UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

PEDRO TINO PINHEIRO FREITAS

SISTEMA DE AUTOMAÇÃO RESIDENCIAL PARA DEFICIENTES VISUAIS BASEADO EM RECONHECIMENTO DE VOZ

FORTALEZA

2019

PEDRO TINO PINHEIRO FREITAS

SISTEMA DE AUTOMAÇÃO RESIDENCIAL PARA DEFICIENTES VISUAIS BASEADO EM RECONHECIMENTO DE VOZ

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Arthur Plínio de Souza Braga
Co-orientador: Prof. Dr. Nícolas de Araújo Moreira

FORTALEZA
2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F937s Freitas, Pedro Tino Pinheiro.
Sistema de automação residencial para deficientes visuais baseado em reconhecimento de voz / Pedro Tino Pinheiro Freitas. – 2019.
78 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2019.
Orientação: Prof. Dr. Arthur Plínio de Souza Braga.
Coorientação: Prof. Dr. Nícolas de Araújo Moreira.

1. Automação Residencial. 2. Reconhecimento de Voz. 3. Acessibilidade. I. Título.

CDD 621.3

PEDRO TINO PINHEIRO FREITAS

SISTEMA DE AUTOMAÇÃO RESIDENCIAL PARA DEFICIENTES VISUAIS BASEADO EM RECONHECIMENTO DE VOZ

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Arthur Plínio de Souza Braga
Universidade Federal do Ceará (UFC)

_____
Dr. Eng. Nícolas de Araújo Moreira
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Paulo Peixoto Praça
Universidade Federal do Ceará (UFC)

Aos meus pais, família e amigos.

AGRADECIMENTOS

Aos meus pais, Lindoya e Marcus, por terem me dado todo o suporte que precisei durante toda a faculdade e o caminho até ela, desde menino até a minha formação.

Ao meu pai e a meu avô, Tino e Francisco, que quando tive dificuldades em casa sempre me ofereceram auxílio para passar por esses problemas, seja passando um tempo fora, ou mesmo apenas saindo para uma conversa.

À minha irmã, Camila, por ser minha companheira de ir assistir filmes nas sessões de meia noite de quinta-feira durante esses 5 anos.

À minha avó, Gleide, que é a pessoa mais especial desse mundo, e que forneceu a casa dela para os melhores encontros já vistos na história do curso de Engenharia Elétrica, conhecidos também como os maiores eventos de vatapá do mundo. Ver a comida dela no fim de semana e conversar com ela fazia tudo valer a pena.

Ao Marçal, por ter iniciado essa jornada comigo no Farias Brito, passado pela Engenharia Metalúrgica e pela aventura da lista de espera para entrar na Engenharia Elétrica. E, principalmente, por ter me incentivado tanto a passar pela minha melhor experiência na Universidade, o PET.

À AGUFC, o melhor grupo de amigos aleatórios (Airton, Bruno, Caio, Torres, Airton, Léo, Vanderson, e o nosso maior integrante, Alencar) que eu poderia ter encontrado na faculdade e que tanto me apoiaram em todos os momentos, inclusive me carregando nas costas em algumas cadeiras. Espero que nossa amizade dure o máximo que possa durar.

À Aline e Eloisa, por terem me dado amor como se eu fosse família por muito tempo, me dando uma casa para descansar nos dias em que eu tinha aula de 8h às 10h e a outra aula apenas 16h, ou mesmo quando precisava apenas de um lugar para me sentir bem e descansado, e por diversos outros motivos, principalmente apoio emocional e amizade.

Ao Sonoplastas, grupo dos meus amigos do colégio e novos próximos da faculdade, que sempre me deram apoio para conversar sobre seja lá o que fosse, e foram meus companheiros nos momentos de descontração. Pessoas especiais incluem, Lucas, Fernando,

Vítor, Matheus Vitor, João, Danilo, Marcos, Leonardo, Darcy Fontenele, Guilherme, Thiago, Haroldo, entre tantos outros.

Ao grupo PET Engenharia Elétrica, por ter me proporcionado dois dos melhores anos da minha vida, nos quais me construí como a pessoa que hoje sou, e espero ter passado o melhor que pude a todos que por lá passaram depois de mim, assim como fizeram comigo. Também por todas as noites que me permitiram virar lá fazendo projetos, pela caixinha de som que tocou por vários fins de semanas consecutivos durante as cadeiras de Instalações Prediais e Industriais, e por todo o suporte físico. Realmente não sei o que eu não devo a esse grupo.

À Melissa, por ter me dado suporte como amiga e companheira de coordenação da SEEL durante o que foram talvez os meses mais difíceis da minha graduação.

