlar (Wetherall, 2011)).

O modelo TCP/IP usa 4 das camadas do modelo padrão. Na Figura 2.11 temos a representação da camada de enlace de dados, camada de rede, camada de transporte e a camada de aplicação do modelo OSI. O protocolo TCP/IP utiliza protocolos específicos em suas camadas de rede (IP) e de transporte (TCP) (Peterson e Davie, 2013).

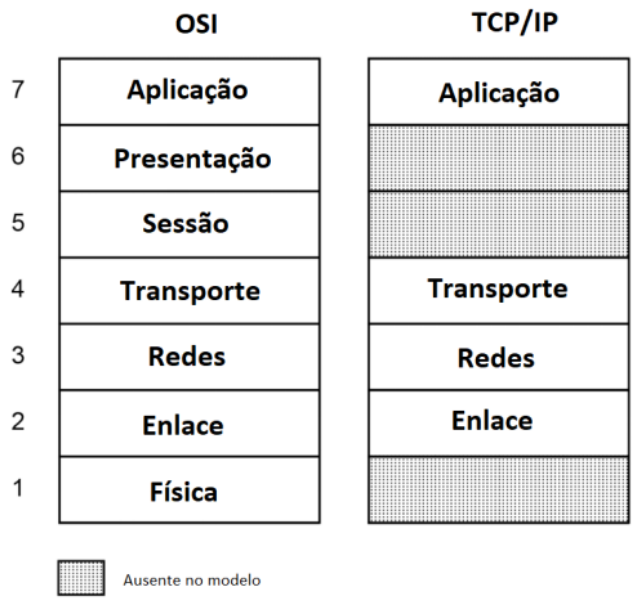


Figura 2.11: Modelos de referência OSI e TCP/IP. Fonte: Adaptado de (Tanenbaum, 2011)

Nas seções a seguir, serão discutidas brevemente as funcionalidades de cada camada do modelo de referência TCP/IP.

A Camada de Enlace de Dados

A camada de enlace de dados mascara os erros reais de transmissão, de forma que a camada de rede não perceba. Esse mascaramento ocorre porque o transmissor organiza os dados de entrada em blocos de *bits* chamados quadros (*frames*). Logo, a camada reconhece o início e o fim do *frame*, inserindo um padrão de *bits* antes e depois, ajudando a controlar ou não o sucesso da transmissão da informação, com o envio de volta de um *frame* de confirmação (Wetherall, 2011).

As funções e serviços da camada de enlace, dependem do protocolo a ela atribuído no enlace. Um exemplo é que alguns protocolos garantem a entrega do *frame* entre enlaces, isto é, desde o nó transmissor, passando por um único enlace, até o nó receptor. Os diferentes protocolos de enlace estão relacionados às diferentes tecnologias utilizadas no meio de transmissão ou *hardware*, como SONET (*Synchronous Optical Network*) relacionada ao meio de fibra óptica; e Ethernet relacionada ao uso de quatro pares trançados (RJ45) (Kurose e Ross, 2013).

As redes de *broadcast*[1] devem lidar, ainda, com o controle de acesso ao canal de dados que é compartilhado por diversas fontes de informação. Para tal tarefa foi criado o conceito de uma sub camada especial de enlace de dados, denominada MAC (*Media Access Control*) (Kurose e Ross, 2013).

A Camada de Rede

Uma rede de dispositivos conectados pode ser formada por diferentes tipos de enlaces e cabe a camada de rede mascarar a existência de tais enlaces. A camada de rede tem como função principal, concatenar enlaces logicamente, para formar rotas. Usa os serviços da camada de enlace para transportar pacotes de dados nas rotas. Ela provê endereços globalmente únicos e que normalmente estão relacionados à topologia da rede e podem fazer o controle de congestionamento de pacotes. A camada de rede ainda permite que redes (ou sub-redes) heterogêneas (com diferentes endereçamento e tamanho de pacotes) sejam conectadas. A Figura 2.12 mostra como os pacotes passam por essas rotas até alcançar o seu destino (Kurose e Ross, 2013).

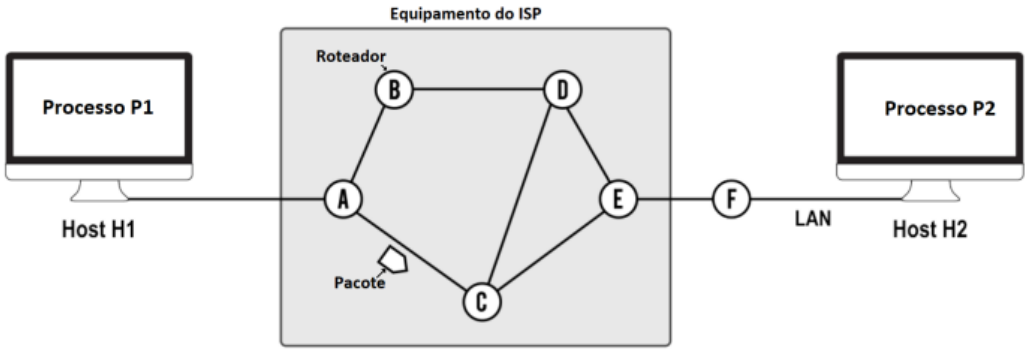


Figura 2.12: Rota percorrida pelo pacote, através dos roteadores, até chegar ao seu destino.
Fonte: Adaptado de (Caviglione et al., 2012)

A camada de rede, por sua vez, fornece serviços para a camada de transporte. Esses serviços devem ser independentes da tecnologia dos roteadores, que são os equipamentos que interligam as diferentes sub-redes. No serviço sem conexões, os pacotes são injetados individualmente na sub-rede e roteados independentemente uns

[1] *Broadcast* são redes que transmitem para todos os dispositivos conectados a rede simultaneamente.

dos outros. Nesse contexto, os pacotes são frequentemente chamados de datagramas e a sub-rede será denominada sub-rede de datagramas. O que ocorre nesse tipo de serviço é que a camada de transporte acrescenta um cabeçalho de transporte no início da mensagem e entrega o resultado à camada de rede (Kurose e Ross, 2013).

A camada de rede, se necessário for, divide a informação da camada de transporte em pacotes menores e os envia ao primeiro roteador que possui contato com a cadeia de roteadores que interligam a origem e o destino da mensagem (no caso da Figura 2.11 é o roteador A). A partir daí os pacotes estão sob o domínio do ISP (*Internet Service Provider*) que possui uma rede de roteadores interligados com diversos caminhos até o destino. Para escolher o melhor caminho que o pacote pode percorrer, existem inúmeros algoritmos que gerenciam as tabelas de rotas desses equipamentos, conhecidos como algoritmos de roteamento. Essas tabelas variam de pacote para pacote. Atendendo ao que está especificado no padrão ISO OSI, a camada de rede fica livre para estabelecer especificações detalhadas dos serviços que oferecerá à camada de transporte (Kurose e Ross, 2013).

A Camada de Transporte

A camada de transporte tem a função básica de aceitar dados de camadas superiores, dividindo-os em unidades menores (se necessário), e passando-os para a camada de rede. Ele também deve garantir que todas as partes da mensagem cheguem com sucesso ao seu destino. É a primeira camada do modelo que estabelece uma comunicação origem-destino. Além disso, permite que dois processos em computadores diferentes se comuniquem (Forouzan, 2013).

A camada de transporte é a última etapa de verificação de erros antes de entregar os dados à camada de aplicação. Ela é responsável por mover os dados, de forma eficiente e confiável, entre os processos executados em computadores conectados a uma rede de computadores, independentemente do tipo de rede subjacente. Deve ser capaz de regular o fluxo de dados e garantir confiabilidade, garantindo que os dados cheguem ao seu destino sem erros e na sequência em que foram iniciados.

Seu objetivo é fornecer serviços à camada superior que sejam confiáveis, eficientes e de baixo custo computacional. Por sua vez, a camada de transporte utiliza vários serviços oferecidos pela camada de rede (Kurose e Ross, 2013).

Assim como existem dois tipos de serviços de rede (com conexão e sem conexão), também existem dois tipos de serviços de transporte: orientados à conexão (as conexões têm três fases: estabelecimento, transferência de dados e encerramento) e sem conexão. A grande diferença entre os serviços da camada de rede e a camada de transporte é que o código da camada de transporte funciona inteiramente nos nós finais, e a camada de rede funciona principalmente por meio de roteadores (Wetherall, 2011).

Para estabelecer a conexão entre a origem e o destino na camada de transporte, foi criado um protocolo de conexão denominado *handshake*, ilustrado na Figura 2.12. O *handshake* de três etapas ocorre da seguinte forma: O cliente envia um sinalizador SYN para indicar uma solicitação de conexão TCP ao servidor. Se o servidor estiver em execução, ele oferece o serviço desejado e se puder aceitar a conexão de entrada, ele envia sua própria solicitação de conexão sinalizada por um novo SYN para o cliente e reconhece a solicitação de conexão do cliente com um sinalizador ACK. Tudo isso é feito em um pacote. Finalmente, se o cliente recebe os sinalizadores SYN e ACK e deseja continuar a conexão, ele envia um único ACK final ao servidor. Isso indica que o cliente recebeu a solicitação de conexão do servidor e a aceitou. Depois que o *handshake* de três etapas for executado dessa maneira, a conexão será estabelecida. Os dados agora podem ser trocados entre os dois nós (Kurose e Ross, 2013).

Existem duas maneiras de fechar a conexão previamente estabelecida. A primeira é quando um dos nós, cliente ou servidor, sinaliza, com uma mensagem com o *flag* END em 1, que deseja encerrar a sessão. O nó receptor retorna o sinal com um sinalizador ACK (ou seja, confirma o recebimento da solicitação). Isso termina apenas metade da conexão. Em seguida, o outro nó também deve enviar uma mensagem com o *bit* FIN definido como 1, e o nó receptor deve confirmar o recebimento.

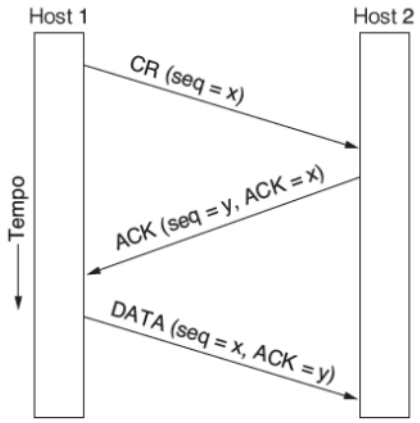


Figura 2.13: Estabelecimento do *handshake*. Fonte: (Wetherall, 2011)

O segundo método de término é um término abrupto da conexão. Isso é feito quando um nó envia o sinalizador RESET para o outro, indicando o desejo de encerrar a conexão abruptamente (Wetherall, 2011).

A Camada de Aplicação

A camada de aplicação é a camada responsável por entregar os serviços ao usuário final mais diretamente. Ela contém vários protocolos úteis para os usuários, como protocolos de transferência de arquivos, *login* remoto, *e-mail*, execução remota, HTTP (*HyperText Transfer Protocol*, que forma a base da *World Wide Web*), etc. Na indústria, os protocolos de aplicação geralmente estão vinculados aos projetos de um fabricante (protocolos proprietários) ou são utilizados padrões internacionalmente aceitos, dentre os quais se destaca o protocolo Modbus/TCP (Tanenbaum, 2011).

IP: O Protocolo da Camada de Rede

O protocolo *Internet Protocol* (IP), fornece um sistema de entrega ponto a ponto entre nós pertencentes a redes diferentes. É sem conexão, sem controle ou detecção de erros. Este protocolo possui algumas características, como a utilização do serviço que tenta o máximo possível entregar os dados (*best-effort*), sem garantia na entrega dos dados (não confiável). Além disso, permite que o tamanho dos datagramas (pacotes de dados) seja variável e fragmente os datagramas com sua posterior

recombinação no destino, o que garante suporte para redes intermediárias com diferentes MTUs (*Maximum Transmission Units*). Ele é responsável pelo roteamento dos datagramas (ou fragmentos dos mesmos) e provê o envio e recebimento de erros através de um protocolo chamado ICMP. É o protocolo IP responsável por transportar informações de uma ponta a outra na Internet, passando por várias redes potencialmente muito diferentes (Wetherall, 2011).

TCP: O Protocolo da Camada de Transporte

O protocolo TCP é um protocolo orientado à conexão, ou seja, estabelece uma conexão entre o transmissor e o receptor. Assim, cada endereço de rede para um nó na Internet consiste no endereço IP do nó e um número local de 16 *bits* para esse nó, chamado de porta, e fornecido pelo protocolo TCP. Para que o serviço TCP funcione, uma conexão deve ser explicitamente estabelecida entre uma máquina transmissora e uma máquina receptora. Além disso, todas as conexões TCP são *full-duplex*, ponto a ponto e orientadas a fluxo de *bytes*. O método básico de controle de fluxo de dados usado por entidades TCP é conhecido como ”janela deslizante”. Quando um segmento é enviado, o transmissor também inicia um temporizador local. Quando o segmento chega ao destino, a entidade TCP receptora retorna outro segmento (com ou sem dados, dependendo das circunstâncias) com um número de confirmação igual ao próximo número de sequência que espera receber. Se o temporizador do transmissor expirar antes de receber a confirmação, o segmento será retransmitido (Wetherall, 2011).

2.3 Aprendizagem de Máquina

Aprendizagem de Máquina - ou, em inglês, *Machine Learning* - é a área que estuda o desenvolvimento de algoritmos computacionais capazes de aprender com a experiência, de extrair conhecimento a partir de dados (Zhou, 2016) (Müller e Guido, 2016). Um algoritmo de aprendizagem detecta padrões significativos nos dados de experiência e utiliza-os para adquirir características destes dados (modelo

descritivo) e/ou realizar previsões em novas observações (modelo preditivo) (Zhou, 2016) (Alpaydin, 2014).

Diferentemente dos programas tradicionais, que implementam um conjunto de regras nos dados inseridos na entrada, os programas de Aprendizagem de Máquina obtêm, experimentalmente, a partir dos dados e de suas respectivas respostas, as regras que justificam os resultados apresentados. Estas regras podem então ser aplicadas em novos dados para se obter a resposta correspondente. A Figura 2.14 representa esta mudança de paradigma (Chollet, 2018).

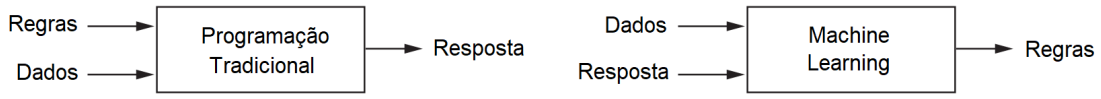


Figura 2.14: Mudança de paradigma na programação. Fonte: Adaptado de (Chollet, 2018)

A Aprendizagem de Máquina é especialmente útil na programação de programas muito complexos - como programas que queiram replicar habilidades humanas, como o reconhecimento de imagem - ou com necessidade de adaptabilidade - programas cujo comportamento deve variar conforme os dados de entrada, tal como um programa de reconhecimento de escrita à mão, que pode se adaptar às variações de caligrafia de diferentes usuários -, apresentando melhor desempenho que a programação direta da tarefa (Shalev-Shwartz e Ben-David, 2014).

2.3.1 Ramos de Aprendizagem de Máquina

As técnicas de Aprendizagem de Máquina podem ser classificadas como Supervisionadas, Não Supervisionadas e por Reforço. A Figura 2.15 apresenta esta classificação (Raghav Bali e Lesmeister, 2016).

A Aprendizagem Supervisionada se refere a algorítmos treinados com um conjunto de exemplos predefinidos, denominado *dataset* de treinamento, constituído de pares de entradas e suas respectivas saídas desejadas. O algorítmo de Aprendizagem Supervisionada usa o *dataset* de treinamento para gerar a função desejada, que é

então utilizada para mapear corretamente novos dados, conjunto denominado *dataset* de teste (Raghav Bali e Lesmeister, 2016).

Os principais tipos de algoritmos de Aprendizagem Supervisionada são os algoritmos de classificação e os algoritmos de regressão. Os algoritmos de classificação, ou classificadores, constroem modelos preditivos para atribuir classes às amostras, classificando-as conforme o *dataset* de treinamento. Já os algorítmos de regressão são utilizados para prever um número contínuo ou ponto flutuante baseado no *dataset* de treinamento (Raghav Bali e Lesmeister, 2016).

A Aprendizagem Não-Supervisionada, em contrapartida, é realizada a partir de um conjunto de dados constituído apenas de entradas, isto é, as saídas não são fornecidas. O algoritmo analisa estas entradas em busca de padrões, e realiza agrupamentos com base em características em comum. Posteriormente é realizado uma análise para determinar o que cada agrupamento representa no contexto em que o problema está inserido (Raghav Bali e Lesmeister, 2016).

Na Aprendizagem por Reforço, a aprendizagem é realizada por um agente tomador de decisões, que realiza ações em um ambiente, recebendo recompensas ou punições na tentativa de resolver o problema. Depois de *sets* de tentativa e erro, o algoritmo aprende a sequência de ações que maximiza o total de recompensas (Alpaydin, 2014).

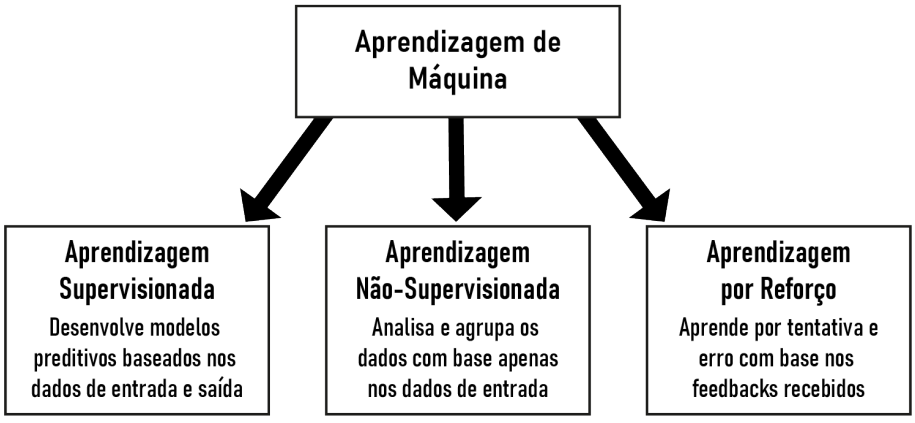


Figura 2.15: Classificação de Algoritmos de Aprendizagem de Máquina. Fonte: Autoria própria, com base em (Raghav Bali e Lesmeister, 2016)

2.3.2 Etapas de Aprendizagem de Máquina

Existem muitas formas de dividir as etapas da construção de um algoritmo de Aprendizagem de Máquina. Entretanto, a base do processo é constante, sempre constituída por Treinamento, Validação e Teste, como ilustrado na Figura 2.16.