À minha terapeuta, Jordana, e ao meu psiquiatra, Severo, que me ajudaram, sobretudo nos últimos meses a conseguir me focar no que precisava terminar e a por me auxiliar a lidar com a minha ansiedade.

À Jéssica e Saynarah, por me darem tanto auxílio emocional e me fazerem companhia em quase todos os dias do meu último ano de graduação.

Ao Prof. Dr. Nicolas de Araújo Moreira, por ter me auxiliado na elaboração desse projeto, desde o fornecimento do tema até o desenvolvimento e às revisões sempre muito próximo.

Ao Prof. Dr. Arthur Plínio de Souza Braga, por ter aceitado desenvolver esse trabalho e me auxiliar no fim da minha formação.

Por último, o mais especial, ao Prof. Dr. Ernani de Sousa Ribeiro Júnior. Não sei nem por onde começar a descrever o quanto ele foi importante para eu me formar. Quando tudo parecia não ter direção, o curso parecia ir a lugar nenhum, e eu estava pensando em desistir de tudo, foi a pessoa que eu menos esperava (um professor) quem veio conversar comigo e contar sua história de vida e terminou com as seguintes palavras: “Eu sei do seu potencial, tenho certeza de que você terá muito sucesso na sua vida e um dia eu terei orgulho de dizer que tive o prazer de ter sido seu professor em uma disciplina”. Eu realmente não sei dizer o quanto as palavras que me disse me ajudaram a chegar até aqui. Eu quem tenho todo o orgulho do mundo de ter o tido como meu professor.

“Ninguém se forma sozinho. (Ditado Popular).”

“É só isso, não tem mais jeito, acabou, boa sorte. (Wanessa da Mata)”

“Tu te tornas eternamente responsável por aquilo que cativas (Antoine de Saint-Exupéry)”.

RESUMO

O presente trabalho objetiva desenvolver um protótipo de sistema de automação residencial para deficientes visuais baseado em reconhecimento de voz usando dispositivos móveis tais como celulares e tablets baseado em plataformas livres e de baixo custo como MIT AppInventor, Google Cloud API e comunicação Bluetooth com Arduino. O sistema apresenta um bom índice de acerto do reconhecimento dos comandos em diversos usuários com diversos timbres, a diversas distâncias, e em situações com ou sem ruído, apesar dos testes não terem indicado boa viabilidade para casos em que o dispositivo estava distante e havia ruído, mostrando-se uma solução viável e de fácil implementação e replicação. Ademais, para dar segurança ao usuário, os comandos são confirmados por sinais sonoros aos deficientes visuais.

Palavras-chave: *Automação Residencial, Reconhecimento de Voz, Acessibilidade.*

ABSTRACT

The present text aims to develop a prototype of home automation system for blind users based on voice recognition using artificial neural networks embedded on mobile devices such as mobile phones and tablets for example, simple, easy to use and based on free and low cost platform such as MIT AppInventor, Google Cloud API and Bluetooth communication with Arduino device. The system presents a good hit rate of commands recognition with different users with different voice tones and under different scenarios, such as different distances and quiet and noisy ambients, although not having great results in scenarios on which the device was far from the user and the ambient was noisy, proving itself as a viable and easy to implement and to replicate solution. In addition, with the intention of ensuring the user the command has been given, the commands are confirmed via sound signals.

Keywords: *Home Automation, Voice Recognition and Accessibility*.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

OMS	Organização Mundial de Saúde
PcDs	Pessoas com Deficiências
MIT	*Massachusetts Institute of Technology*
API	Interface de Programação de Aplicativos (*Application Programming Interface*)
ICMC USP	Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo
MLP	Perceptron Múltiplas Camadas (*Multi-layer Perceptron)*
WPAN	Rede Pessoal Sem Fio (*Wireless personal area network*)
FHSS	Espectro de Difusão em Frequência Variável (*Frequency-hopping spread spectrum*)
IDE	Ambiente de Desenvolvimento Integrado (*Integrated Development Environment*)
ICSP	Programação Serial em Circuito (*In Circuit Serial Programming*)
JSAPI	*Java Speech API*
LVCSR	Reconhecimento Contínuo de Voz com Vocabulário Amplo (*Large Vocabulary Continuous Speech Recognition*)
HMM	Modelo de Markov Oculto *(Hidden Markov Model)*.
FPGA	Arranjo de Portas Programáveis em Campo. *(Field Programmable Gate Array).*
SOM	*Self-Organizing Map*
UFOP	Universidade Federal de Outro Preto.

SUMÁRIO

1 INTRODUÇÃO

De acordo com o primeiro relatório mundial sobre visão publicado pela Organização Mundial de Saúde (OMS) mais de 1 bilhão de pessoas em todo o mundo que convive com algum grau de deficiência visual. O relatório, lançado às vésperas do Dia Mundial da Visão, celebrado em 10 de outubro, constatou que o envelhecimento da população, a mudança de estilo de vida e o acesso limitado à assistência oftalmológica, principalmente em países de baixa e média renda, estão entre os principais fatores do crescente número de pessoas que vivem com deficiência visual. (NAÇÕES UNIDAS DO BRASIL, 2019).

A Figura 1 mostra os resultados do relatório da OMS quanto ao número total de pessoas com deficiência visual, e o número de pessoas que estão com deficiência visual por conta de situação que podem ser prevenidas ou que ainda podem ser resolvidas, além da separação por tipo de doença do segundo grupo.

Figura 1: Resumo relatório OMS

Fonte: OMS (2019)

Um dos aspectos importantes na inclusão de um indivíduo portador de deficiência é a inclusão digital. O conceito de inserção digital envolve o acesso de todas as pessoas, deficientes ou não, ao mundo virtual. E, se o processo de inclusão do indivíduo portador de deficiência é complexo, o processo de sua inclusão no mundo digital apresenta uma série de dificuldades e obstáculos. (REINALDI; JÚNIOR; CALAZANS, 2011).

Pensando nisso, se faz necessário o desenvolvimento de sistemas e processos que facilitem a inclusão digital dos deficientes visuais, já que estes representam parte significativa da população mundial.

1.1 Motivação

Sabe-se que a ausência de adaptações apropriadas, em residências, prejudica a acessibilidade e o desempenho de pessoas com deficiências (PcDs). Estas pessoas, em seu cotidiano, deparam-se com diversas dificuldades de locomoção e acesso, o que limita ou mesmo inviabiliza sua independência e autonomia. Assim, é necessário que estas pessoas utilizem materiais, equipamentos adaptados, adequação do mobiliário e estrutura arquitetônica, ou seja, recursos que lhes propiciem condições seguras de mobilidade e conforto. (PAULUS; PILOTI; ANTONIAZZI; ANTUNES, 2013).

Tendo isso em mente, veio a ideia de juntar as necessidades de inclusão e integração tecnológica dos deficientes visuais com o conforto, cada vez mais exigido nos dias de hoje, por meio de um protótipo de automação residencial acionado por voz. Além disso, pensou-se em procurar um meio de que esse protótipo fosse realizado com o mais baixo custo possível, permitindo a disponibilidade para uma maior parte da população que precisa de cuidados.

1.2 Objetivos

O objetivo geral deste trabalho é criar um protótipo simples e fácil de ser manuseado para por comando de voz auxiliar um deficiente visual a realizar o controle de algumas ações domésticas, utilizando aplicativo criado pelo Massachusetts Institute of Technology (*MIT) AppInventor*, com reconhecimento de voz via *Google Cloud API* (*Application Programming Interface* - Interface de Programação de Aplicativos), e comunicação *Bluetooth* por meio de um *Arduino*.

1.2.1 *Objetivos específicos*

Entre os objetivos específicos deste trabalho estão:

- Estudar os meios de reconhecimento de voz já existentes no mercado;
- Criar um aplicativo capaz de reconhecer a voz e por meio dela criar uma *string* que corresponda ao que foi falado;
- Estabelecer uma comunicação *Bluetooth* entre o celular e o *Arduino*;

– Ativar ou desativar um comando conforme a *string* recebida;

– Dar um retorno sonoro ao usuário se a tarefa foi ou não executada.

1.3 Estrutura do Trabalho

Este trabalho está estruturado com os seguintes capítulos.

O Capítulo 1 apresenta uma ideia geral do tema abordado, a motivação, os objetivos traçados e a estrutura no qual o trabalho foi organizado.

O Capítulo 2 mostra um breve apanhado de trabalhos relacionados a reconhecimento de voz, e, mais especificamente, aos projetos de reconhecimento para deficientes visuais, buscando situar o leitor diante das soluções existentes no mercado e finalizando com a definição do que será procurado no Capítulo 3, ou seja, as ferramentas necessárias para o desenvolvimento do protótipo proposto no texto.

O Capítulo 3 traz uma revisão bibliográfica de conceitos envolvendo as ferramentas utilizadas no trabalho: *Google Cloud API, MITAppInventor, Bluetooth* e *Arduino.* Em detalhe, serão discutidos: a escolha da API para o projeto, com atenção aos princípios e equações utilizadas para realizar o reconhecimento de voz por cada tipo de API estudada, e com mais detalhes no da API escolhida; as propriedades do sistema desenvolvido no *MITAppInventor*, com as bibliotecas aplicadas para a criação do aplicativo; o sistema de comunicação *Bluetooth* em geral; e a plataforma *Arduino* com as bibliotecas usadas no projeto.