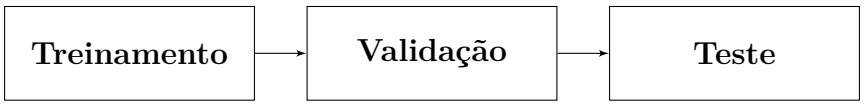


Figura 2.16: Processo de Aprendizagem de Máquina. Fonte: Autoria própria

Serão apresentadas as etapas definidas pela *Google Tech Cloud*, que expande o conceito base, adicionando etapas relativas à preparação do *dataset*, escolha do algoritmo, entre outros, totalizando 7 etapas (Guo, 2017).

Definição do Problema

Inicialmente, é necessário definir o problema a ser solucionado. Deve-se considerar as informações que se deseja prever e a disponibilidade dos dados para treinamento. Identificar o tipo de problema também é importante, uma vez que este pode determinar o tipo de algoritmo adotado (Guo, 2017).

Construção do *Dataset*

Uma vez definido o problema, é preciso coletar dados para treinar o algoritmo. A quantidade e qualidade dos dados coletados vão determinar a precisão das previsões realizadas pelo modelo. O algoritmo só conseguirá reconhecer os padrões que foi treinado para reconhecer, isto é, os padrões presentes no *dataset* de treinamento, de forma que este deve representar fielmente o sistema (Guo, 2017).

Preparação do *Dataset*

Nesta etapa, realiza-se a preparação do *dataset* para o uso no treinamento do algorítmo, aplicando ajustes e formatações nos dados coletados. Agrupa-se todos os dados, randomiza-se a ordem das amostras e, caso necessário, realiza-se modificações

adicionais, como a remoção de dados duplicados, correção de erros, conversão para o tipo de dado compatível, normalização dos valores, entre outros (Guo, 2017).

Também é nesta etapa que divide-se o *dataset* em duas partes: o *dataset* de treinamento, utilizado para treinar o algoritmo, e o *dataset* de validação, utilizado para validar o modelo treinado e avaliar seu desempenho. Não se deve utilizar os mesmos dados utilizados para o treinamento na validação, uma vez que estes dados já são conhecidos pelo modelo e o objetivo é avaliar sua capacidade de realizar previsões para dados novos (Guo, 2017).

Escolha do Algoritmo

De posse do *dataset* devidamente preparado, deve-se escolher um algoritmo de Aprendizagem de Máquina para solucionar o problema em questão. A seleção deve ser feita de acordo com o tipo de problema - como um problema de classificação, por exemplo, ou de agrupamento de dados -, tipo de dado - como números, texto, imagem, som -, aplicação desejada, taxa de complexidade, entre outros. A Seção 2.3.3 descreve alguns dos tipos de algoritmos de Aprendizagem de Máquina (Guo, 2017).

Treinamento

O treinamento é a base da Aprendizagem de Máquina. Nesta etapa, utiliza-se os dados de treinamento para melhorar, a cada iteração, a habilidade do modelo de realizar previsões (Guo, 2017).

O processo de treinamento envolve a inicialização dos parâmetros do modelo com valores randômicos, gerando uma previsão, que é então comparada com o valor esperado, por fim ajustando os parâmetros de forma a melhor descrever o sistema. O processo se repete continuamente, ajustando os parâmetros a cada interação, como ilustrado na Figura 2.17, a seguir (Guo, 2017).

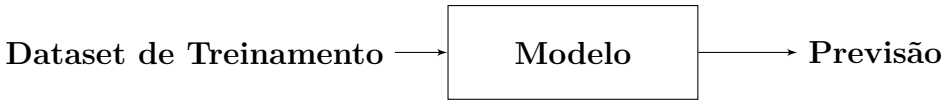


Figura 2.17: Processo de Treinamento de um Algorítmo de Aprendizagem de Máquina. Fonte: Autoria própria

Validação

Uma vez treinado, deseja-se validar o funcionamento do modelo e avaliar seu desempenho. Na validação, observa-se o comportamento do modelo com novos dados (*dataset* de validação), testando sua aptidão para realizar novas previsões (Guo, 2017).

Ajuste dos Parâmetros

A partir dos resultados obtidos na validação, pode-se desejar melhorar o desempenho do modelo. É nesta etapa que se realizam ajustes nos parâmetros do algoritmo, tais quais o número de *epochs* (interação sobre todas as amostras do *dataset*), taxa de aprendizagem, valores de inicialização, entre outros (Guo, 2017).

O ajuste destes parâmetros, também chamados de ”hiperparâmetros”, é um processo experimental que depende das especificidades do *dataset*, modelo e processo de treinamento (Guo, 2017).

Teste

Após ser utilizado para otimizar o algoritmo e definir o modelo, o *dataset* de validação se torna, efetivamente, parte do *dataset* de treinamento. Nesse contexto, para definir o desempenho do modelo final, deve-se utilizar um terceiro *dataset*, denominado *dataset* de teste ou *dataset* de publicação, com dados não utilizados nas etapas anteriores (Alpaydin, 2014).

Por fim, pode-se colocar o modelo em prática, utilizando-o na aplicação desejada.

2.3.3 Algoritmos de Aprendizagem de Máquina

Pode-se observar, na Figura 2.18, os diferentes tipos de algoritmos de Aprendizagem de Máquina posicionados em termos de performance e interpretabilidade.

A Regressão Linear é utilizada para modelar um conjunto de pontos em uma reta que me melhor se aproxima às amostras. A Árvore de Decisões é utilizada para realizar tomadas de decisão baseadas em condições. O algoritmo KNN (*K-Nearest Neighbors*) categoriza conjuntos de pontos em N classes distintas, de acordo com as condições de agrupamento. O algoritmo Random Forest realiza a criação de várias árvores de decisão, onde as variáveis são escolhidas randomicamente - diferentemente de uma árvore de decisão simples, onde todas as váriáveis são consideradas -, e, ao fim, todas as árvores são analisadas. Os métodos de Kernel são uma classe de algoritmos que analisa padrões, buscando relações entre os dados. As Redes Neurais Profundas - do inglês, *Deep Neural Networks* -, são algoritmos que tentam modelar sistemas complexos através de grafos de múltiplas camadas (Badillo et al., 2020).

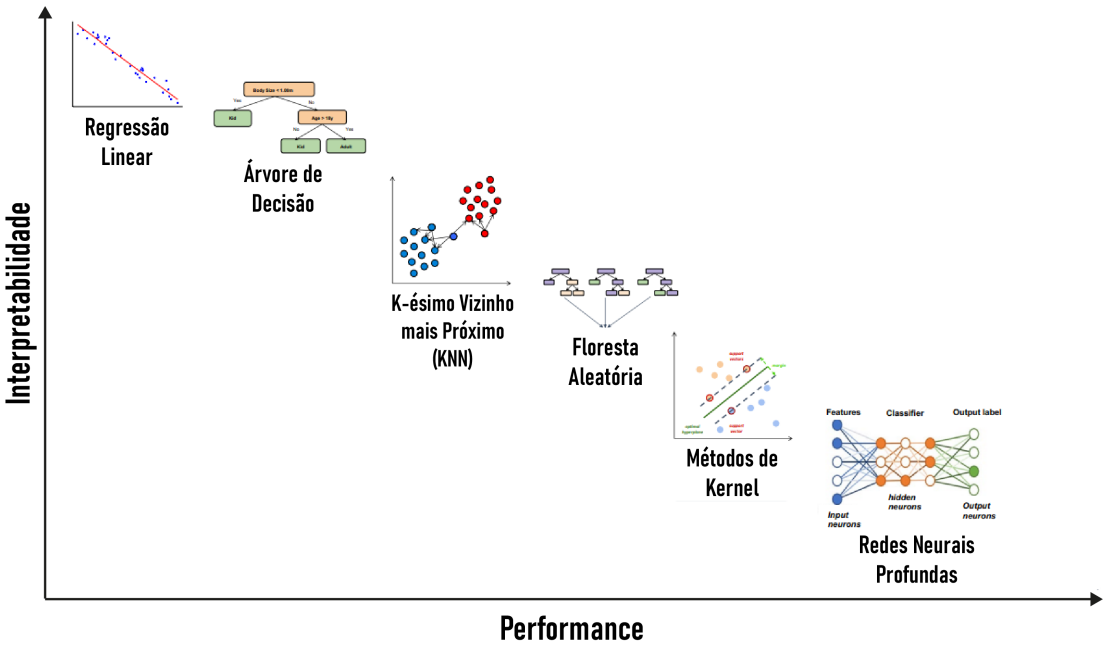


Figura 2.18: Visualização dos métodos de Aprendizagem de Máquina posicionados conforme interpretabilidade e performance. Fonte: Adaptado de (Badillo et al., 2020)

Capítulo 3

Metodologia

Este capítulo descreve de que forma o projeto será desenvolvido. Serão apresentados os componentes utilizados e suas especificações, assim como as etapas de execução do projeto.

3.1 Etapas de Execução do Projeto

De forma a alcançar os objetivos propostos, divide-se a execução do projeto nas seguintes etapas:

- Definir a placa de desenvolvimento, sistema de comunicação e sistema de detecção sonora para realizar os objetivos gerais;
- Desenvolver algoritmo de Aprendizagem de Máquina, para classificação de palavras faladas;
- Treinar o algoritmo utilizando um *dataset* de teste pré-existente;
- Validar o algoritmo por meio de testes;
- Criar arquivo JSON (*JavaScript Object Notation*) para o armazenamento de informações estruturadas, enviado entre cliente (computador) e servidor (Raspberry Pi 4);

- Criar arquivo em Python para a codificação e decodificação das variáveis utilizadas para arquivo JSON;
- Desenvolver a interface gráfica de usuário para o sistema, utilizando o programa QtDesigner;
- Importar a interface gráfica no programa Python, programando as interações com os elementos gráficos;
- Gravar as amostras de áudio das palavras "socorro" e "ajuda" para a construção do *dataset*;
- Preparar o *dataset* através de ajustes e formatações nos dados coletados;
- Treinar o algoritmo de Aprendizagem de Máquina com o *dataset* construído;
- Realizar testes no algoritmo, analisando a taxa de acerto para validação dos parâmetros;
- Implementar modificações no *dataset* e nos parâmetros, de acordo com os resultados obtidos, retreinando o algoritmo e realizando novos testes. Esta etapa se repete até a obtenção de resultados satisfatórios;
- Compilar programa do servidor na Raspberry Pi 4;
- Realizar testes no algoritmo final na Raspberry Pi com diferentes vozes, mensurando acurácia e taxa de acerto;
- Analisar os resultados e validar o sistema.

3.2 *Hardware*

Para determinar o *hardware* a ser utilizado, é preciso analisar características do projeto que possam ser relevantes para esta escolha, tais como consumo de energia, utilização de periféricos, velocidade e capacidade de processamento, tamanho da memória, entre outros. De acordo com os objetivos pré-estabelecidos, deseja-se uma

placa que permita implementar protocolos de comunicação, que seja compatível com periféricos de áudio e que possua uma boa capacidade de processamento, para que o algoritmo de reconhecimento sonoro possa ser executado em um tempo satisfatório.

3.2.1 Microcontroladores e Microprocessadores

Entre os possíveis microcontroladores disponíveis no mercado, pode-se dizer que as placas com microcontroladores ATmega, como a família de placas Arduíno, são as mais utilizadas em projetos de protótipos, o que garante uma vasta coleção de artigos de referência e material de consulta. Embora este tipo de microprocessador seja compatível com módulos de áudio e módulos RJ45, a capacidade de processamento é limitada, principalmente quando se deseja executar algoritmos de Aprendizagem de Máquina e *multithreading*.

A partir desta limitação, buscou-se no mercado opções com maior nível de processamento. Sistemas *System-on-Chip*[1] demonstraram melhor desempenho que microcontroladores, por conter mais de uma CPU e suporte a protocolos e conexões avançadas (Ethernet, USB, HDMI, Wi-Fi, entre outros). Estudou-se três opções de microprocessadores baseados em Linux; FPGAs SoC, BeagleBones ou Raspberry PI.

A Tabela 3.1 apresenta uma comparação entre as placas mencionadas, descartando as FPGAs devido ao alto custo e sobre-dimensionamento para a aplicação final.

[1] *System-on-Chip* é um circuito integrado que reúne variados componentes complexos - tais como processadores, memórias e uma arquitetura de comunicação entre estes elementos - dentro de um único chip (Pasricha e Dutt, 2008)

Nome	Arduíno Uno	Raspberry Pi	BeagleBone
Modelo Testado	R3	Pi 4 - Modelo B	Rev A5
Preço (US$)	$29,95[2]	$35,00	$89,00
Processador	ATMega 328	ARM11	ARM Cortex-A8
Número de Núcleos	1	4	1
Freq. de Clock	16MHz	1.5GHz	700MHz
RAM	2KB	256MB	256MB
Flash	32KB	(SD Card)	(microSD)
EEPROM	1KB	-	-
Tensão de Entrada	7-12V	5V	5V
Corrente Mínima	42mA	540mA [3]	170mA
Potência Mínima	0.3W	3.5W	0.85W
GPIO Digital	14	8	66
Analog Input	6 10-bit	N/A	7 12-bit
PWM	6	-	8
TWI/I2C	1	1	2
SPI	1	1	1
UART	1	1	5
DEV IDE	Arduino Tool	IDLE, Scratch Squeak/Linux	Python, Scratch, Squeak, Cloud9/Linux
Ethernet	N/A[4]	10/100	10/100
USB Master	N/A	2 USB 2.0	1 USB 2.0
Video Out	N/A	HDMI, Composite	N/A
Audio Output	N/A	HDMI, Analog	Analog

Tabela 3.1: Tabela Comparativa entre um Microcontrolador e Microprocessadores. Fonte: Autoria própria

Os microprocessadores podem ser comparados a computadores de uso geral em

[2]Preço médio relativo à placa Arduíno Uno original. Foram encontradas placas que buscam replicar o Arduíno Uno, similares em design e funcionalidade, por US$4,00.

[3]Fonte: (Geerling, 2023)

[4]Sem o uso de shields.

termos da ampla gama de aplicações que podem atender, devido a sua elevada capacidade de processamento. Os microprocessadores requerem periféricos em torno do processador, tais como memória RAM (Memória de Acesso Randômico), placa de vídeo e memórias para armazenamento de dados, diferentemente de microcontroladores, que englobam estes chips adicionais em um único chip.

Alguns fatores fizeram com que a Raspberry Pi fosse selecionada para a realização do projeto. O primeiro é que as especificações da placa são compatíveis com as características do projeto, uma vez que oferece alto poder de processamento, permite o uso de várias linguagens de programação e há disponibilidade para dispositivos de entrada. De forma complementar, em termos de comunidade de desenvolvimento, há uma grande quantidade de desenvolvedores que trabalham com a Raspberry Pi, que fornecem suporte, bibliotecas e diversos exemplos. Além disso, por poder operar com um sistema operacional (usualmente, Linux), os recursos são vastos para o desenvolvimento. Usar a Raspberry Pi também garante maior liberdade na escolha do periférico captador de áudio, uma vez que garante-se compatibilidade com microfones de diferentes tipos de entradas (GPIO, USB e P2) sem o uso de um adaptador, que seria uma possível fonte de ruído.

Raspberry Pi 4 Model B

A *Raspberry Pi 4 Model B* faz uso de um SoC Broadcom BCM2711B0, que apresenta quatro núcleos da Unidade Central de Processamento (CPU) ARM Cortex-A72 de 64 *bits* e frequência de *clock* de 1.5GHz, e uma Unidade de Processamento Gráfico (GPU) Broadcom VideoCore VI de 500MHz de *clock* (Pasricha e Dutt, 2008) (Halfacree, 2020).

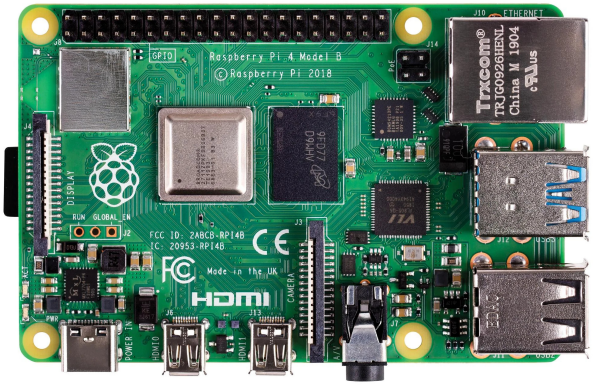
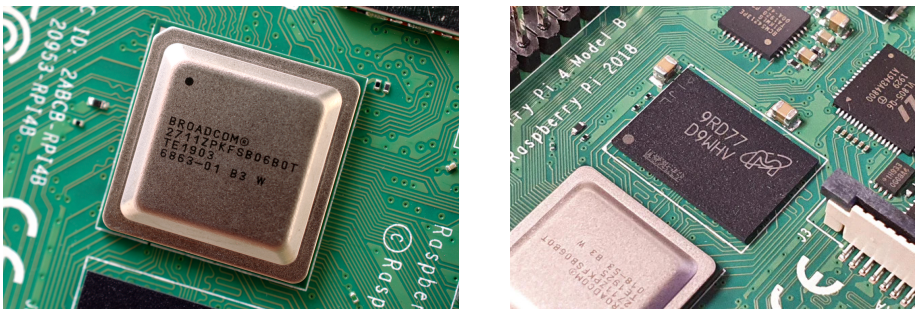


Figura 3.1: Visão superior da placa *Raspberry Pi 4 Model B*. Fonte: (Halfacree, 2019)

O *System-on-chip* é conectado a uma RAM LPDDR4 (*Low-Power Double-Data-Rate 4*) de 4GB com um *clock* de 3200MHz. Além disso, as opções de conectividade são; Wi-Fi 2,4/5,0 GHz IEEE 802.11.b/g/n/ac, Bluetooth 5.0, *Gigabit Ethernet*. Com relação a portas e conectores, inclui 40 pinos GPIO, 2 portas micro-HDMI 2.0, 2 portas USB 3.0s, 2 portas USB 2.0, Conector de 3.5mm para saída de áudio, Interface para camera (CSI) e Interface para *display* (DSI) (Raspberry, 2019).