No Capítulo 4 é apresentado um modelo do projeto, seguindo as etapas, iniciando com o projeto do aplicativo, junto ao fluxograma do *AppInventor* com a explicação e o código em anexo; partindo para o do Arduino, também com as explicações e terminando mostrando como ficou o protótipo.

No Capítulo 5 são mostrados os resultados obtidos pelo projeto, por meio de demonstrações qualitativas, mostrando que a solução é viável e de fácil solução e implementação, sendo citadas as vantagens e desvantagens do sistema e comparando com outras soluções disponíveis no mercado. Por fim, serão apresentadas métricas quantitativas para mostrar o quão efetivo é o protótipo.

No Capítulo 6 são apresentadas as conclusões deste trabalho, bem como uma análise crítica dos resultados obtidos e trabalhos futuros.

2 TRABALHOS RELACIONADOS E SOLUÇÕES EXISTENTES

Neste capítulo é realizado um breve apanhado de trabalhos relacionados a reconhecimento de voz, e, mais especificamente, aos projetos de reconhecimento para deficientes visuais, buscando situar o leitor diante das soluções existentes no mercado.

Se iniciará o estudo exemplificando alguns projetos acadêmicos que trabalham o reconhecimento de voz junto à automação, passando por produtos existentes relacionados e terminando o capítulo com discussões sobre aspectos em que o presente trabalho se assemelha aos já existentes e no que ele se propõe a fazer.

2.1 *Trabalhos Acadêmicos Relacionados*

Durante o trabalho de (CORRÊA, 2013) se discorre sobre um sistema de automação e segurança residencial de baixo custo utilizando aplicativo *Android*, *Bluetooth* e reconhecimento de voz em conjunto com *Arduino*, mas não cita a problemática da acessibilidade e deficiência visual e nem realiza testes sobre o índice de acerto do sistema de reconhecimento de voz sob diversas condições de distância e ruído.

Dando continuidade, (SIMAS; RIBEIRO, 2001) utilizaram método de decisão multi-atributo difuso para reconhecer vogais da língua portuguesa.

A revisão de trabalhos também viu que em (CIPRIANO, 2001) é proposto a implementação de um sistema de reconhecimento automático de voz baseado em algoritmos otimizados, dando particular atenção aos Modelos de Markov Ocultos, embarcados em FPGA (*Field Programmable Gate Array* -Arranjo de Portas Programáveis em Campo).

Houve também o estudo apresentado em (MARANGONI; PRECIPITO, 2006) mostrando uma discussão sobre as vantagens, desvantagens e utilização do *Java Speech API*.

Durante a leitura de (BRAGA, 2006) é apresentada uma aplicação de uma rede neural artificial do tipo *Self-Organizing Map* (SOM), utilizado no reconhecimento de fala através do modelo de subdivisão fonética, além de técnicas de pré-processamento e extração de características do sinal de fala por meio do modelo cepstrum. O trabalho descreve desde as técnicas de pré-ênfase até os métodos de extração de coeficientes mel-cepstrais e energia. O trabalho realiza testes envolvendo reconhecimento de vogais, fonemas e frases, obtendo resultados razoáveis, 77%, e 68% para vogais e fonemas, em face da complexidade do problema

apresentado. Entretanto, para o reconhecimento de frases não atingiu o objetivo desejado. O trabalho sugere o uso de outros tipos de redes neurais e técnicas de pré-processamento e extração de características. Uma conclusão interessante é que certos fonemas se mostraram mais difíceis que outros, a saber: /p/, /b/ e /t/, por exemplo.

É discutido em (SANTOS, 2001) diversos aspectos de acessibilidade e inclusão digital, discutindo conceitos, fazendo um levantamento das ferramentas *Java* que proporcionam acessibilida às aplicações desktop, com particular enfoque na API *Java acessibility*, que contém classes e interfaces que ao serem aplicadas, garantem ao software se tornar acessível às tecnologias assistivas.

(Sun Microsystems, 2003) discorre as principais ferramentas de acessibilidade para a interface gráfica *Gnome* para *Linux*.

Em (BIDARRA; RODRIGUES, 2005) é mostrado um *software* livre amplificador de tela digital e inteligente voltado para usuários com baixa visão.

(MOREIRA, 2015) desenvolve um sistema livre e gratuito de inclusão digital para deficientes visuais baseado em um front-end multiplataforma desenvolvido em *Java* para o sintetizador de voz MBROLA, mas que também pode ser substituído facilmente por outras APIs e *softwares*, e um conjunto de programas acessíveis, que inclui editor de texto, cliente de *chat*, lente de aumento virtual, ente outros. Por ser livre, a comunidade pode modificar e melhorar o *software*.