(a) **(b)**

Figura 3.2: *Raspberry Pi 4 Model B*. (a) *System on Chip*. (b) Memória RAM. Fonte: (Halfacree, 2019)

3.2.2 Microfone

O microfone escolhido para o projeto foi o *GXT 232 Mantis*, da marca *Trust*, como pode ser visto na Figura 3.3. Se trata de um microfone de conexão USB (compatível com a Raspberry Pi 4) e padrão de captação omnidirecional, isto é, capta o som de forma aproximadamente igual de todas as direções. O microfone é do tipo condensador, o que significa que possui um diafragma que é fino e condutivo,

situado próximo a uma fina placa de metal. O sistema funciona como um capacitor eletrônico, onde a pressão sonora faz o diafragma vibrar, causando uma variação na capacitância que pode ser detectada por um circuito eletrônico, convertendo assim a energia de aúdio em energia elétrica. Devido à sua alta sensibilidade, o som captado deste tipo de microfone é o mais fiel ao original (Trust, 2023) (do Valle, 2015).

Figura 3.3: Microfone *GXT 232 Mantis*. Fonte: (Trust, 2023)

O modelo possui resposta em frequência no intervalo 50 Hz - 16000 Hz, impedância de 32 Ohm, sensibilidade de -38dB e conta com redução de ruído e redução de eco, conforme os dados apresentados na Tabela 3.2 (Trust, 2023).

Resposta em Frequência	50 Hz - 16000 Hz
Impedância	32 Ohm
Sensibilidade	-38 dB
Tipo de Sensor	Condensador
Padrão de Captação	Omnidirecional

Tabela 3.2: Especificações do Microfone *GXT 232 Mantis*. Fonte: (Trust, 2023)

3.2.3 Configuração Final

Para a definição do protocolo de comunicação, analisa-se as características do projeto. Embora não haja demanda de altas taxas de transmissão, é significativo efetuar confirmação de pacotes, de forma que define-se o protocolo TCP/IP como forma de comunicação.

O sistema final, portanto, é constituído pelo computador, Raspberry Pi e microfone. O computador, cliente, se comunica com a Raspberry Pi, servidor, através do protocolo TCP-IP, e a Raspberry Pi realiza a captação do áudio através do microfone a ela acoplado. A Figura 3.4 apresenta a configuração final do sistema.

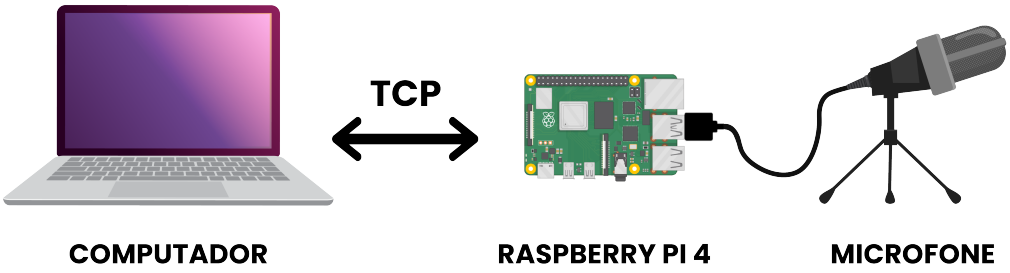


Figura 3.4: Configuração final do sistema. Fonte: Autoria própria

3.3 *Software*

O desenvolvimento do projeto a nível de *software* requer algumas etapas de preparação, entre as quais pode-se citar a definição da linguagem de programação, pesquisa de possíveis bibliotecas e validação de cada biblioteca apta para a linguagem escolhida.

Na definição da linguagem de programação, descartaram-se linguagens *single thread*, como o Javascript, devido à necessidade de executar *threads* em paralelo, para que múltiplas comunicações possam ocorrer simultaneamente. As linguagens C++ e Python foram consideradas para o desenvolvimento do projeto, por apresentarem bibliotecas para o desenvolvimento de projetos com Aprendizagem de Máquina, interfaces gráficas e diversos tipos de comunicação (Müller e Guido, 2016).

Python é uma *interpreted language*, o que significa que seu código-fonte é convertido em um *bytecode*[5] e precisa ser executado por uma *Parallel Virtual Machine* (PVM) de Python. Essa estrutura difere das estruturas de C e C++, classificadas

[5]O bytecode é um código de programação orientada a objeto, compilado para executar em uma máquina virtual (VM) em vez de uma unidade de processamento central (CPU) (Chen et al., 2014).

como *compiled language*, nas quais o compilador traduz o código diretamente para um executável a ser executado na máquina local (Power e Rubinsteyn, 2013). Devido à simplicidade da linguagem Python, garantindo maior facilidade na programação do sistema, e à quantidade de *frameworks* e bibliotecas voltadas para Aprendizagem de Máquina e manipulação de áudio, a linguagem Python foi escolhida para a programação do projeto (Müller e Guido, 2016).

As seguintes seções apresentarão as bibliotecas utilizadas, o funcionamento do sistema a nível de *software*, a preparação dos *datasets* de treinamento, o algoritmo de Aprendizagem de Máquina, a definição dos pacotes de comunicação e a implementação da interface gráfica.

3.3.1 Diagrama de Funcionamento

A Figura 3.5 apresenta o diagrama de funcionamento do sistema. O computador, que atua como cliente, se comunica com a Raspberry Pi, que atua como servidor, através da comunicação TCP-IP. O cliente utiliza a interface gráfica para enviar comandos ao servidor, enviados através de pacotes de comunicação. O servidor gerencia as informações que chegam dos pacotes e realiza os comandos recebidos para iniciar e parar a recepção de áudio, fazendo uso do microfone acoplado à Raspberry Pi. Em seguida, realiza o processamento do áudio captado, utilizando o algoritmo de Aprendizagem de Máquina e, por fim, envia o resultado obtido de volta ao cliente.

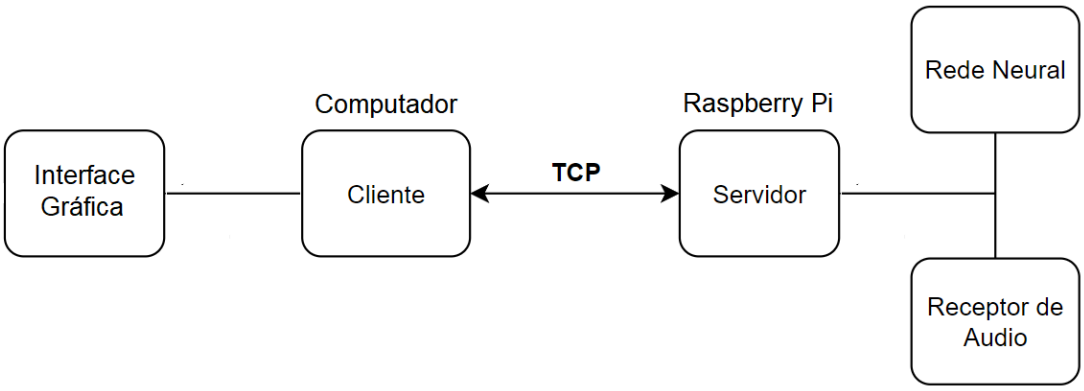


Figura 3.5: Diagrama esquemático do sistema. Fonte: Autoria própria

3.3.2 Bibliotecas

As próximas seções serão destinadas à explicação e detalhamento de todas a bibliotecas que auxiliaram na criação do código final.

NumPy

NumPy é tido como um dos pacotes fundamentais para computação científica em Python. Ele contém funcionalidade para matrizes multidimensionais, funções matemáticas especiais, como operações de álgebra linear e a transformada de Fourier, e geradores de números pseudo-aleatórios. No código criado, utiliza-se o Numpy para arredondar e limitar as casas decimais, entre outras aplicações (Müller e Guido, 2016).

JSON

JSON (*JavaScript Object Notation*) é um formato para a representação de dados estruturados. É um formato leve para transferências de dados e de intuitiva compreensão. Esta estrutura é fundamental no projeto, por ser utilizada para estabelecer os parâmetros iniciais da interface gráfica, assim como os pacotes de dados enviados pela comunicação TCP (Müller e Guido, 2016).

Logging

Logging é uma classe de Python para o gerenciamento de logs de código, informações usadas para identificar o estado atual do sistema, incluindo erros, problemas ou pequenos avisos, assim como registros da data e hora que acontecem. Em alguns casos é possível identificar a origem, o usuário, o endereço IP, entre outros campos (Müller e Guido, 2016).

Librosa

Librosa é um pacote Python para manipulação de música e análise de áudio. Ele fornece os métodos necessários para criar sistemas de recuperação de informações

musicais e contém um conjunto de algoritmos para processamento e análise de sistemas sonoros (Raguraman et al., 2019). No projeto, é utilizado para a conversão da taxa de amostragem de amostras do *dataset*.

Datetime

A classe *Datetime* tem o propósito de obter a data e hora atual da máquina que está em processo. Esta classe é utilizada para o armazenamento de dados e gerenciamento dos logs criados pelo código (Müller e Guido, 2016).

Threading

A técnica de *multithreading* permite o processamento simultâneo de várias instâncias de código, independentes, chamadas de *threads*. Uma *thread* explícita pode ser definida pelo programador e gerenciado pelo sistema operacional, de outra forma uma *thread* implícita é gerada estaticamente pelo compilador quando se detecta que são instruções contíguas. No projeto foi necessário a utilização de *muiltithreads* em dois momentos, na instância de código que está o SOS *Server* e no SOS *Client*, em ambos os casos foi necessário que toda a comunicação fosse gerenciada por *threads* explícitas (Müller e Guido, 2016).

Socket

Os *Sockets* são as extremidades de um canal de comunicação bidirecional, que podem se comunicar dentro de um processo, na mesma máquina ou em máquinas diferentes. Podem ser implementados através de um número diferente de canais e, dependendo de sua configuração, podem atuar com diferentes protocolos, como UNIX, TCP, UDP, etc. A biblioteca de *Socket* de Python fornece classes específicas para lidar com o transporte de dados, possibilitando diferentes configurações de rede. No projeto, a classe *Socket* foi utilizada na comunicação de dados, conforme detalhado na Seção 3.3.4 (Müller e Guido, 2016).

Pytorch

Pytorch é uma biblioteca com estruturas para aplicações de Aprendizagem de Máquina, visão computacional e processamento. É de código aberto baseada na linguagem de programação Python e na biblioteca *Torch*. No projeto, ela foi utilizada para toda a base do algorítimo RNN (Müller e Guido, 2016).

PyQT

PyQt é a biblioteca de Python para Qt, um conjunto de bibliotecas e ambientes de desenvolvimento de C++ que possibilita a criação de interfaces gráficas. Também fornece ferramentas para rede, encadeamento, expressões regulares, bancos de dados SQL, SVG, OpenGL, XML e muitos outros recursos. No projeto, utiliza-se Qt para a criação da interface gráfica, posteriormente descrito na Seção 3.3.5 (Müller e Guido, 2016).

FFmpeg

O FFmpeg é um programa multiplataforma em linha de comando, para a gravação e manipulação de arquivos de áudio e vídeo. O programa permite a codificação e decodificação de arquivos, conversão para diversas extensões e formatos, entre outros. No projeto, o FFmpeg é utilizado para a conversão das amostras do *dataset* para o formato WAV, e para a mudança da taxa de amostragem para 16kHz (FFmpeg, 2008).

3.3.3 Aprendizagem de Máquina

O algoritmo foi construído conforme os passos descritos na Fundamentação Teórica. Analisando o problema a ser resolvido, conforme os objetivos estabelecidos, deseja-se que o algoritmo seja capaz de identificar as palavras “socorro” e “ajuda” em amostras sonoras, realizando a classificação do áudio nas classes “socorro”, “ajuda” e “palavras não encontradas”.

Se trata, portanto, de um problema de Aprendizagem Supervisionada, onde o algoritmo é treinado com um conjunto de exemplos predefinidos, neste caso, exemplos de amostras de áudio e suas respectivas classificações.

Consultando a disponibilidade dos dados, pôde-se verificar que *datasets* sonoros com as palavras estabelecidas se encontram indisponíveis nos bancos de dados de referência, como o *Kaggle* e o *UC Irvine Machine Learning Repository*. Nesse contexto, o *dataset* foi construído através da gravação individual das amostras, utilizando o microfone GXT 232 Mantis, cujos dados foram detalhados na Seção 3.2.2.

Foram gravadas 124 amostras para "ajuda" e 120 amostras para "socorro", todas no formato M4A. As amostras de áudio gravadas possuem frequência de amostragem de 48kHz, taxa de amostragem de 16 *bits* e monocanal, informações obtidas através do programa Winamp, conforme pode ser observado na Figura 3.6, a seguir.

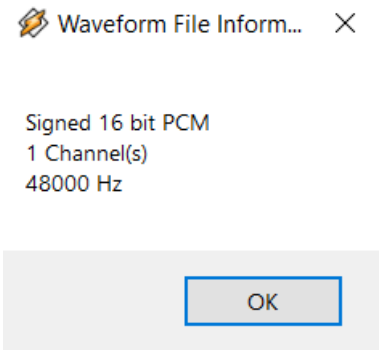


Figura 3.6: Visualização das características de áudio das amostras gravadas. Fonte: Autoria própria

Foram utilizadas as vozes de 4 pessoas para a gravação das amostras, de forma a garantir quatro timbres distintos, e procurou-se realizar variações de altura na voz - isto é, variando de pronuncias mais agudas à mais graves -, assim como variações na intensidade da fala e variações de distância ao microfone, de forma a garantir diversidade das amostras no *dataset*.

De posse das amostras gravadas, realizou-se a preparação do *dataset* para seu uso no algoritmo. Os arquivos gravados na extensão M4A foram convertidos para o formato WAV através da biblioteca ffmpeg do programa Audacity, pelo próprio *prompt* de comando, conforme mostra a Figura 3.7. Não foi necessário realizar

uma mudança na quantidade de canais, uma vez que as amostras obtidas já eram monocanais. Os arquivos também foram separados por pastas, de acordo com sua classificação.

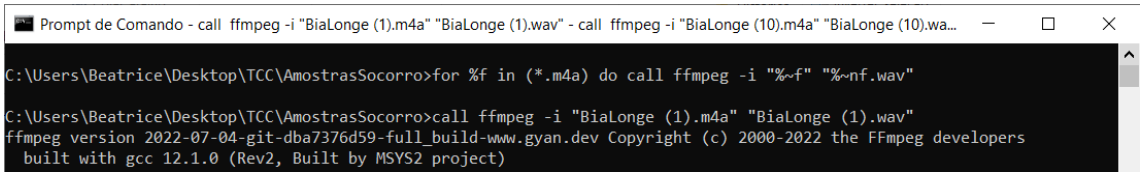


Figura 3.7: Conversão do dataset da extensão .M4A para a extensão .WAV. Fonte: Autoria própria

De acordo com o teorema de Nyquist, a taxa de amostragem tem que ser maior que o dobro da frequência do sinal amostrado para que não ocorra perda de informações do sinal original. Como sinais de voz estão no espectro de baixa frequência, de até 4kHz, deve-se garantir uma frequência de amostragem de pelo menos 8kHz (Rossing et al., 2014). Em contrapartida, embora frequências de amostragem altas garantam menos perdas de informação, elas demandam um maior custo computacional.

Nesse contexto, optou-se por reduzir a taxa de amostragem para 16kHz, sem que haja perda de informação, por obedecer a taxa de Nyquist e garantindo um processamento mais rápido. A conversão da taxa de amostragem de 48kHz para 16kHz, foi realizada também através da biblioteca ffmpeg. A Figura 3.8 ilustra o comando para a conversão, realizada através do Prompt de comando.

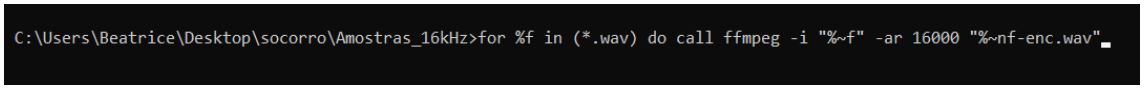


Figura 3.8: Conversão do *dataset* da taxa de amostragem 48kHz para 16k Hz. Fonte: Autoria própria

Uma vez preparado o *dataset*, realiza-se a escolha do algoritmo. Sabe-se que se trata de um algoritmo preditivo, portanto deve-se utilizar Aprendizagem Supervisionada. Por ser um problema de classificação, decide-se utilizar Redes Neurais Recorrentes (RNN).

O treinamento foi realizado utilizando a proporção de 70% das amostras para

o *dataset* de treinamento, 25% para o *dataset* de validação e 5% para o *dataset* de teste. Foram utilizadas 3 *layers* e 15 *epochs*.

Realiza-se, então, testes de validação no *dataset* treinado. A partir de seus resultados, ajusta-se os parâmetros do algoritmo, até a obtenção de resultados satisfatórios, mensurados pela taxa de acerto e acurácia.

Por fim, realiza-se testes finais para avaliar o desempenho do programa. São realizados 20 testes para cada palavra, e duas vozes, uma que foi utilizada para treinar o algoritmo e uma que não.

3.3.4 Comunicação

O servidor, nesse contexto, é uma habilitação de rede que, dependendo de suas configurações, possibilita receber conexão de outros sistemas, denominados clientes. Para a criação do servidor, inicialmente criou-se o objeto de programação com a configuração inicial para trabalhar com o protocolo TCP/IP, habilitando qualquer direcionamento de IP e uma porta de rede específica. A classe foi declarada como *thread*, de forma que a inicialização da *thread* coloca o objeto em modo de espera para a conexão com outros dispositivos. No Apêndice A.1, pode-se consultar a parte do código do servidor relativa à comunicação.

Na programação do cliente, utiliza-se a mesma porta de comunicação configurada no servidor, além de configurar a IP para a IP atual do servidor. Caso o servidor estiver conectado à rede local, pode-se utilizar a IP de *loopback* "127.0.0.1". Após a configuração do cliente e a execução do servidor, que fica em modo de espera, o cliente pode enviar solicitações para estabelecer a conexão. O código do cliente pode ser visto no Apêndice A.2.

Uma vez estabelecida a comunicação, ambos os sistemas ficam em espera de um evento na interface de usuário. Quando o evento ocorre, o cliente registra a informação recebida, realiza o empacotamento dos estados das variáveis do sistema em formato JSON e envia este pacote através da comunicação TCP/IP. O servidor, ao receber o pacote, computa a informação e inicia a execução de acordo com o que

foi recebido. Pode-se observar um exemplo do JSON gerado no Código 3.2.4.1.

```
{
  "SendtoServerParameters": {
    "conectar_servidor": false,
    "gravar_auto": false,
    "gravar_manual": false,
        "SettingsParameters": {
      "nome_usuario": "Beatrice",
      "email_usuario": "exemplo@gmail.com",
            "telefone_usuario": "819XXXX-XXXX",
            "notif_email": false,
      "notif_Telegram": false
    }
  },
  "ReceiveFromServerParameters": {
    "conectado_servidor": false,
    "AutoParameters": {
      "gravando_auto": false,
      "counter_time_voice_auto": 0,
      "voice_intensity_auto": 0
    },
    "ManualParameters": {
      "gravando_manual": false,
      "counter_time_voice_manual": 0,
      "voice_intensity_manual": 0
    },
    "PrecisionParameters": {
      "prob_socorro": 0,
      "prob_ajuda": 0,
      "prob_nenhuma": 0
    },
    "IdentificationParameters": {
      "palavra_identificado": false,
      "socorro_identificado": 0,
      "ajuda_identificado": 0
```

```
35      }
36    }
37 }
```

Código 3.3.4.1: Exemplo de um possivel pacotes de dados. Fonte: Autoria própria

Para realizar a comunicação por arquivo JSON, criou-se o programa "ClassSettings.py", apresentado no Apêndice A.3, que atua como um codificador e decodificador para este tipo de arquivo. O programa realiza a leitura do arquivo JSON, importando os estados das variáveis para o código, assim como a geração do arquivo JSON, exportando os estados das variáveis.

3.3.5 Interface Gráfica do Usuário

A Interface Gráfica do Usuário - do inglês *Graphical User Interface* (GUI) - é um modelo que permite a interação do usuário com dispositivos digitais através de elementos gráficos, tais como janelas, menus, ícones e botões. Dessa forma, as interações que se davam exclusivamente por linhas de comando passam a ser realizadas por uma interface visual, tornando o uso do sistema intuitivo ao usuário (Myers, 2004).

Para o desenvolvimento da GUI, optou-se por utilizar Qt, um sistema multiplataforma para a criação de interfaces gráficas, e PyQt, um empacotador da linguagem Python para a biblioteca Qt, tornando o código da interface compatível com a linguagem Python, utilizada no resto do código.

Embora seja possível criar a interface gráfica em PyQt codificando manualmente em código Python, optou-se por utilizar o editor gráfico de interfaces Qt Designer, cujo ambiente de desenvolvimento pode ser observado na Figura 3.9. Nesta plataforma, é possível criar componentes gráficos e organizá-los em uma interface usando diferentes gerenciadores de *layout*.

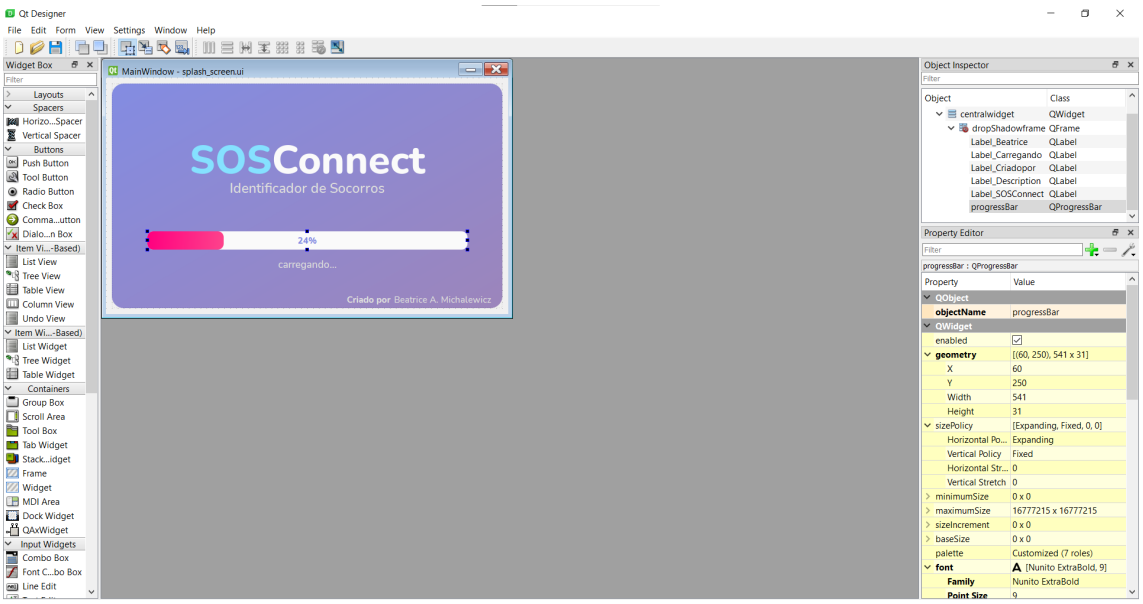


Figura 3.9: Ambiente de desenvolvimento do QtDesigner. Fonte: Autoria própria

Foram desenvolvidas 3 janelas no Qt Designer: a Tela de Carregamento, Menu Principal e Menu de Configurações. Optou-se pela criação da Tela de Carregamento, ilustrada na Figura 3.10, para ilustrar visualmente ao usuário que o programa está sendo inicializado, evitando a impressão que o sistema não está sendo responsivo no período de carregamento.

Figura 3.10: Tela de carregamento do programa SOS Connect. Fonte: Autoria própria

O Menu Principal é a tela que comporta as funcionalidades do programa, onde se realizarão as interações relativas à detecção sonora (Figura 3.11).

Inicialmente, para realizar a conexão com o servidor, deve-se clicar no botão "CONECTAR", na seção "SOS Server". Uma vez conectado, será possível iniciar a

detecção nos modos Automático e Manual, pressionando “INICIAR” em suas respectivas seções. O resultado da classificação - ’Socorro Identificado’, ’Ajuda Identificado’ ou ’Não Identificado’ - poderá ser observado no lado direito da tela, assim como as probabilidades de cada categoria.

Adicionalmente, o ícone de engrenagem irá abrir o Menu de Configurações e o ícone de “x” realizará a finalização do programa e subsequente fechamento da janela.

Figura 3.11: Menu Principal do programa SOS Connect. Fonte: Autoria própria

O Menu de Configurações, exibido na Figura 3.12, permite executar a alteração no nome exibido no Menu Inicial e atualizar as informações referentes às notificações de socorro, relativas ao e-mail e ao número de telefone salvos. Clicar no ícone de engrenagem a partir desta tela irá retornar ao Menu Principal.

Figura 3.12: Menu de Configurações do programa SOS Connect. Fonte: Autoria própria

Capítulo 4

Resultados

Este capítulo é destinado à apresentação dos resultados obtidos. Serão mostrados o funcionamento do sistema, a avaliação do programa através de testes de validação e a análise de desempenho do projeto.

4.1 Treinamento

Conforme mencionado na Seção 3.3.3, são utilizadas 15 *epochs* para o treinamento do algoritmo. Foram realizados uma série de treinamentos, realizando pequenas modificações nas proporções de *datasets*, quantidade de camadas, taxa de aprendizagem, entre outros, até se obter um resultado satisfatório. No algoritmo final, foram utilizadas 3 camadas e taxa de aprendizagem de 0.1, e reduziu-se a taxa de amostragem das amostras de treinamento para 16 kHz. Pode-se observar o desempenho do treinamento através da análise das acurácias ao longo das *epochs*, mostrado na Figura 4.1.

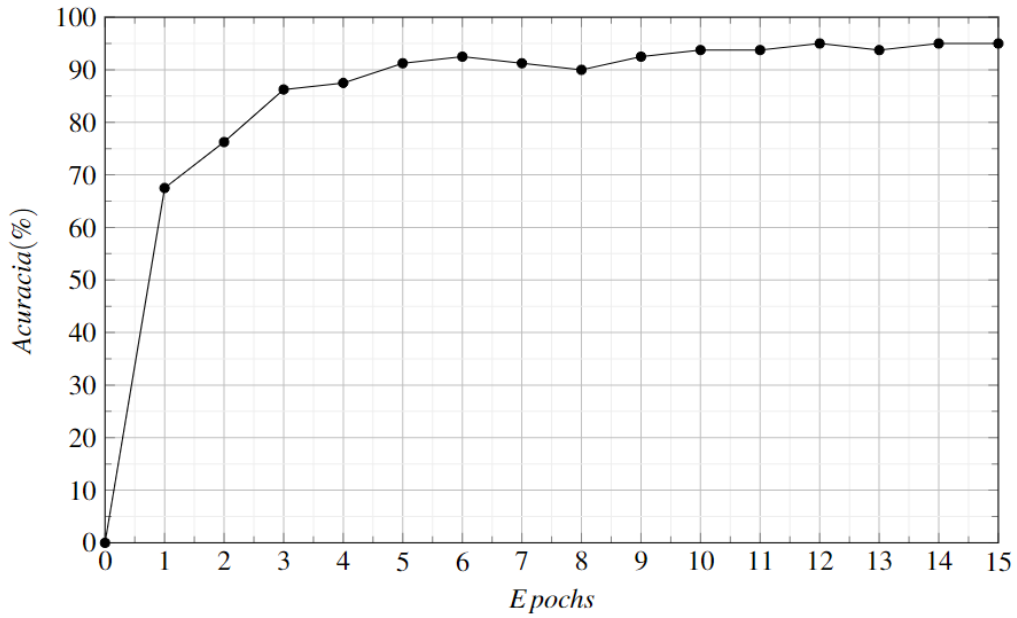


Figura 4.1: Gráfico de acurácia do sistema em função do número de *epochs*. Fonte: Autoria própria

4.2 Validações

Para realizar a validação do sistema de detecção, foram realizados 20 testes em cada palavra, de forma a quantizar os acertos e medir o desempenho do programa. O limiar de detecção foi estabelecido como 80%, de forma que a palavra é considerada identificada se a sua porcentagem for maior que este valor. Devido ao arredondamento da última casa decimal, pode-se considerar uma margem de erro de $0,1\%$.

Como forma de comparar a atuação do algoritmo em diferentes tipos de vozes, foram feitos testes com uma das vozes utilizadas para treinar o algoritmo, como base, e com uma voz que não foi usada para treinar o algoritmo. Dessa forma, foram realizados 4 *sets* de testes, para cada palavra combinada a cada voz, conforme apresentado nas Tabelas 4.1, 4.2, 4.3 e 4.4.

Tabela 4.1: Testes da palavra Socorro com voz utilizada para treinar o algoritmo. Fonte: Autoria própria

Nº	Palavra	Socorro (%)	Ajuda (%)	Nenhuma (%)	Conclusão	Resultado
1	Socorro	98.0	1.6	0.4	Socorro	Acerto
2	Socorro	97.6	1.8	0.6	Socorro	Acerto
3	Socorro	98.0	1.5	0.5	Socorro	Acerto
4	Socorro	98.1	1.6	0.3	Socorro	Acerto
5	Socorro	97.9	1.5	0.6	Socorro	Acerto
6	Socorro	97.9	1.7	0.4	Socorro	Acerto
7	Socorro	98.0	1.6	0.4	Socorro	Acerto
8	Socorro	96.6	2.1	1.3	Socorro	Acerto
9	Socorro	98.1	1.5	0.4	Socorro	Acerto
10	Socorro	97.8	1.6	0.6	Socorro	Acerto
11	Socorro	97,7	1,6	0,7	Socorro	Acerto
12	Socorro	98.0	1.5	0.5	Socorro	Acerto
13	Socorro	97.9	1.6	0.5	Socorro	Acerto
14	Socorro	98.0	1.6	0.4	Socorro	Acerto
15	Socorro	98.0	1.5	0.5	Socorro	Acerto
16	Socorro	98.1	1.5	0.4	Socorro	Acerto
17	Socorro	98.0	1.6	0.4	Socorro	Acerto
18	Socorro	97.6	1.8	0.6	Socorro	Acerto
19	Socorro	98.0	1.5	0.5	Socorro	Acerto
20	Socorro	98.1	1.6	0.3	Socorro	Acerto

Tabela 4.2: Testes da palavra Socorro com voz não utilizada para treinar o algoritmo. Fonte: Autoria própria

Nº	Palavra	Socorro (%)	Ajuda (%)	Nenhuma (%)	Conclusão	Resultado
1	Socorro	97.5	1.5	0.5	Socorro	Acerto
2	Socorro	97.9	1.5	0.6	Socorro	Acerto
3	Socorro	97.1	2.1	0.8	Socorro	Acerto
4	Socorro	96.8	2.4	0.8	Socorro	Acerto
5	Socorro	96.8	1.9	1.3	Socorro	Acerto
6	Socorro	96.7	2.4	0.9	Socorro	Acerto
7	Socorro	97.9	1.6	0.5	Socorro	Acerto
8	Socorro	97.7	1.6	0.7	Socorro	Acerto
9	Socorro	97.2	2.1	0.7	Socorro	Acerto
10	Socorro	97.8	1.6	0.6	Socorro	Acerto
11	Socorro	96.9	2.1	1.0	Socorro	Acerto
12	Socorro	97.8	1.7	0.5	Socorro	Acerto
13	Socorro	97.8	1.6	0.6	Socorro	Acerto
14	Socorro	97.9	1.6	0.5	Socorro	Acerto
15	Socorro	96.8	2.7	0.5	Socorro	Acerto
16	Socorro	96.5	2.4	1.1	Socorro	Acerto
17	Socorro	95.3	3.0	1.7	Socorro	Acerto
18	Socorro	97.9	1.5	0.6	Socorro	Acerto
19	Socorro	97.7	1.6	0.7	Socorro	Acerto
20	Socorro	97.7	1.7	0.6	Socorro	Acerto

Tabela 4.3: Testes da palavra Ajuda com voz utilizada para treinar o algoritmo. Fonte: Autoria própria

Nº	Palavra	Socorro (%)	Ajuda (%)	Nenhuma (%)	Conclusão	Resultado
1	Ajuda	3.1	96.5	0.4	Ajuda	Acerto
2	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
3	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
4	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
5	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
6	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
7	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
8	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
9	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
10	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
11	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
12	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
13	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
14	Ajuda	3.1	96.5	0.4	Ajuda	Acerto
15	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
16	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
17	Ajuda	3.0	96.6	0.4	Ajuda	Acerto
18	Ajuda	3.1	96.5	0.4	Ajuda	Acerto
19	Ajuda	3.0	96.5	0.5	Ajuda	Acerto
20	Ajuda	3.0	96.5	0.5	Ajuda	Acerto

Tabela 4.4: Testes da palavra Ajuda com voz não utilizada para treinar o algoritmo. Fonte: Autoria própria

Nº	Palavra	Socorro (%)	Ajuda (%)	Nenhuma (%)	Conclusão	Resultado
1	Ajuda	3.1	96.5	0.4	Ajuda	Acerto
2	Ajuda	97.6	1.7	0.7	Socorro	Erro
3	Ajuda	3.1	96.5	0.4	Ajuda	Acerto
4	Ajuda	5.9	92.8	1.3	Ajuda	Acerto
5	Ajuda	94.1	2.4	3.5	Socorro	Erro
6	Ajuda	17.2	65.4	17.4	Nenhuma	Erro
7	Ajuda	3.3	96.3	0.4	Ajuda	Acerto
8	Ajuda	3.1	96.6	0.3	Ajuda	Acerto
9	Ajuda	97.3	2.0	0.7	Socorro	Erro
10	Ajuda	3.4	96.2	0.4	Ajuda	Acerto
11	Ajuda	5.1	1.9	93.0	Nenhuma	Erro
12	Ajuda	3.1	96.5	0.4	Ajuda	Acerto
13	Ajuda	84.9	3.5	11.6	Socorro	Erro
14	Ajuda	78.9	3.8	17.3	Nenhuma	Erro
15	Ajuda	97.6	1.7	0.7	Socorro	Erro
16	Ajuda	3.2	96.4	0.4	Ajuda	Acerto
17	Ajuda	6.3	3.1	90.6	Nenhuma	Erro
18	Ajuda	3.2	96.5	0.3	Ajuda	Acerto
19	Ajuda	3.2	96.4	0.4	Ajuda	Acerto
20	Ajuda	30.5	62.9	6.6	Nenhuma	Erro

4.2.1 Discussão dos Resultados

A partir dos testes, pode-se observar uma taxa de acerto de 100% no reconhecimento das palavras “Socorro” e “Ajuda” quando testadas com a voz utilizada para o treinamento do algoritmo. Entretanto, quando testado com uma voz não utilizada no treinamento, o algoritmo obteve pior desempenho, reconhecendo a palavra “So-

corro" 100% das vezes, e a palavra "Ajuda" em 50% dos testes.

Embora se comporte como desejado quando testado com um timbre conhecido, este resultado mostra que o *dataset* de treinamento não possui diversidade suficiente de vozes para que o sistema seja capaz de reconhecer as palavras em questão com alta precisão.