Tendo visto os trabalhos acadêmicos relacionados, se pode iniciar um breve estudo dos produtos já comerciais que trabalham soluções similares.

2.2 *Produtos Existentes Relacionados*

2.2.1 *Amazon Echo*

A empresa *Amazon* fabrica possui três tipos de smart speakers com a assistente virtual *Alexa*: *Echo Dot* (R$ 349,00), *Echo Show* (R$ 599,00) - com conteúdo em imagem e som, incluindo uma câmera e uma tela de 5,5" com resolução de 960 por 480 *pixels*, e o *Echo* (R$ 699,00) - com qualidade de som superior e omnidirecional.

A assistente virtual *Alexa* permite que o usuário peça música nos principais serviços de streaming de música, pergunte sobre notícias e previsão meteorológica, controle dispositivos de casas inteligentes, como acendez luzes, trancar fechadura, veja câmeras de segurança e acione eletrodomésticos inteligentes como smart TVs, cafeteiras, geladeiras, microondas e demais dispositivos conectados. Tais dispositivos permitem também realizar chamadas de voz, interagir por meio de perguntas, acessar agenda de compromissios e até mesmo requisitar serviços de entrega a domicílio. (*Amazon, 2019*).

2.2.2 *App Casa*

O aplicativo Casa, desenvolvido pela *Apple*, permite o controle de todos os acessórios para *HomeKit*, por meio da assistente virtual *Siri*. Com ele, é possível mandar apagar as luzes, acessar câmeras de segurança por meio de dispositivos Apple, e controlar remotamente uma AppleTV.

Mais de 100 marcas estão desenvolvendo produtos compatíveis com *HomeKit*, como TVs, caixas de som, sistemas de iluminação, interruptores, tomadas, ventiladores, condicionadores de ar, umidificadores, purificadores de ar e termostatos, sensores, controles de portas, garagens, fechaduras, câmera e campainhas.

É possível solicitar ativação ou desativação de dispositivos para cada cômodo de forma individual e independente. O *App* Casa permite que um único comando controle os parâmetros de diversos dispositivos ao mesmo tempo, por exemplo, um comando "saindo de casa" pode ser configurado para desligar todas as luzes, condicionadores de ar, televisão e tranque as portas, por exemplo. (*Apple*, 2019).

2.2.3 *Google Home/Google Nest Mini*

O *Google Nest Mini* permite usar a tecnologia de reconhecimento de voz para acessar músicas e dispositivos de *Apps* por meio do Google Assistente.

Assim como no caso da *Siri*, da *Apple* ou da *Alexa*, é possível consultar a previsão do tempo ou ouvir notícias ou informações personalizadas, como programação, lembretes, acionar temporizadores, alarmes, diminuir luz, ligar TV ou alterar a temperatura do ambiente.

O *Google Nest Mini* pode ser pendurado na parede. Ele detecta presença do usuário e é feito a partir de materiais sustentáveis, como garrafas plásticas recicláveis. No site da

Empresa é possível ver quais *apps* e produtos são compatíveis. É possível encontrar o Google Nest Mini em diversas lojas de varejo no Brasil por R$ 349,00. (*Google Store,* 2019).

Tendo visto os trabalhos acadêmicos e os produtos existentes relacionados ao trabalho, se pode realizar uma comparação destes ao projeto proposto nesse texto.

2.3 *Projeto proposto e os Trabalhos Acadêmicos e Produtos já Existentes*

2.3.1 *Comparações com os Trabalhos Acadêmicos.*

O presente trabalho se aprofunda no algoritmo de redes neurais artificiais utilizado pelo Google Cloud API, mas também efetua comparação com Cadeias de Markov Ocultas, utilizada pelo Framework Julius. É feita também uma breve discussão sobre o Java Speech API. Se mostrando como um trabalho que apresenta uma revisão de diversas ferramentas existentes para o reconhecimento de voz, além das técnicas utilizadas por estas.

Além disso, esta monografia atualiza a discussão sobre os produtos existentes para automação residencial, com novos produtos e funcionalidades que surgiram recentemente.

O trabalho traz, também, a problemática da acessibilidade e deficiência visual ao ramo da automação residencial, com a realização de testes sobre o índice de acerto de um sistema de reconhecimento de voz sob diversas condições de distância e ruído. Buscando, ainda, novos dados sobre o número de deficientes visuais no mundo de acordo com relatório recente da OMS.

2.3.2 *Comparações com os Produtos Existentes*

Os produtos existentes no mercado, são, como visto, soluções de custo elevado, com muitas funcionalidades, proporcionando conforto, mas que não se adequam a realidade financeira de muitos.

O protótipo sugerido por este trabalho tem como objetivo ser um teste simples de conceito, facilmente adaptável para adquirir mais funcionalidades, que mostre que é possível, sim, trazer acessibilidade aos deficientes visuais por meio de um sistema de baixo custo e que apresente uma boa eficiência dentro de condições adequadas.

Tendo realizado as comparações com os produtos e trabalhos existentes, pode-se iniciar os estudos acerca da fundamentação dos conceitos e aspectos que envolvem o protótipo.

3 FUNDAMENTAÇÃO E REVISÃO BIBLIOGRÁFICA

Neste capítulo serão abordados os conceitos e aspectos que envolvem a criação do protótipo, passando pelo método de reconhecimento de voz, a forma de comunicação utilizada e o microprocessador selecionado para processar as informações e realizar os comandos.

Se iniciará o estudo pela análise de APIs de reconhecimento de voz existentes.

3.1 *Plataformas de Desenvolvimento*

Existem diversas plataformas para realizar reconhecimento de voz, nessa seção serão tratados de algumas APIs existentes, mais especificamente sobre *Julius Framework*, *Java Speech API* (JSAPI) e *Google Cloud Speech API*, além de explicar os métodos de reconhecimento utilizados por cada um deles.

3.1.1 *Julius Framework*

Toda a explicação sobre as principais características do *Julius* a seguir teve como base (LEE, 2010).

Julius é um software decodificador LVCSR (*Large Vocabulary Continuous Speech Recognition* - Reconhecimento Contínuo de Voz com Vocabulário Amplo) de código aberto baseado em HMM (*Hidden Markov Model* - Modelo de Markov Oculto) para trabalho com processamento de fala, capaz de realizar processamento rápido (quase tempo real) e exigindo pouca memória.

Os modelos acústicos e de linguagem podem criar diversos tipos de sistemas de reconhecimento de voz, possibilitando modelos personalizados e módulos para tarefas específicas.

Para realizar o reconhecimento, devem ser selecionadas a língua e as tarefas desejadas, definindo o modelo.

Para essas seleções, deve-se modificar ou alterar os seguintes parâmetros: dicionário, definindo as palavras a serem reconhecidas, assim como as pronúncias definidas, a partir de uma sequência de fonemas; modelo de língua, selecionando as regras de sintaxe, determinando as conexões entre as palavras, e definindo os padrões das sentenças, esse modelo pode ser escrito por meio de regras ou por meio de um modelo de gramática probabilístico; e o

modelo acústico, que é o modelo estocástico para padrões de entrada de formas de onda. Por padrão, o modelo acústico adotado é o HMM.

Desse modo, a gramática representa a sintaxe possível, junto a ligação de palavras e categorias, ou padrões de palavras para tarefas específicas.

Quando ocorre uma entrada de fala o Julius busca pela sequência de palavras mais similares existentes na gramática. A gramática é descrita por dois arquivos diferentes, um para a gramática propriamente dita e outro para as palavras candidatas.

Com isso, para finalizar o entendimento de como é realizado o reconhecimento de voz pelo Julius, será discutido um pequeno resumo sobre o HMM.

3.1.1.1 Modelo de Markov Oculto

Toda a explicação sobre os modelos HMM é baseada em (HOWARD; RORRES, 2001).

Suponha que um processo físico ou matemático esteja mudando de maneira na qual cada momento possa ocupar um número de estados finito. Esse sistema está mudando de um estado para outro de tempo em tempo e em alguns tempos pré-determinados esse sistema é observado. Se o estado desse sistema não pode ser previsto com certeza, mas a probabilidade de um certo estado pode ser prevista de acordo com somente o seu estado anterior, o processo de mudança de estados é chamado de Processo de Markov, ou Cadeia de Markov.

Sendo os k estados possíveis de uma cadeia de Markov denotados como $1,2,...,k$, a probabilidade do sistema estar em um estado i durante qualquer observação se o último estado for j denotada por p_{ij} (probabilidade da transição do estado j para o estado i). A matriz $P=[p_{ij}]$ é chamada de matriz de transição da cadeia de *Markov*.

Se P é a matriz de transição de uma cadeia de Markov com k estados, então, para cada j se tem:

$$p_{ij}+p_{2j}+...+p_{kj}=1 \quad (1),$$

uma vez que o sistema esteja no estado j durante uma observação, é certo que essa vá assumir k possíveis estados. Uma matriz com essa propriedade é chamada de matriz estocástica, matriz

de probabilidade ou matriz de Markov. Como é possível observar, uma matriz de transição é sempre uma matriz estocástica.