4.3 Funcionamento da Interface Gráfica

Após a criação da Interface gráfica através do QtDesigner, exportação do arquivo em formato python (.py), incorporação no código do cliente e programação das interações com os elementos gráficos, pode-se, por fim, testar o funcionamento da Interface Gráfica.

Realiza-se a inicialização da interface gráfica abrindo o SOS Client através do Prompt de Comando, como pode ser observado na Figura 4.2.

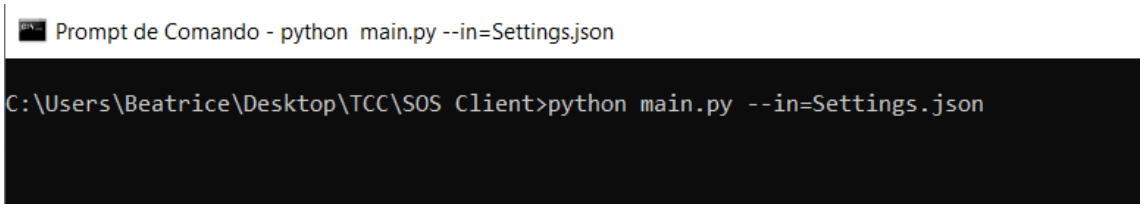


Figura 4.2: Inicialização do SOS Client pelo Prompt de Comando. Fonte: Autoria própria

Em resposta, o sistema abre a Tela de Carregamento (Figura 3.10), seguido do Menu Principal (Figura 3.11). Conforme mencionado previamente, conecta-se com o servidor clicando-se no botão "CONECTAR", na seção "SOS Server", o que habilita os Modos Manual e Automático, conforme pode ser observado na Figura 4.3.

Figura 4.3: Menu Principal do programa SOS Connect conectado ao servidor. Fonte: Autoria própria

Assim, é possível iniciar a detecção nos modos Automático e Manual, pressionando "INICIAR" em suas respectivas seções. A realização dos testes mostrou os possíveis resultados da classificação - 'Socorro Identificado', 'Ajuda Identificado' ou 'Não Identificado' -, conforme mostrado na Figura 4.4.

(a) **(b)** **(c)**

Figura 4.4: Possíveis resultados da detecção. Fonte: Autoria própria (a) 'Socorro' identificado. (b) 'Ajuda' identificado. (c) Não identificado.

Pode-se perceber, após a realização dos testes, que a interface gráfica se comporta conforme planejado, de acordo com o que foi estabelecido na Metodologia, indicando um resultado satisfatório.

Também é possível observar que os valores de probabilidade exibidos na Interface Gráfica correspondem aos valores simultâneos do arquivo JSON, indicando que a

comunicação também está se comportando satisfatoriamente.

Capítulo 5

Considerações Finais

O SOS Connect foi desenvolvido com o intuito de fazer o monitoramento remoto de pessoas em situação de vulnerabilidade e realizar a detecção automática de situações de risco. Seu objetivo é garantir que os pedidos de socorros sejam detectados imediatamente, informando aos responsáveis e agilizando o processo de atendimento.

5.1 Conclusão

Neste trabalho, pôde-se implementar satisfatoriamente um algoritmo de Aprendizagem de Máquina para detecção das palavras-chave “Socorro” e “Ajuda”; além de executá-lo remotamente na Raspberry Pi.

Para a comunicação remota, pôde-se estabelecer, com êxito, uma comunicação TCP-IP entre o computador e a Raspberry, através do envio de pacotes no formato JSON.

De forma a tornar a utilização do sistema intuitiva ao usuário, realizou-se a implementação de uma interface gráfica, permitindo que o sistema possa ser utilizada por todos, independente do grau de conhecimento computacional.

A realização de testes para a validação do SOS Connect permitiu observar que o sistema de detecção sonora provou-se eficaz 100% das vezes quando testado com vozes utilizadas no treinamento; 100% das vezes quando testado com a palavra

"Socorro" com vozes não utilizadas previamente e 50% das vezes quando testado com a palavra "Ajuda" com vozes não utilizadas previamente.

Com base nos resultados obtidos com as vozes utilizadas no treinamento, pode-se induzir que esta queda de eficiência pode ser contornada ao se realizar um novo treinamento, utilizando amostras de voz do futuro usuário do sistema.

5.2 Dificuldades Encontradas

Ao longo do desenvolvimento deste trabalho, foram encontradas dificuldades em sua execução, obstáculos que foram eventualmente contornados. Apresenta-se, a seguir, as principais dificuldades encontradas.

- Houve dificuldade na definição do microfone para o projeto. Escolheu-se, inicialmente, o módulo de gravação e reprodução de áudio ISD1820, mas posteriormente se constatou que este módulo só é capaz de gravar 10 segundos por vez. Por fim, devido à alta qualidade de captação e compatibilidade com a Raspberry Pi, escolheu-se o *GXT 232 Mantis* como microfone utilizado no projeto;

- Ausência de *datasets* sonoros com as palavras-chave escolhidas nas bases de dados consultados. Como consequência, foi necessário criar o *dataset* manualmente, e obteve-se um *dataset* limitado quanto a quantidade de timbres;

- Durante um dos testes com a Raspberry Pi, esta passou a não responder à comunicação, independentemente das alterações realizadas. Assim, foi necessário reinstalar o Linux na Raspberry Pi para conseguir executar o programa e estabelecer a comunicação satisfatoriamente.

5.3 Trabalhos Futuros

O aprimoramento das funcionalidades deste trabalho e adição de melhorias podem dar origem a trabalhos futuros. Apresenta-se, a seguir, possíveis avanços a

serem realizados a partir do projeto atual.

- Expandir o *dataset* de forma a incluir uma maior variedade de vozes e, consequentemente, uma maior diversidade de timbres, pode tornar o algoritmo mais eficiente na detecção de diferentes tipos de vozes;

- Incluir a detecção de novos identificadores de situações de risco, tais como barulhos de queda, gritos, tiros, entre outros, garantindo mais formas de identificar situações de perigo;

- Reproduzi-lo em grande escala, usando-o para conectar a população a órgãos governamentais, emitindo alertas diretamente a unidades de atendimento médicas ou policiais;

- Implementar um sistema de armazenamento de dados pessoais, como um cadastro com informações pessoais e médicas, para permitir a rápida identificação do usuário e agilizar a burocracia relacionada ao internamento, relativa ao preenchimento de todos os dados e apresentação de documentos. O sistema de armazenamento de dados também pode incluir as informações geradas pelo aplicativo, gerando um histórico que pode ser consultado posteriormente.

Referências

Alpaydin, E. (2014). *Introduction to Machine Learning*. The MIT Press, Cambridge, 3ª edição.

Apple (2022). Usar o recurso sos de emergência no iphone. https://support.apple.com/pt-br/HT208076. Acesso em: 19/12/2022.

Badillo, S., Banfai, B., Birzele, F., Davydov, I. I., Hutchinson, L., Kam-Thong, T., Siebourg-Polster, J., Steiert, B., e Zhang, J. D. (2020). An introduction to machine learning. *Clinical Pharmacology & Therapeutics*, 107(4):871–885.

Buksman, S., Vilela, A., Pereira, S., Lino, V., e Santos, V. (2008). *Quedas em Idosos: Prevenção*. Sociedade Brasileira de Geriatria e Gerontologia.

Caviglione, L., Merlo, A., e Migliardi, M. (2012). Green-aware security: Towards a new research field. *Journal of Information Assurance and Security*, 7:338–346.

Chen, Z., Chen, L., e Xu, B. (2014). Hybrid information flow analysis for python bytecode. In *2014 11th Web Information System and Application Conference*, páginas 95–100.

Chollet, F. (2018). *Deep Learning with Python*. Manning Publications Co, Shelter Island, 1ª edição.

Da Costa, C. A., Yamin, A. C., e Geyer, C. F. R. (2008). *Toward a General Software Infrastructure for Ubiquitous Computing*. IEEE Pervasive Computing. IEEE CS.

da Costa, E. C. (2003). *Acústica Técnica*. Blucher, São Paulo, 1ª edição.

do Valle, S. (2015). *Microfones*. Editora Música & Tecnologia, 2ª edição.

Everest, A. F. (2001). *The Master Handbook of Acoustics*. McGraw-Hill, 4ª edição.

FFmpeg (2008). Manual ffmpeg. https://ffmpeg.org/ffmpeg-all.html. Acesso em: 09/12/2022.

Forouzan, B. A. (2013). *Comunicação de dados e redes de computadores*. McGraw Hill Brasil, 4ª edição.

Frešer, M., Košir, I., Mirchevska, V., e Luštrek, M. (2019). An elderly-care system based on sound analysis. *Signals and Telecommunication Journal*, 8:54–59.

Geerling, J. (2023). Power consumption benchmarks. https://www.pidramble.com/wiki/benchmarks/power-consumption. Acesso em: 01/04/2023.

Google (2022). Receber ajuda durante uma emergência com o smartphone android. https://support.google.com/android/answer/9319337?hl=pt-BR. Acesso em: 19/12/2022.

Guo, Y. (2017). The 7 steps of machine learning. https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e. Acesso em: 25/09/2022.

Halfacree, G. (2019). Benchmarking the raspberry pi 4. https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b. Acesso em: 15/01/2023.

Halfacree, G. (2020). *The Official Raspberry Pi Beginner's Guide*. Raspberry Pi PRESS, Cambridge, 4ª edição.

Halliday, D. e Resnick, R. (2014). *Fundamentos de Física, Volume 2: Gravitação, Ondas e Termodinâmica*. LTC, 9ª edição.

Ivo, A. (2004). Fonética acústica. https://grad.letras.ufmg.br/arquivos/monitoria/Aula%2004%20apoio.pdf. Acesso em: 08/02/2023.

Khan, Z. A., Yar, H., Salahuddin, e Nasir, M. (2018). Raspberry pi based elderly fall detection system. In *4th International Conference on Next Generation Computing*.

Kurose, J. F. e Ross, K. W. (2013). *Redes de computadores e a Internet: uma abordagem top-down*. Pearson Education do Brasil, São Paulo, 6ª edição.

Lathi, B. P. (2008). *Sinais e Sistemas Lineares*. Bookman, 2ª edição.

Lathi, B. P. e Green, R. A. (2018). *Linear Systems and Signals*. Oxford University Press, New York, 3ª edição.

Madhubala, J. S. e Umamakeswari, A. (2015). A vision based fall detection system for elderly people. *Indian Journal of Science and Technology*, 8:167.

Muheidat, F., Tawalbeh, L., e Tyrer, H. (2018). Context-aware, accurate, and real time fall detection system for elderly people. In *2018 12th IEEE International Conference on Semantic Computing*, IEE.

Myers, B. A. (2004). *Computer Science Handbook*. CRC Press, 1^a edição.

Müller, A. C. e Guido, S. (2016). *Introduction to Machine Learning with Python*. O'Reilly Media, Sebastopol, 1^a edição.

Naeem, A., Safdar, W., e Qadar, A. (2014). Voice controlled intelligent wheelchair using raspberry pi. *International Journal of Technology and Research*.

Negri, F. e Ledel, L. C. (2016). Plataforma de hardware e software com interface de reconhecimento de voz. Dissertação de mestrado, IFSP, São Paulo.

Nussenzveig, H. M. (2014). *Curso de Física Básica, Volume 2: fluidos, oscilações e ondas, calor*. Blucher, São Paulo, 5^a edição.

Organization, W. H. (2007). *WHO Global Report on Falls Prevention in Older Age*. World Health Organization, Geneva.

Pasricha, S. e Dutt, N. (2008). *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers, 1^a edição.

Peterson, L. L. e Davie, B. S. (2013). *Redes de computadores: Uma abordagem de sistemas*. Elsevier Editora Ltda., Rio de Janeiro.

Power, R. e Rubinsteyn, A. (2013). How fast can we make interpreted python? *arXiv*.

Queiroz, S. S. F. (2016). Sos móvel: Sistema para auxiliar pessoas na solicitação de socorro. Dissertação de mestrado, UERN, Mossoró.

Raghav Bali, Dipanjan Sarkar, B. L. e Lesmeister, C. (2016). *R: Unleash Machine Learning Techniques*. Packt Publishing, Birmingham, 1^a edição.

Raguraman, P., R., M., e Vijayan, M. (2019). Librosa based assessment tool for music information retrieval systems. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, páginas 109–114.

Raspberry (2019). *Raspberry Pi 4 Model B Datasheet*. Raspberry.

Rossing, T. D., Moore, R. F., e Wheeler, P. A. (2014). *The Science of Sound*. Pearson, Harlow, 3^a edição.

Sehili, M. A., Lecouteux, B., Vacher, M., Portet, F., Istrate, D., Dorizzi, B., e Boudy, J. (2012). Sound environment analysis in smart home. In *Ambient Intelligence - Third International Joint Conference*, AmI, páginas 208–223.

Shalev-Shwartz, S. e Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, 1^a edição.

Tanenbaum, A. S. (2011). *Redes de computadores*. Artmed Editora, Porto Alegre, 5^a edição.

Tinetti, M. E., Liu, W.-L., e Claus, E. B. (1993). Predictors and prognosis of inability to get up after falls among elderly persons. *JAMA*, 269(1).

Todd, C. e Skelton, D. (2004). What are the main risk factors for falls amongst older people and what are the most effective interventions to prevent these falls? *World Health Organization*.

Trust (2023). Microfone usb para transmissão em fluxo gxt 232 mantis. https://www.trust.com/pt/product/22656-gxt-232-mantis-usb-streaming-microphone. Acesso em: 05/02/2023.

Valentim, A., Côrtes, M., e Gama, A. C. (2010). Análise espectrográfica da voz: efeito do treinamento visual na confiabilidade da avaliação. *Revista Sociedade Brasileira de Fonoaudiologia*.

Waheed, S. A. e Khader, D. P. S. A. (2017). A novel approach for smart and cost effective iot based elderly fall detection system using pi camera. In *2017 IEEE International Conference on Computational Intelligence and Computing Research*, IEE.

Wetherall, A. S. T. D. J. (2011). *Computer Networks*. Pearson Education Limited, Harlow.

Yacchiremaa, D., de Pugaa, J. S., Palaua, C., e Esteve, M. (2018). Fall detection system for elderly people using iot and big data. *Elsevier*.

Zhou, Z.-H. (2016). *Machine Learning*. Springer, Singapura, 1^a edição.

Zuben, P. (2004). *Música e tecnologia: o som e seus novos instrumentos*. Irmãos Vitale, 1^a edição.

Apêndice A

Apêndice

A.1 Código 'Main' do SOS Server. Fonte: Autoria própria

```
#################################################################
#         INTERFACE GRAFICA   -   SOSConnect (SOSServer)        #
#      Por Beatrice A. Michalewicz       Versao: 0.1.0          #
#################################################################

from __future__ import division
from __future__ import print_function
#from collections import namedtuple
import os, sys, time, datetime, logging, threading, socket, select
#import keyboard
import json
import ClassSettings as cs
import utils.lib_rnn as lib_rnn
import utils.lib_datasets as lib_datasets
from utils.lib_gui import GuiForAudioClassification
from utils.lib_record_audio import *


# -- Main class for reading audio from microphone, doing inference,
```

```
    and display result
class AudioClassifierWithGUI(object):

    def __init__(self, src_weight_path, src_classes_path,
    dst_audio_folder):

        # Init model
        model, classes = lib_rnn.setup_default_RNN_model(
    src_weight_path, src_classes_path)
        self._CLASSES = classes
        print("Number of classes = {}, classes: {}".format(len(
    classes), classes))
        model.set_classes(classes)
        self._model = model

        # Set up GUI
        self._gui = GuiForAudioClassification(classes, hotkey="R")

        # Set up audio recorder
        self._DST_AUDIO_FOLDER = dst_audio_folder
        self._recorder = AudioRecorder()

    def record_audio_and_classifiy(self, is_shout_out_result=False)
    :
        model, classes = self._model, self._CLASSES
        gui, recorder = self._gui, self._recorder

        # -- Record audio
        #gui.enable_img1_self_updating()
        print(self._DST_AUDIO_FOLDER)
        recorder.start_record(self._DST_AUDIO_FOLDER)  # Start
    record
        # while not gui.is_key_released():  # Wait for key released
        #     time.sleep(0.001)
        time.sleep(2)
```

```
49          recorder.stop_record()  # Stop record
50
51          # -- Do inference
52          audio = lib_datasets.AudioClass(filename=recorder.filename)
53          probs = model.predict_audio_label_probabilities(audio)
54          predicted_idx = np.argmax(probs)
55          predicted_label = classes[predicted_idx]
56          max_prob = probs[predicted_idx]
57          print("\nAll word labels: {}".format(classes))
58          print("\nPredicted label: {}, probability: {}\n".format(
    predicted_label, max_prob))
59          PROB_THRESHOLD = 0.8
60          final_label  = predicted_label if max_prob > PROB_THRESHOLD
     else "none"
61
62          #Update Json Parameters
63          if final_label == "socorro":
64              Settings.palavra_identificado = True
65              Settings.socorro_identificado = True
66              Settings.ajuda_identificado = False
67          elif final_label =="ajuda":
68              Settings.palavra_identificado = True
69              Settings.socorro_identificado = False
70              Settings.ajuda_identificado = True
71          else:
72              Settings.palavra_identificado = False
73              Settings.socorro_identificado = False
74              Settings.ajuda_identificado = False
75
76          Settings.prob_socorro = np.round(100*probs[0], 2)
77          Settings.prob_ajuda = np.round(100*probs[1], 2)
78          Settings.prob_nenhuma = np.round(100*probs[2], 2)
79          print(audio.get_len_s())
80          Settings.counter_time_voice_auto = audio.get_len_s()
81          #Settings.voice_intensity_auto = 54
```

```
82
83          # Send data
84          filetosend = Settings.EncoderFile()
85          print(filetosend)
86          servidor.client.send(filetosend.encode())
87
88          # -- Update the image
89
90          # Update image1: first stop self updating,
91          # then set recording_length and voice_intensity to zero
92          #gui.reset_img1()
93
94          # Update image 2: the prediction results
95          #gui.set_img2(
96          #    final_label=final_label,
97          #    predicted_label=predicted_label,
98          #    probability=max_prob,
99          #    length=audio.get_len_s(),
100         #    valid_length=audio.get_len_s(), # TODO: remove the
    silent voice,
101         #)
102
103         # Update image 3: the probability of each class
104         #gui.set_img3(probabilities=probs)
105
106         # -- Shout out the results. e.g.: two is one
107         if is_shout_out_result:
108             lib_datasets.shout_out_result(
109                 recorder.filename, # Raw audio to shout out
110                 final_label,
111                 middle_word="is",
112                 cache_folder="data/examples/")
113
114         return predicted_label, max_prob
115
```

```
116     def is_key_quit_pressed(self):
117         return self._gui.is_key_quit_pressed()
118 
119     def is_key_pressed(self):
120         return self._gui.is_key_pressed()
121 
122     def is_key_released(self):
123         return self._gui.is_key_released()
124 
125 
126 def classifier():
127 
128     while True:
129         if (Settings.gravando_auto):
130             # Audio recorder and classifier
131             audio_clf = AudioClassifierWithGUI(SRC_WEIGHT_PATH,
    SRC_CLASSES_PATH, DST_AUDIO_FOLDER)
132             print("Gravando...")
133             shout_out_result=False
134             audio_clf.record_audio_and_classifiy(shout_out_result)
135         time.sleep(0.1)
136 
137 
138 
139 class SockServer(threading.Thread):
140     """ Classe para prover servidor de conexao socket usando
    threads
141         Sintaxe: servidor = SockServer(host,port)
142     """
143 
144     def __init__(self, host, port):
145         # Metodo para construir a classe
146         threading.Thread.__init__(self)
147         self.running = True
148         self.host = host
```

```
149         self.port = port
150 
151         # Configurando socket para TCP
152         self.sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
153         self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR
    , 1)
154         self.sock.bind((self.host, self.port))
155 
156         self.server_address = (self.host, self.port)
157         print('starting up on {} port {}'.format(*self.
    server_address))
158 
159     def stop(self):
160         # Metodo para interromper a thread do servidor.
161         self.running = False
162 
163     def _select(self, socket):
164         # Metodo para detectar se ocorreu recepcao de dados no
    socket
165         # sem interromper o loop no comando 'recv'.
166         # Retorna 'True' se existirem dados no buffer.
167         reads, writes, errors = select.select([socket], [], [],
    0.0001)
168         return socket in reads
169 
170     def run(self):
171         # Metodo obrigatorio para a thread aceitar ativacao atraves
     de 'start()'.
172 
173         # Ativa a recepcao de conexao dos clientes.
174         self.sock.listen(5)
175         print('Waiting for a connection')
176         self.client, self.address = self.sock.accept()
177         print('connection from', self.address)
```

```
178         self.running = True
179         self.exec_cmd()
180         # # Rotina que trava a execucao da recepcao de dados ate o
    momento da
181         # # deteccao da conexao de um cliente.
182         # while self.running and not self._select(self.sock):
183         #     time.sleep(0.1)
184         # # Estabelece a conexao com o cliente e executa a rotina
    para processamento
185         # # das informacoes recebidas do cliente.
186         # if self._select(self.sock):
187 
188 
189     def exec_cmd(self):
190         # Metodo que  executa o comando recebido do cliente
191         size = 2048
192         while self.running:
193             #try:
194             # Rotina para tratar a chegada de dados no socket
195             # if self._select(client):
196             time.sleep(0.1)
197             print("Waiting to receive")
198             try:
199                 cmd = self.client.recv(size).decode()
200             except:
201                 print("excided time")
202             if cmd:
203                 print('received {!r}'.format(cmd))
204 
205                 jsonSettings = json.loads(cmd)
206                 Settings.DecoderFile(jsonSettings)
207                 # Settings = cs.ClassSettings(jsonSettings)
208 
209                 if Settings.gravar_auto == True:
210                     print(Settings.gravar_auto)
```

```
211                     Settings.gravando_auto = True
212                     # Send data
213                     filetosend = Settings.EncoderFile()
214                     self.client.send(filetosend.encode())
215                     print("Package with gravando_auto is sended")
216 
217                 if Settings.gravar_auto == False:
218                     print(Settings.gravar_auto)
219                     Settings.gravando_auto = False
220                     # Send data
221                     filetosend = Settings.EncoderFile()
222                     self.client.send(filetosend.encode())
223 
224 
225 
226 def timestamp(tipo):
227     """ Funcao para gerar timestamp de uso geral """
228 
229     if tipo == 'hmsms':
230         return str('{0:%H%M%S%f}'.format(datetime.datetime.now()))
    [0:9]
231     elif tipo == 'ms':
232         return str('{:%f}'.format(datetime.datetime.now()))[0:3]
233 
234 def closeall():
235     servidor.stop()
236     closeStructures=True
237 
238 # Chamada para camada de comunicacao socket
239 if __name__ == "__main__":
240 
241     ROOT = os.path.dirname(os.path.abspath(__file__)) # root of the
    project
242     sys.path.append(ROOT)
243 
```

```
244
245     # -- Settings
246     SRC_WEIGHT_PATH = os.path.join(ROOT, "weights_SOSConnect.ckpt")
247     SRC_CLASSES_PATH = os.path.join(ROOT, "labels_SOSConnect.txt")
248     DST_AUDIO_FOLDER = os.path.join(ROOT, "data/data_tmp/")
249
250     # os.system('clear')
251
252     # Criando variaveis de ambiente e verificando diretorios.
253     # Detectando em qual diretorio a aplicacao esta sendo executada
254     dir_atual = os.getcwd()
255
256     # Verificando se existe o diretorio de 'log'.
257     # Se nao existir - Criar.
258     log_dir = dir_atual + '/log'
259     if not os.path.exists(log_dir):
260         os.mkdir(log_dir)
261
262     # Criando o arquivo 'simul.log'.
263     nome_arq_log = log_dir + '/' + 'sock_' + timestamp('hmsms') + '
    .log'
264     with open(nome_arq_log, 'w') as arq_log:
265         arq_log.write('\n')
266
267     # Variavel com o nome da aplicacao
268     #nome_prog = sys.argv[0].split('./')[1]
269     nome_prog ='SOS connect'
270
271     # Ajustando recursos de logging
272     logger = logging.getLogger(nome_prog)
273     handler = logging.FileHandler(nome_arq_log)
274     formatter = logging.Formatter('%(asctime)s %(levelname)s %(
    message)s')
275     handler.setFormatter(formatter)
276     logger.addHandler(handler)
```

```
277
278     # Setando para debug
279     logger.setLevel(logging.DEBUG)
280
281     # CONFIGURAcaO DO SERVIDOR
282     # Endereco IP da maquina LOCAL
283     ip_local = ''
284     port_local = 10000
285
286     closeStructures = False
287
288     Settings = cs.ClassSettings()
289
290     servidor = SockServer(ip_local, port_local)
291     servidor.start()
292
293     classifier()
294
295     sys.exit(closeall())
```