O estado de um vetor em uma observação de uma Cadeia de Markov com k estados é um vetor coluna **x** cuja componente x_i é a probabilidade de o sistema assumir esse estado durante a observação atual. Todas as entradas desse vetor de estados são não negativas, e tem sua soma igual a 1. Um vetor coluna com essa propriedade é chamado de vetor probabilidade.

Se P é a matriz de transição e $\boldsymbol{X^n}$ é o vetor de estados durante a n-ésima observação, então:

$$\boldsymbol{X}^{n+1}=P\boldsymbol{X}^n \quad (2).$$

A matriz de transição é regular se uma potência positiva dessa matriz tem todas as entradas positivas. Em outras palavras, para uma matriz de transição regular P, existe um inteiro m tal que todas as entradas de P^m são positivas. Uma Cadeia de Markov que segue uma matriz de transição é chamada Cadeia de Markov Regular.

Se P é uma matriz de transição regular, então:

$$P \rightarrow \begin{bmatrix} q_1 & q_1 & \cdots & q_1 \\ q_2 & q_2 & \cdots & q_2 \\ \vdots & \vdots & \ddots & \vdots \\ q_k & q_k & \cdots & q_k \end{bmatrix} (3),$$

quando $k \rightarrow \infty$, onde q_i são números positivos, os quais:

$$\sum_{i=1}^{k} q_i = 1 (4).$$

Denotando:

$$Q = \begin{bmatrix} q_1 & q_1 & \cdots & q_1 \\ q_2 & q_2 & \cdots & q_2 \\ \vdots & \vdots & \ddots & \vdots \\ q_k & q_k & \cdots & q_k \end{bmatrix} \text{ e } q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{bmatrix} (5)$$

então, Q é uma matriz de transição em que todas as colunas são iguais ao vetor probabilidade **q**. Essa matriz possui a propriedade de que se x é qualquer vetor probabilidade, então:

$$Qx = \begin{bmatrix} q_1 & q_1 & \cdots & q_1 \\ q_2 & q_2 & \cdots & q_2 \\ \vdots & \vdots & \ddots & \vdots \\ q_k & q_k & \cdots & q_k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} q_1x_1 & q_1x_2 & \cdots & q_1x_k \\ q_2x_1 & q_2x_2 & \cdots & q_2x_k \\ \vdots & \vdots & \ddots & \vdots \\ q_kx_1 & q_kx_2 & \cdots & q_kx_k \end{bmatrix} = (x_1 + x_2 + \ldots + x_k) \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{bmatrix} = q \quad (6)$$

Isso mostra que Q transforma qualquer vetor de probabilidade **x** em um vetor de probabilidade fixo **q**.

Se P é uma matriz de transição regular e $\boldsymbol{x}$ é o vetor probabilidade, então:

$$P^n\mathbf{x} \rightarrow \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{bmatrix} = \mathbf{q} \quad (7)$$

quando $k \rightarrow \infty$. Em que $\boldsymbol{q}$ é o vetor de probabilidade fixo, independente de **n**, com todas as entradas positivas.

O vetor **q** é chamado de vetor de estado estacionário de uma Cadeia de Markov regular. Isso acontece tal que **q** é o único vetor que satisfaz $P\boldsymbol{q}=\boldsymbol{q}$.

O Modelo de Markov Oculto é um modelo estatístico de Markov no qual o sistema modelado é assumido ser um processo de Markov com estados escondidos. É utilizado em diversas aplicações de reconhecimento de padrões e pode ser considerado como uma generalização de um modelo em que as variáveis escondidas que controlam os componentes a serem selecionados a cada observação são relacionados por meio de um processo de Markov, em vez de serem independentes umas das outras.

3.1.2 *Java Speech API*

Toda a explicação sobre as principais características do *Julius* teve como base (Sun Microsystems, 1998).

O *Java Speech API* é uma interface de *software* padrão, multiplataforma e de utilização simples.

Ela suporta duas tecnologias: o reconhecimento e a síntese de fala. O reconhecimento de fala representa aos computadores a habilidade de "ouvir" a linguagem falada e informar o que foi dito pelo usuário, podendo processar essa informação de diversas formas. Mais especificamente, processa uma entrada de áudio e a converte para texto. A síntese trata do processo oposto, ou seja, gera uma saída em linguagem falada por meio de uma entrada de texto escrito (*Text-to-Speech*).

O passo a passo do reconhecedor de voz são: o design da gramática, definindo as palavras que podem ser faladas pelo usuário e os padrões os quais estas podem ser faladas; o processamento do sinal, analisando as características de frequência da entrada de áudio; o reconhecimento dos fonemas, comparando os padrões da frequência de entrada com os da linguagem sendo reconhecida; reconhecimento das palavras, com a comparação da sequência de fonemas possíveis contra as palavras e padrões de palavras especificadas pela gramática; por fim, a geração dos resultados.