A.2 Código 'Main' do SOS Client. Fonte: Autoria própria

```
1  ######################################################################
2  #         INTERFACE GRAFICA   -   SOSConnect (SOSClient)       #
3  #       Por Beatrice A. Michalewicz       Versao: 0.1.0        #
4  ######################################################################
5
6  """ Qt + Server SOS
7  Usage:
8      main.py (-h | --help)
9      main.py --in=
10     main.py
11 Options:
```

```
12      -h --help                               Show this screen
13  """
14  from ast import Try
15  import os, sys, time, datetime, logging, threading, socket, select
16  import platform
17  from PySide2 import QtCore, QtGui, QtWidgets
18  from PySide2.QtCore import (QCoreApplication, QPropertyAnimation,
        QDate, QDateTime, QMetaObject, QObject, QPoint, QRect, QSize,
        QTime, QUrl, Qt, QEvent)
19  from PySide2.QtGui import (QBrush, QColor, QConicalGradient,
        QCursor, QFont, QFontDatabase, QIcon, QKeySequence,
        QLinearGradient, QPalette, QPainter, QPixmap, QRadialGradient)
20  from PySide2.QtWidgets import *
21
22  import struct
23  import json
24  from json import JSONDecodeError, JSONEncoder
25  from docopt import docopt
26  import ClassSettings as cs
27  from threading import Timer
28  import math
29
30
31  ########################################################################
32  #               IMPORTANDO JANELAS DO QTDESIGNER                       #
33  ########################################################################
34
35  ## == SPLASH SCREEN
36  from uisplashscreen import UiSplashScreen
37
38  ## == MAIN WINDOW
39  from uimain import UiMainWindow
40
41  ## == CONFIGURACOES
42  from uiconfiguracoes import UiConfiguracoes
```

```
43
44 import FilesResource
45
46 ######################################################################
47 #                            TELA PRINCIPAL                          #
48 ######################################################################
49
50     # MODOS DE OPERACAO
51
52     ## 0 - Desconectado
53     ## 1 - Conectado
54     ## 2 - Conectado e em modo automatico
55     ## 3 - Conectado e em modo manual
56
57 # YOUR APPLICATION
58 class MainWindow(QMainWindow):
59     def __init__(self):
60         QMainWindow.__init__(self)
61         self.ui = Ui_MainWindow()
62         self.ui.setupUi(self)
63
64         # REMOVE TITLE BAR
65         self.setWindowFlag(QtCore.Qt.FramelessWindowHint)
66         self.setAttribute(QtCore.Qt.WA_TranslucentBackground)
67
68         self.ServerDesconectado()
69
70         #Se eu pressionar o botAo para conectar, ir para "
    ServerConectando"
71         self.ui.Button_Server.clicked.connect(self.ServerConectando
    )
72
73         #Se eu pressionar o botAo para gravar Auto, ir para "
    GravandoAuto"
74         self.ui.Button_Auto.clicked.connect(self.GravarAuto)
```

```
75          self.ui.Button_Manual.clicked.connect(self.GravarManual)
76
77          #Se pressionar botAo "x", fechar a pAgina
78          self.ui.Exit_Button.clicked.connect(self.close)
79
80          #Se pressionar botAo de ConfiguraCOes, ir para
    ConfiguraCOes
81          self.ui.Config_Button.clicked.connect(self.Configurar)
82
83          self.ui.CheckBox_EMail.stateChanged.connect(self.
    EnviarEmail)
84          self.ui.CheckBox_Telegram.stateChanged.connect(self.
    EnviarTelegram)
85
86          self.updatingScreen = False
87          self.timer = QtCore.QTimer()
88          self.timer.setInterval(100)  # 100 milliseconds = 0.1
    seconds
89          self.timer.timeout.connect(self.UpdateScreen)  # Connect
    timeout signal to function
90          self.timer.start()  # Set the timer running
91
92      def UpdateScreen(self):
93          self.repaint()
94
95      def mousePressEvent(self, event):
96          self.oldPosition = event.globalPos()
97
98      def mouseMoveEvent(self, event):
99          delta = QPoint(event.globalPos() - self.oldPosition)
100         self.move(self.x() + delta.x(), self.y() + delta.y())
101         self.oldPosition = event.globalPos()
102
103     def UpdateProbSocorro(self, socorroValue):
104         self.formatGrayStyleSheet = "QPushButton {\n" "background-
```

```
    color: rgb(165, 165, 175);\n"  "border-radius: 5px;\n" "}"
105         self.formatPurpleStyleSheet = "QPushButton {\n" "background
    -color: rgb(254, 66, 139);\n"  "border-radius: 5px;\n" "}"
106         self.ui.Label_ProbSocorro.setText(str(socorroValue))
107         if socorroValue>=10:
108             self.ui.ProbSocorro01.setStyleSheet(self.
    formatPurpleStyleSheet)
109         else:
110             self.ui.ProbSocorro01.setStyleSheet(self.
    formatGrayStyleSheet)
111         if socorroValue>=20:
112             self.ui.ProbSocorro02.setStyleSheet(self.
    formatPurpleStyleSheet)
113         else:
114             self.ui.ProbSocorro02.setStyleSheet(self.
    formatGrayStyleSheet)
115         if socorroValue>=30:
116             self.ui.ProbSocorro03.setStyleSheet(self.
    formatPurpleStyleSheet)
117         else:
118             self.ui.ProbSocorro03.setStyleSheet(self.
    formatGrayStyleSheet)
119         if socorroValue>=40:
120             self.ui.ProbSocorro04.setStyleSheet(self.
    formatPurpleStyleSheet)
121         else:
122             self.ui.ProbSocorro04.setStyleSheet(self.
    formatGrayStyleSheet)
123         if socorroValue>=50:
124             self.ui.ProbSocorro05.setStyleSheet(self.
    formatPurpleStyleSheet)
125         else:
126             self.ui.ProbSocorro05.setStyleSheet(self.
    formatGrayStyleSheet)
127         if socorroValue>=60:
```

```
128             self.ui.ProbSocorro06.setStyleSheet(self.
    formatPurpleStyleSheet)
129         else:
130             self.ui.ProbSocorro06.setStyleSheet(self.
    formatGrayStyleSheet)
131         if socorroValue>=70:
132             self.ui.ProbSocorro07.setStyleSheet(self.
    formatPurpleStyleSheet)
133         else:
134             self.ui.ProbSocorro07.setStyleSheet(self.
    formatGrayStyleSheet)
135         if socorroValue>=80:
136             self.ui.ProbSocorro08.setStyleSheet(self.
    formatPurpleStyleSheet)
137         else:
138             self.ui.ProbSocorro08.setStyleSheet(self.
    formatGrayStyleSheet)
139         if socorroValue>=90:
140             self.ui.ProbSocorro09.setStyleSheet(self.
    formatPurpleStyleSheet)
141         else:
142             self.ui.ProbSocorro09.setStyleSheet(self.
    formatGrayStyleSheet)
143         if socorroValue==100:
144             self.ui.ProbSocorro10.setStyleSheet(self.
    formatPurpleStyleSheet)
145         else:
146             self.ui.ProbSocorro10.setStyleSheet(self.
    formatGrayStyleSheet)
147
148
149     def UpdateProbAjuda(self, ajudaValue):
150         self.formatGrayStyleSheet = "QPushButton {\n" "background-
    color: rgb(165, 165, 175);\n"  "border-radius: 5px;\n" "}"
151         self.formatPurpleStyleSheet = "QPushButton {\n" "background
```

```
    -color: rgb(254, 66, 139);\n"  "border-radius: 5px;\n" "}"
152         self.ui.Label_ProbAjuda.setText(str(ajudaValue))
153         if ajudaValue>=10:
154             self.ui.ProbAjuda01.setStyleSheet(self.
    formatPurpleStyleSheet)
155         else:
156             self.ui.ProbAjuda01.setStyleSheet(self.
    formatGrayStyleSheet)
157         if ajudaValue>=20:
158             self.ui.ProbAjuda02.setStyleSheet(self.
    formatPurpleStyleSheet)
159         else:
160             self.ui.ProbAjuda02.setStyleSheet(self.
    formatGrayStyleSheet)
161         if ajudaValue>=30:
162             self.ui.ProbAjuda03.setStyleSheet(self.
    formatPurpleStyleSheet)
163         else:
164             self.ui.ProbAjuda03.setStyleSheet(self.
    formatGrayStyleSheet)
165         if ajudaValue>=40:
166             self.ui.ProbAjuda04.setStyleSheet(self.
    formatPurpleStyleSheet)
167         else:
168             self.ui.ProbAjuda04.setStyleSheet(self.
    formatGrayStyleSheet)
169         if ajudaValue>=50:
170             self.ui.ProbAjuda05.setStyleSheet(self.
    formatPurpleStyleSheet)
171         else:
172             self.ui.ProbAjuda05.setStyleSheet(self.
    formatGrayStyleSheet)
173         if ajudaValue>=60:
174             self.ui.ProbAjuda06.setStyleSheet(self.
    formatPurpleStyleSheet)
```

```
175         else:
176             self.ui.ProbAjuda06.setStyleSheet(self.
    formatGrayStyleSheet)
177         if ajudaValue>=70:
178             self.ui.ProbAjuda07.setStyleSheet(self.
    formatPurpleStyleSheet)
179         else:
180             self.ui.ProbAjuda07.setStyleSheet(self.
    formatGrayStyleSheet)
181         if ajudaValue>=80:
182             self.ui.ProbAjuda08.setStyleSheet(self.
    formatPurpleStyleSheet)
183         else:
184             self.ui.ProbAjuda08.setStyleSheet(self.
    formatGrayStyleSheet)
185         if ajudaValue>=90:
186             self.ui.ProbAjuda09.setStyleSheet(self.
    formatPurpleStyleSheet)
187         else:
188             self.ui.ProbAjuda09.setStyleSheet(self.
    formatGrayStyleSheet)
189         if ajudaValue==100:
190             self.ui.ProbAjuda10.setStyleSheet(self.
    formatPurpleStyleSheet)
191         else:
192             self.ui.ProbAjuda10.setStyleSheet(self.
    formatGrayStyleSheet)
193
194     def UpdateProbNenhuma(self, nenhumaValue):
195         self.formatGrayStyleSheet = "QPushButton {\n" "background-
    color: rgb(165, 165, 175);\n"  "border-radius: 5px;\n" "}"
196         self.formatPurpleStyleSheet = "QPushButton {\n" "background
    -color: rgb(254, 66, 139);\n"  "border-radius: 5px;\n" "}"
197         self.ui.Label_ProbNenhuma.setText(str(nenhumaValue))
198         if nenhumaValue>=10:
```

```
199             self.ui.ProbNenhuma01.setStyleSheet(self.
    formatPurpleStyleSheet)
200         else:
201             self.ui.ProbNenhuma01.setStyleSheet(self.
    formatGrayStyleSheet)
202         if nenhumaValue>=20:
203             self.ui.ProbNenhuma02.setStyleSheet(self.
    formatPurpleStyleSheet)
204         else:
205             self.ui.ProbNenhuma02.setStyleSheet(self.
    formatGrayStyleSheet)
206         if nenhumaValue>=30:
207             self.ui.ProbNenhuma03.setStyleSheet(self.
    formatPurpleStyleSheet)
208         else:
209             self.ui.ProbNenhuma03.setStyleSheet(self.
    formatGrayStyleSheet)
210         if nenhumaValue>=40:
211             self.ui.ProbNenhuma04.setStyleSheet(self.
    formatPurpleStyleSheet)
212         else:
213             self.ui.ProbNenhuma04.setStyleSheet(self.
    formatGrayStyleSheet)
214         if nenhumaValue>=50:
215             self.ui.ProbNenhuma05.setStyleSheet(self.
    formatPurpleStyleSheet)
216         else:
217             self.ui.ProbNenhuma05.setStyleSheet(self.
    formatGrayStyleSheet)
218         if nenhumaValue>=60:
219             self.ui.ProbNenhuma06.setStyleSheet(self.
    formatPurpleStyleSheet)
220         else:
221             self.ui.ProbNenhuma06.setStyleSheet(self.
    formatGrayStyleSheet)
```

```
222         if nenhumaValue>=70:
223             self.ui.ProbNenhuma07.setStyleSheet(self.
    formatPurpleStyleSheet)
224         else:
225             self.ui.ProbNenhuma07.setStyleSheet(self.
    formatGrayStyleSheet)
226         if nenhumaValue>=80:
227             self.ui.ProbNenhuma08.setStyleSheet(self.
    formatPurpleStyleSheet)
228         else:
229             self.ui.ProbNenhuma08.setStyleSheet(self.
    formatGrayStyleSheet)
230         if nenhumaValue>=90:
231             self.ui.ProbNenhuma09.setStyleSheet(self.
    formatPurpleStyleSheet)
232         else:
233             self.ui.ProbNenhuma09.setStyleSheet(self.
    formatGrayStyleSheet)
234         if nenhumaValue==100:
235             self.ui.ProbNenhuma10.setStyleSheet(self.
    formatPurpleStyleSheet)
236         else:
237             self.ui.ProbNenhuma10.setStyleSheet(self.
    formatGrayStyleSheet)
238
239     def ServerDesconectado(self):
240         # MODO DE OPERACAO: 0 - Desconectado
241         self.operation_mode = 0
242
243         self.ui.Label_UserName.setText(''' <span style="color:#
    ffffff;">OlA, </span>
244                                         <span style="color:#85
    e1ff;">'''+Settings.nome_usuario+'''</span><span style="color:#
    ffffff;">!</span>''')
245
```

```
246         #SOS Server
247         self.ui.Button_Server.setStyleSheet("QPushButton {\n" "
    background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1,
    y2:1, stop:0 rgba(131, 140, 226, 255), stop:1 rgba(156, 133,
    187, 255));\n"
248         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}\
    n" "QPushButton:hover {\n" "background-color: rgb(133, 225, 255)
    ;\n"
249         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}\
    n"
250         "QPushButton:pressed {\n" "background-color: rgb(103, 195,
    225);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
251         self.ui.Button_Server.setText("CONECTAR")
252         self.ui.Label_Conectando.setText("")
253         #Modo AutomAtico
254         self.ui.Label_Time_Auto.setText("")
255         self.ui.Button_Auto.setStyleSheet("QPushButton {\n" "
    background-color: rgb(165, 165, 175);\n"
256         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}"
    )
257         self.ui.Label_Gravando_Auto.setText("")
258         self.ui.Voice_Auto.setValue(0)
259         #Modo Manual
260         self.ui.Label_Time_Manual.setText("")
261         self.ui.Button_Manual.setStyleSheet("QPushButton {\n" "
    background-color: rgb(165, 165, 175);\n"
262         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}"
    )
263         self.ui.Label_Gravando_Manual.setText("")
264         self.ui.Voice_Manual.setValue(0)
265         #IdentificaCAo de Palavra
266         self.ui.Label_Identificado.setText("NAO IDENTIFICADO")
267         self.ui.Label_Identificado.setStyleSheet("QPushButton {\n"
    "background-color: rgb(165, 165, 175);\n"
```

```
268         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}"
    )
269         self.ui.Label_Palavra.setText("NAO IDENTIFICADO")
270         self.ui.Label_Palavra.setStyleSheet("QPushButton {\n" "
    background-color: rgb(165, 165, 175);\n"
271         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}"
    )
272         self.ui.Label_ProbSocorro.setText(str(Settings.prob_socorro
    ))
273         self.ui.Label_ProbAjuda.setText(str(Settings.prob_ajuda))
274         self.ui.Label_ProbNenhuma.setText(str(Settings.prob_nenhuma
    ))
275
276         self.UpdateProbSocorro(0)
277         self.UpdateProbAjuda(0)
278         self.UpdateProbNenhuma(0)
279
280
281
282     def ServerConectando(self):
283
284         # MODOS DE OPERACAO
285
286         ## 0 - Desconectado
287         ## 1 - Conectado
288         ## 2 - Conectado e em modo automatico
289         ## 3 - Conectado e em modo manual
290
291         if self.operation_mode == 0:
292             #SOS Server
293             self.ui.Label_Conectando.setText("conectando...")
294             self.ui.Button_Server.setStyleSheet("QPushButton {\n" "
    background-color: rgb(103, 195, 225);\n"
295             "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
```

```
296
297             #Enviar ao servidor pedido de conexAo
298             Settings.conectar_servidor = True
299             try:
300                 cliente.running = True
301                 cliente.start()
302             except:
303                 Settings.conectado_servidor = False
304                 self.ServerDesconectado()
305                 return
306
307             #Receber esse dado do servidor
308             Settings.conectado_servidor = True
309
310             if Settings.conectado_servidor:
311                 self.operation_mode = 1
312                 self.ServerConectado()
313
314         elif self.operation_mode == 1:
315             #SOS Server
316             self.ui.Label_Conectando.setText("desconectando...")
317             self.ui.Button_Server.setStyleSheet("QPushButton {\n" "
    background-color: rgb(103, 195, 225);\n"
318             "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
319
320             #Enviar ao servidor pedido de desconexAo
321             Settings.conectar_servidor = False
322
323             try:
324                 cliente.running = False
325             except:
326                 Settings.conectado_servidor = True
327                 return
328
```

```
329             #Receber esse dado do servidor
330             Settings.conectado_servidor = False
331 
332             if not Settings.conectado_servidor:
333                 self.operation_mode = 0
334                 self.ServerDesconectado()
335 
336         elif self.operation_mode == 2:
337             return
338         elif self.operation_mode == 3:
339             return
340         else:
341             return
342 
343     def ServerConectado(self):
344         #SOS Server
345         self.ui.Label_Conectando.setText("CONECTADO")
346         self.ui.Label_Conectando.setFont("Nunito ExtraBold")
347         self.ui.Button_Server.setText("DESCONECTAR")
348         self.ui.Button_Server.setStyleSheet("QPushButton {\n" "
    background-color: rgb(103, 195, 225);\n" "color: rgb(255, 255,
    255);\n" "border-radius: 10px;\n" "}\n"
349         "QPushButton:hover {\n" "background-color: rgb(133, 225,
    255);\n" "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n"
     "}\n"
350         "QPushButton:pressed {\n" "background-color: rgb(103, 195,
    225);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
351         #Modo AutomAtico
352         self.ui.Label_Time_Auto.setText("")
353         self.ui.Button_Auto.setText("INICIAR")
354         self.ui.Button_Auto.setStyleSheet("QPushButton {\n" "
    background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1,
    y2:1, stop:0 rgba(131, 140, 226, 255), stop:1 rgba(156, 133,
    187, 255));\n"
```

```
355         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}\
    n" "QPushButton:hover {\n" "background-color: rgb(133, 225, 255)
    ;\n"
356         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}\
    n"
357         "QPushButton:pressed {\n" "background-color: rgb(103, 195,
    225);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
358         self.ui.Label_Gravando_Auto.setText("")
359         self.ui.Voice_Auto.setValue(0)
360         #Modo Manual
361         self.ui.Label_Time_Manual.setText("")
362         self.ui.Button_Manual.setText("INICIAR")
363         self.ui.Button_Manual.setStyleSheet("QPushButton {\n" "
    background-color: qlineargradient(spread:pad, x1:0, y1:0, x2:1,
    y2:1, stop:0 rgba(131, 140, 226, 255), stop:1 rgba(156, 133,
    187, 255));\n"
364         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}\
    n" "QPushButton:hover {\n" "background-color: rgb(133, 225, 255)
    ;\n"
365         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}\
    n"
366         "QPushButton:pressed {\n" "background-color: rgb(103, 195,
    225);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
367         self.ui.Label_Gravando_Manual.setText("")
368         self.ui.Voice_Manual.setValue(0)
369
370         self.UpdateProbSocorro(0)
371         self.UpdateProbAjuda(0)
372         self.UpdateProbNenhuma(0)
373
374     def GravarAuto(self):
375         if self.operation_mode == 1:
376             #Enviar ao servidor pedido de gravaCAo
```

```
377             Settings.gravar_auto = True
378             tmpStartReportTime = time.time()
379             while not Settings.gravando_auto:
380                 if (time.time() - tmpStartReportTime)>2:
381                     print("Exceded time to receive data: timelapse"
    , (time.time() - tmpStartReportTime))
382                     Settings.gravar_auto = False
383                     return
384             self.operation_mode = 2
385             self.GravandoAuto()
386 
387         elif self.operation_mode == 2:
388             #Enviar ao servidor pedido de parar gravaCAo
389             Settings.gravar_auto = False
390             while Settings.gravando_auto:
391                 if (time.time() - tmpStartReportTime)>3:
392                     print("Exceded time to receive data: timelapse"
    , (time.time() - tmpStartReportTime))
393                     Settings.gravar_auto = True
394                     return
395             self.operation_mode = 1
396             self.ServerConectado()
397 
398         else:
399             return
400 
401     def GravandoAuto(self):
402         self.updatingScreen = True
403         #Modo AutomAtico
404         self.ui.Label_Time_Auto.setText(str(Settings.
    counter_time_voice_auto))
405         self.ui.Button_Auto.setText("PARAR")
406         self.ui.Button_Auto.setStyleSheet("QPushButton {\n" "
    background-color: rgb(255, 0, 127);\n" "color: rgb(165, 165,
    175);\n" "border-radius: 10px;\n" "}\n"
```

```
407         "QPushButton:hover {\n" "background-color: rgb(254, 66,
    139);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}\n"
408         "QPushButton:pressed {\n" "background-color: rgb(255, 0,
    127);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
409         self.ui.Label_Gravando_Auto.setText("gravando...")
410         # self.ui.Voice_Auto.setValue(Settings.voice_intensity_auto
    )
411         #Modo Manual
412         self.ui.Label_Time_Manual.setText("")
413         self.ui.Button_Manual.setStyleSheet("QPushButton {\n" "
    background-color: rgb(165, 165, 175);\n"
414         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}"
    )
415         self.ui.Label_Gravando_Manual.setText("")
416         # self.ui.Voice_Manual.setValue(0)
417 
418         logger.info(' - Updating UI with: ' + str(Settings.
    prob_ajuda) + ";" + str(Settings.prob_socorro)+ ";" + str(
    Settings.prob_nenhuma))
419         if Settings.palavra_identificado == True:
420             # IdentificaCAo de Palavra
421             self.ui.Label_Identificado.setText("IDENTIFICADO")
422             self.ui.Label_Identificado.setStyleSheet("QPushButton
    {\n" "background-color: rgb(254, 66, 139);\n"
423             "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n"
    "}")
424         else:
425             self.ui.Label_Identificado.setText("NAO IDENTIFICADO")
426             self.ui.Label_Identificado.setStyleSheet("QPushButton
    {\n" "background-color: rgb(165, 165, 175);\n"
427             "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n"
     "}")
428 
```

```
429         if Settings.socorro_identificado == True:
430             # IdentificaCAo de Socorro
431             self.ui.Label_Palavra.setText("SOCORRO")
432             self.ui.Label_Palavra.setStyleSheet("QPushButton {\n" "
    background-color: rgb(254, 66, 139);\n"
433             "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n"
     "}")
434         elif Settings.ajuda_identificado == True:
435             self.ui.Label_Palavra.setText("AJUDA")
436             self.ui.Label_Palavra.setStyleSheet("QPushButton {\n" "
    background-color: rgb(254, 66, 139);\n"
437             "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n"
     "}")
438         else:
439             self.ui.Label_Palavra.setText("NAO IDENTIFICADO")
440             self.ui.Label_Palavra.setStyleSheet("QPushButton {\n" "
    background-color: rgb(165, 165, 175);\n"
441             "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n"
     "}")
442
443
444         self.UpdateProbSocorro(Settings.prob_socorro)
445         self.UpdateProbAjuda(Settings.prob_ajuda)
446         self.UpdateProbNenhuma(Settings.prob_nenhuma)
447
448         self.updatingScreen = False
449
450
451     def GravarManual(self):
452         if self.operation_mode == 1:
453             #Enviar ao servidor pedido de gravaCAo
454             Settings.gravar_manual = True
455             #Receber esse dado do servidor
456             Settings.gravando_manual = True
457
```

```
458             if Settings.gravando_manual:
459                 self.operation_mode = 3
460                 self.GravandoManual()
461 
462         elif self.operation_mode == 3:
463             #Enviar ao servidor pedido de parar gravaCAo
464             Settings.gravar_manual = False
465             #Receber esse dado do servidor
466             Settings.gravando_manual = False
467 
468             if not Settings.gravando_manual:
469                 self.operation_mode = 1
470                 self.ServerConectado()
471         else:
472             return
473 
474     def GravandoManual(self):
475         #Modo AutomAtico
476         self.ui.Label_Time_Auto.setText("")
477         self.ui.Button_Auto.setStyleSheet("QPushButton {\n" "
    background-color: rgb(165, 165, 175);\n"
478         "color: rgb(255, 255, 255);\n" "border-radius: 10px;\n" "}"
    )
479         self.ui.Label_Gravando_Auto.setText("")
480         self.ui.Voice_Auto.setValue(0)
481         #Modo Manual
482         self.ui.Label_Time_Manual.setText(Settings.
    counter_time_voice_manual)
483         self.ui.Button_Manual.setText("PARAR")
484         self.ui.Button_Manual.setStyleSheet("QPushButton {\n" "
    background-color: rgb(255, 0, 127);\n" "color: rgb(165, 165,
    175);\n" "border-radius: 10px;\n" "}\n"
485         "QPushButton:hover {\n" "background-color: rgb(254, 66,
    139);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}\n"
```

```
486         "QPushButton:pressed {\n" "background-color: rgb(255, 0,
    127);\n" "color: rgb(165, 165, 175);\n" "border-radius: 10px;\n"
     "}")
487         self.ui.Label_Gravando_Manual.setText("gravando...")
488         self.ui.Voice_Manual.setValue(Settings.
    voice_intensity_manual)
489 
490         if Settings.palavra_identificado == True:
491 
492             if Settings.socorro_identificado == True:
493                 #IdentificaCAo de Palavra
494                 self.ui.Label_Identificado.setText("IDENTIFICADO")
495                 self.ui.Label_Identificado.setStyleSheet("
    QPushButton {\n" "background-color: rgb(254, 66, 139);\n"
496                 "color: rgb(255, 255, 255);\n" "border-radius: 10px
    ;\n" "}")
497                 self.ui.Label_Palavra.setText("SOCORRO")
498                 self.ui.Label_Palavra.setStyleSheet("QPushButton {\
    n" "background-color: rgb(254, 66, 139);\n"
499                 "color: rgb(255, 255, 255);\n" "border-radius: 10px
    ;\n" "}")
500             elif Settings.ajuda_identificado == True:
501                 #IdentificaCAo de Palavra
502                 self.ui.Label_Identificado.setText("IDENTIFICADO")
503                 self.ui.Label_Identificado.setStyleSheet("
    QPushButton {\n" "background-color: rgb(254, 66, 139);\n"
504                 "color: rgb(255, 255, 255);\n" "border-radius: 10px
    ;\n" "}")
505                 self.ui.Label_Palavra.setText("AJUDA")
506                 self.ui.Label_Palavra.setStyleSheet("QPushButton {\
    n" "background-color: rgb(254, 66, 139);\n"
507                 "color: rgb(255, 255, 255);\n" "border-radius: 10px
    ;\n" "}")
508         else:
509                 #IdentificaCAo de Palavra
```

```
510                 self.ui.Label_Identificado.setText("NAO
    IDENTIFICADO")
511                 self.ui.Label_Identificado.setStyleSheet("
    QPushButton {\n" "background-color: rgb(165, 165, 175);\n"
512                 "color: rgb(255, 255, 255);\n" "border-radius: 10px
    ;\n" "}")
513                 self.ui.Label_Palavra.setText("NAO IDENTIFICADO")
514                 self.ui.Label_Palavra.setStyleSheet("QPushButton {\
    n" "background-color: rgb(165, 165, 175);\n"
515                 "color: rgb(255, 255, 255);\n" "border-radius: 10px
    ;\n" "}")
516
517         self.UpdateProbSocorro(Settings.prob_socorro)
518         self.UpdateProbAjuda(Settings.prob_ajuda)
519         self.UpdateProbNenhuma(Settings.prob_nenhuma)
520
521     def Configurar(self):
522         #So vou para a tela de configuraCOes se estiver no modo 0(
    desconectado) ou modo 1(conectado)
523         if self.operation_mode == 0:
524             # SHOW CONFIGURACOES
525             self.main = Configuracoes()
526             self.main.show()
527             # CLOSE MAIN WINDOW
528             self.close()
529         elif self.operation_mode == 1:
530             # SHOW CONFIGURACOES
531             self.main = Configuracoes()
532             self.main.show()
533             # CLOSE MAIN WINDOW
534             self.close()
535         else:
536             return
537
538     def EnviarEmail(self):
```

```
539         if self.ui.CheckBox_EMail.isChecked():
540             Settings.notif_email = True
541         else:
542             Settings.notif_email = False
543 
544     def EnviarTelegram(self):
545         if self.ui.CheckBox_Telegram.isChecked():
546             Settings.notif_Telegram = True
547         else:
548             Settings.notif_Telegram = False
549 
550 
551 
552 ################################################################
553 #                        Thread de Envio                       #
554 ################################################################
555 
556 class SockClient(threading.Thread):
557     """ Classe para prover cliente de conexAo socket usando threads
558         Sintaxe: cliente = SockClient(host,port)
559     """
560     def __init__(self, host, port):
561         # MEtodo para construir a classe
562         threading.Thread.__init__(self)
563         self.running = False
564         self.host = host
565         self.port = port
566 
567         # Configurando socket para TCP
568         self.sock = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
569         self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR
    , 1)
570         self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_RCVTIMEO,
     self._seconds_to_sockopt_format(5))
```

```
571         self.dest = (self.host, self.port)
572 
573     def _seconds_to_sockopt_format(self, seconds):
574         if os.name == "nt":
575             return int(seconds * 1000)
576 
577     def stop(self):
578         # MEtodo para interromper a thread do cliente.
579         self.running = False
580         self.sock.close()
581         logger.critical('- Sem conexAo com o servidor remoto.')
582 
583     def _send(self, received_data):
584         # MEtodo para envio de comandos para o servidor.
585         self.received_data = received_data
586         self.sock.send(self.received_data)
587 
588     def run(self):
589         # MEtodo obrigatorio para a thread aceitar ativaCAo atravEs
    de 'start()'.
590         # Ativa o cliente socket.
591         # try:
592         self.sock.connect(self.dest)
593         print('connecting to {} port {}'.format(*self.dest))
594 
595         # Loop infinito para manter a execuCAo do cliente
596         while self.running:
597             if (Settings.gravar_auto == True and Settings.
    gravando_auto == False) or (Settings.gravar_auto == False and
    Settings.gravando_auto == True):
598                 # Send data
599                 filetosend = Settings.EncoderFile()
600                 cliente._send(filetosend.encode())
601                 print("Data sent by client")
602 
```

```
603             self.sock.setblocking(0)
604             ready = select.select([self.sock], [], [], 2)
605             if ready[0]:
606                 try:
607                     received_data = self.sock.recv(2048).decode()
608                 except ConnectionResetError:
609                     logger.critical('- Connection error.')
610                     received_data=""
611                 except:
612                     logger.critical('- General error to recv data.'
    )
613                     received_data=""
614 
615                 if received_data:
616                     if window.main.updatingScreen==False:
617                         print("Data received by client")
618                         try:
619                             jsonSettings = json.loads(received_data
    )
620                             Settings.DecoderFile(jsonSettings)
621                             received_data = ""
622                         except JSONDecodeError:
623                             logger.critical('- Error decoder data.'
    )
624 
625 
626                         logger.info(' - Data received.' + str(
    Settings.prob_ajuda) + ";" + str(Settings.prob_socorro) + ";" +
    str(Settings.prob_nenhuma))
627                         #Settings = cs.ClassSettings(jsonSettings)
628                         if Settings.gravando_auto:
629                             window.main.GravandoAuto()
630                         else:
631                             window.main.ServerConectado()
632 
```

```
633                     else:
634                         logger.critical('- Could not update Screen'
    )
635                     # try:
636                     # except:
637                     #     logger.critical('- Error decoder data.')
638 
639     def close(self):
640         self.running = False
641 
642 
643 def timestamp(tipo):
644     """ Funcao para gerar timestamp de uso geral """
645 
646     if tipo == 'hmsms':
647         return str('{0:%H%M%S%f}'.format(datetime.datetime.now()))
    [0:9]
648     elif tipo == 'ms':
649         return str('{:%f}'.format(datetime.datetime.now()))[0:3]
650 
651 
652 ##################################################################
653 #                      TELA DE CONFIGURACOES                     #
654 ##################################################################
655 
656 # ConfiguraCOes
657 class Configuracoes(QMainWindow):
658     def __init__(self):
659         QMainWindow.__init__(self)
660         self.ui = Ui_Configuracoes()
661         self.ui.setupUi(self)
662 
663         self.ui.Label_UserName.setText(''' <span style="color:#
    ffffff;">OlA, </span>
664                                            <span style="color:#85
```

```
e1ff;">'''+Settings.nome_usuario+'''</span><span style="color:#
ffffff;">!</span>''')
665         self.ui.Show_Name.setText(Settings.nome_usuario)
666         self.ui.Show_Email.setText(Settings.email_usuario)
667         self.ui.Show_Number.setText(Settings.telefone_usuario)
668 
669         ## REMOVE TITLE BAR
670         self.setWindowFlag(QtCore.Qt.FramelessWindowHint)
671         self.setAttribute(QtCore.Qt.WA_TranslucentBackground)
672 
673         self.ui.Config_Button.clicked.connect(self.VoltarMainWindow
)
674 
675         #Se pressionar botAo "x", fechar a pAgina
676         self.ui.Exit_Button.clicked.connect(self.close)
677 
678         self.ui.Ok_Name.clicked.connect(self.MudarNome)
679         self.ui.Ok_Email.clicked.connect(self.MudarEmail)
680         self.ui.Ok_Number.clicked.connect(self.MudarNumero)
681 
682     def MudarNome(self):
683         if self.ui.Edit_Name.text():
684             Settings.nome_usuario = self.ui.Edit_Name.text()
685             self.ui.Edit_Name.clear()
686             self.ui.Show_Name.setText(Settings.nome_usuario)
687             self.ui.Label_UserName.setText(''' <span style="color:#
ffffff;">OlA, </span>
688                                            <span style="color
:#85e1ff;">'''+Settings.nome_usuario+'''</span><span style="
color:#ffffff;">!</span>''')
689         else:
690             return
691 
692     def MudarEmail(self):
693         if self.ui.Edit_Email.text():
```

```
694             Settings.email_usuario = self.ui.Edit_Email.text()
695             self.ui.Edit_Email.clear()
696             self.ui.Show_Email.setText(Settings.email_usuario)
697         else:
698             return
699 
700     def MudarNumero(self):
701         if self.ui.Edit_Number.text():
702             Settings.telefone_usuario = self.ui.Edit_Number.text()
703             self.ui.Edit_Number.clear()
704             self.ui.Show_Number.setText(Settings.telefone_usuario)
705         else:
706             return
707 
708     def VoltarMainWindow(self):
709         # SHOW MAIN WINDOW
710         self.main = MainWindow()
711         self.main.show()
712 
713         # CLOSE CONFIGURACOES
714         self.close()
715 
716 ########################################################################
717 #                        TELA DE INICIALIZACAO                         #
718 ########################################################################
719 
720 # SPLASH SCREEN
721 class SplashScreen(QMainWindow):
722     def __init__(self):
723         QMainWindow.__init__(self)
724         self.ui = Ui_SplashScreen()
725         self.ui.setupUi(self)
726 
727         ## UI ==> INTERFACE CODES
728         #
```

```
    ########################################################

729
730         ## REMOVE TITLE BAR
731         self.setWindowFlag(QtCore.Qt.FramelessWindowHint)
732         self.setAttribute(QtCore.Qt.WA_TranslucentBackground)
733
734         ## DROP SHADOW EFFECT
735         self.shadow = QGraphicsDropShadowEffect(self)
736         self.shadow.setBlurRadius(20)
737         self.shadow.setXOffset(0)
738         self.shadow.setYOffset(0)
739         self.shadow.setColor(QColor(0, 0, 0, 60))
740         self.ui.dropShadowframe.setGraphicsEffect(self.shadow)
741
742         ## QTIMER ==> START
743         self.counter = 0
744         self.timer = QtCore.QTimer()
745         self.timer.timeout.connect(self.progress)
746         # TIMER IN MILLISECONDS
747         self.timer.start(35)
748
749         # CHANGE DESCRIPTION
750
751         # Initial Text
752         self.ui.Label_Description.setText("Identificador de
    Socorros")
753
754         # Change Texts
755         QtCore.QTimer.singleShot(3000, lambda: self.ui.
    Label_Description.setText("Carregando Interface"))
756
757
758         ## SHOW ==> MAIN WINDOW
759         ########################################################
760         self.show()
```

```
761         ## ==> END ##
762 
763     ## ==> APP FUNCTIONS
764     ########################################################
765     def progress(self):
766 
767         # SET VALUE TO PROGRESS BAR
768         self.ui.progressBar.setValue(self.counter)
769 
770         # CLOSE SPLASH SCREE AND OPEN APP
771         if self.counter > 100:
772             # STOP TIMER
773             self.timer.stop()
774 
775             # SHOW MAIN WINDOW
776             self.main = MainWindow()
777             self.main.show()
778 
779             # CLOSE SPLASH SCREEN
780             self.close()
781 
782         # INCREASE COUNTER
783         self.counter += 1
784 
785 
786 ########################################################################
787 #                                MAIN                                  #
788 ########################################################################
789 
790 if __name__ == "__main__":
791 
792     DEBUGGING = False
793     if 'pydevd' in sys.modules:
794         print ("Debugger Mode")
795         DEBUGGING = True
```

```
796     else:
797         args = docopt(__doc__)
798 
799     Settings = cs.ClassSettings()
800     if not DEBUGGING:
801         if (args['--in'] is not None):
802             str_settings = args['--in']
803     else:
804         str_settings = 'Settings.json'
805 
806     try:
807         with open(str_settings, 'r') as fin:
808             readInput = fin.read()
809             jsonSettings = json.loads(readInput)
810             Settings.DecoderFile(jsonSettings)
811     except:
812         sys.exit('FATAL: Problems to read a the json file')
813 
814     # CONFIGURACAO DO CLIENTE
815     # EndereCo IP da mAquna REMOTA
816     ip_remoto = '127.0.0.1'
817     port_remoto = 10000
818 
819     cliente = SockClient(ip_remoto, port_remoto)
820     cliente.running = False
821 
822     app = QApplication(sys.argv)
823     window = SplashScreen()
824 
825     # ##############################################################
826 
827     # Criando variAveis de ambiente e verificando diretorios.
828     # Detectando em qual diretorio a aplicaCAo estA sendo executada
829     dir_atual = os.getcwd()
830 
```

```
831     # Verificando se existe o diretorio de 'log'.
832     # Se nAo existir - Criar.
833     log_dir = dir_atual + '/log'
834     if not os.path.exists(log_dir):
835         os.mkdir(log_dir)
836 
837     # Criando o arquivo 'simul.log'.
838     nome_arq_log = log_dir + '/' + 'sock_' + timestamp('hmsms') + '
    .log'
839     with open(nome_arq_log, 'w') as arq_log:
840         arq_log.write('\n')
841 
842     # # VariAvel com o nome da aplicaCAo
843     # #nome_prog = sys.argv[0].split('./')[1]
844     nome_prog ='SOS connect'
845 
846     # Ajustando recursos de logging
847     logger = logging.getLogger(nome_prog)
848     handler = logging.FileHandler(nome_arq_log)
849     formatter = logging.Formatter('%(asctime)s %(levelname)s %(
    message)s')
850     handler.setFormatter(formatter)
851     logger.addHandler(handler)
852 
853     # Setando para debug
854     logger.setLevel(logging.DEBUG)
855 
856     sys.exit(app.exec_())
857     #  sys.exit(app.exec_())
```

A.3 Empacotamento de Variáveis. Fonte: Autoria própria

```
1 import numpy as np
```

```
import json
from json import JSONEncoder

class ClassSettings():
    def init(self):
        self.conectarservidor = False
        self.gravarauto = False
        self.gravarmanual = False

        self.nomeusuario = "Beatrice"
        self.emailusuario = "exemplo@gmail.com"
        self.telefoneusuario = "819XXXX-XXXX"
        self.notifemail = False
        self.notifTelegram = False

        self.conectadoservidor = False

        self.gravandoauto = False
        self.countertimevoiceauto = 0
        self.voiceintensityauto = 0

        self.gravandomanual = False
        self.countertimevoicemanual = 0
        self.voiceintensitymanual = 0

        self.probsocorro =0
        self.probajuda = 0
        self.probnenhuma = 0

        self.palavraidentificado = False
        self.socorroidentificado = 0
        self.ajudaidentificado = 0

    def DecoderFile(self,JsonFile):
        self.conectarservidor = JsonFile['SendtoServerParameters'][
```

```
'conectarservidor']
37          self.gravarauto = JsonFile['SendtoServerParameters']['
gravarauto']
38          self.gravarmanual = JsonFile['SendtoServerParameters']['
gravarmanual']
39
40          self.nomeusuario = JsonFile['SendtoServerParameters']['
SettingsParameters']['nomeusuario']
41          self.emailusuario = JsonFile['SendtoServerParameters']['
SettingsParameters']['emailusuario']
42          self.telefoneusuario = JsonFile['SendtoServerParameters']['
SettingsParameters']['telefoneusuario']
43          self.notifemail = JsonFile['SendtoServerParameters']['
SettingsParameters']['notifemail']
44          self.notifTelegram = JsonFile['SendtoServerParameters']['
SettingsParameters']['notifTelegram']
45
46          self.conectadoservidor = JsonFile['
ReceiveFromServerParameters']['conectadoservidor']
47
48          self.gravandoauto = JsonFile['ReceiveFromServerParameters'
]['AutoParameters']['gravandoauto']
49          self.countertimevoiceauto = JsonFile['
ReceiveFromServerParameters']['AutoParameters']['
countertimevoiceauto']
50          self.voiceintensityauto = JsonFile['
ReceiveFromServerParameters']['AutoParameters']['
voiceintensityauto']
51
52          self.gravandomanual = JsonFile['ReceiveFromServerParameters
']['ManualParameters']['gravandomanual']
53          self.countertimevoicemanual = JsonFile['
ReceiveFromServerParameters']['ManualParameters']['
countertimevoicemanual']
54          self.voiceintensitymanual = JsonFile['
```

```
ReceiveFromServerParameters'][''ManualParameters'][''
voiceintensitymanual']

        self.probsocorro = JsonFile['ReceiveFromServerParameters'][
'PrecisionParameters'][''probsocorro']
        self.probajuda = JsonFile['ReceiveFromServerParameters'][''
PrecisionParameters'][''probajuda']
        self.probnenhuma = JsonFile['ReceiveFromServerParameters'][
'PrecisionParameters'][''probnenhuma']

        self.palavraidentificado = JsonFile['
ReceiveFromServerParameters'][''IdentificationParameters'][''
palavraidentificado']
        self.socorroidentificado = JsonFile['
ReceiveFromServerParameters'][''IdentificationParameters'][''
socorroidentificado']
        self.ajudaidentificado = JsonFile['
ReceiveFromServerParameters'][''IdentificationParameters'][''
ajudaidentificado']


    def EncoderFile(self):

        class GenerateEncoderFile:
            def init(self, SendtoServerParameters,
ReceiveFromServerParameters):
                self.SendtoServerParameters =
SendtoServerParameters
                self.ReceiveFromServerParameters =
ReceiveFromServerParameters
            def tojson(self):
                return json.dumps(self, indent = 4, default=lambda
o: o.dict)

        class SendtoServerParameters:
```

```
75          def init(self, conectarservidor, gravarauto,
    gravarmanual, SettingsParameters):
76              self.conectarservidor = conectarservidor
77              self.gravarauto = gravarauto
78              self.gravarmanual = gravarmanual
79              self.SettingsParameters = SettingsParameters
80
81      class SettingsParameters:
82          def init(self, nomeusuario, emailusuario,
    telefoneusuario, notifemail, notifTelegram):
83              self.nomeusuario =nomeusuario
84              self.emailusuario =emailusuario
85              self.telefoneusuario=telefoneusuario
86              self.notifemail =notifemail
87              self.notifTelegram =notifTelegram
88
89      class ReceiveFromServerParameters:
90          def init(self, conectadoservidor, AutoParameters,
    ManualParameters, PrecisionParameters, IdentificationParameters)
    :
91              self.conectadoservidor = conectadoservidor
92              self.AutoParameters = AutoParameters
93              self.ManualParameters = ManualParameters
94              self.PrecisionParameters = PrecisionParameters
95              self.IdentificationParameters =
    IdentificationParameters
96
97
98      class AutoParameters:
99          def init(self, gravandoauto, countertimevoiceauto,
    voiceintensityauto):
100             self.gravandoauto = gravandoauto
101             self.countertimevoiceauto = countertimevoiceauto
102             self.voiceintensityauto = voiceintensityauto
103
```

```
104         class ManualParameters:
105             def init(self, gravandomanual, countertimevoicemanual,
    voiceintensitymanual):
106                 self.gravandomanual = gravandomanual
107                 self.countertimevoicemanual =
    countertimevoicemanual
108                 self.voiceintensitymanual = voiceintensitymanual
109 
110         class PrecisionParameters:
111             def init(self, probsocorro, probajuda, probnenhuma):
112                 self.probsocorro = probsocorro
113                 self.probajuda = probajuda
114                 self.probnenhuma = probnenhuma
115 
116         class IdentificationParameters:
117             def init(self, palavraidentificado, socorroidentificado
    , ajudaidentificado):
118                 self.palavraidentificado = palavraidentificado
119                 self.socorroidentificado = socorroidentificado
120                 self.ajudaidentificado = ajudaidentificado
121 
122         SettingsParameters = SettingsParameters(self.nomeusuario,
    self.emailusuario, self.telefoneusuario, self.notifemail, self.
    notifTelegram)
123         SendtoServerParameters = SendtoServerParameters(self.
    conectarservidor,  self.gravarauto , self.gravarmanual,
    SettingsParameters)
124         IdentificationParameters = IdentificationParameters(self.
    palavraidentificado, self.socorroidentificado,self.
    ajudaidentificado)
125         PrecisionParameters = PrecisionParameters(self.probsocorro,
    self.probajuda, self.probnenhuma)
126         ManualParameters = ManualParameters(self.gravandomanual,
    self.countertimevoicemanual, self.voiceintensitymanual)
127         AutoParameters = AutoParameters(self.gravandoauto, self.
```

```
    countertimevoiceauto, self.voiceintensityauto)
128         ReceiveFromServerParameters = ReceiveFromServerParameters(
    self.conectadoservidor, AutoParameters, ManualParameters,
    PrecisionParameters, IdentificationParameters)
129         GenerateEncoderFile = GenerateEncoderFile(
    SendtoServerParameters, ReceiveFromServerParameters)
130         return GenerateEncoderFile.tojson()
```