Não se é especificado qual o modelo matemático utilizado para essa API.

3.1.3 *Google Cloud Speech API*

Com o *Cloud Speech-to-Text*, os desenvolvedores convertem áudio em texto ao aplicar modelos de rede neural em uma API fácil de usar. A API reconhece 120 idiomas e variantes, oferecendo suporte a usuários de diversos países. Além disso, essa API processa *streaming* em tempo real ou áudio pré-gravado usando a tecnologia de aprendizagem de máquina do Google. (Google, 2019).

Outros recursos da *Cloud Speech-to-Text* são: Reconhecimento de fala personalizado (pode especificar palavras específicas ao seu trabalho); suporte a áudio pré-gravado ou *streaming* em tempo real; detecção de idiomas automática em versão beta; robustez de ruído (lida com áudio barulhento de muitos ambientes sem exigir cancelamento de ruído a mais); filtragem de conteúdo inadequado; pontuação automática em versão beta; seleção de modelo (quatro modelos prontos: padrão, pesquisa e comandos de voz, chamadas telefônicas e transcrição de vídeo); diarização de locutor (saber quem está falando em versão beta); e reconhecimento de diversos canais. (Google, 2019).

A Figura 2 exemplifica o fluxo de trabalho da API, as setas vermelhas representam o caminho lógico de ida da API e as setas azuis o caminho lógico de volta. Inicialmente, um arquivo de áudio é inserido, após isso, é ativado um evento que chama uma função da *Google Cloud*, essa função lê o arquivo, constrói um arquivo correspondente a ele e envia um chamado para a API, referenciando o arquivo de áudio armazenado. Esse arquivo de áudio é comparado com a base de áudios armazenados na *Google Cloud* para decidir qual texto será escrito. Uma vez que foi finalizado o processamento, é retornado um arquivo de texto à função da nuvem. Essa função, então, escreve o arquivo de texto desejado. (Medium Corporation, 2017).

Figura 2: Esquemático do Funcionamento do Google Cloud Speech API

Fonte: Medium Corporation (2017)

Conforme visto anteriormente, o *Cloud Speech-to-Text* aplica os algoritmos de aprendizado de redes a arquivos de áudio para um reconhecimento de fala preciso. A precisão da *Cloud Speech-to-Text* melhora com o tempo, à medida que o Google aperfeiçoa a tecnologia de reconhecimento de fala interna usada pelos próprios produtos.

Para melhor entendimento, serão discutidos os modelos mais conhecidos de redes neurais na literatura.

3.1.3.1 Introdução às Redes Neurais Artificiais

Para iniciar a explicação de o que é uma rede neural, se precisa entender o contexto no qual foram realizados os primeiros estudos e explicações simples sobre as representações de uma rede neural, além de quais os modelos que serão trabalhados.

O modelo de estudo de redes neurais surgiu a partir da necessidade de responder, principalmente, à três questionamentos: “Como a informação é sentida ou detectada pelo sistema biológico?”; “De qual forma a informação é armazenada ou lembrada?”; e “Como a informação contida no armazenamento, ou na memória, influencia no reconhecimento e no comportamento?” (ROSENBLATT, 1958).

O reconhecimento de que o cérebro humano tem uma forma peculiar ao computador digital convencional motivou o trabalho em Redes Neurais Artificiais. O cérebro é um computador altamente complexo, não linear e paralelo. (HAYKIN, 2001).

Um cérebro tem uma grande estrutura e a habilidade de desenvolver suas próprias regras através de experiências. A experiência vai sendo acumulada com o tempo, com desenvolvimento mais intenso do cérebro humano acontece durante os dois primeiros anos de vida, apesar de o desenvolvimento continuar para muito além desse estágio. (HAYKIN, 2001).

Um neurônio em “desenvolvimento” é sinônimo de um cérebro plástico; isso significa que o sistema nervoso em desenvolvimento é capaz de se adaptar ao seu meio ambiente. (HAYKIN, 2001).

Com isso, temos uma resposta básica às três perguntas realizadas inicialmente, podendo, então, partir para uma definição simples de o que é uma rede neural.

Uma rede neural é um processador constituído de unidades de processamento simples, que têm a capacidade de armazenar conhecimento experimental e torná-lo disponível para o uso. Ela se assemelha ao cérebro em dois aspectos: O conhecimento é adquirido a partir de seu ambiente; e as forças de conexão entre neurôniossão utilizadas para armazenar esse conhecimento. (HAYKIN, 2001).

Redes Neurais Artificiais são técnicas computacionais que apresentam um modelo matemático inspirado no cérebro humano e que adquirem conhecimento por meio da

experiência. Uma rede neural artificial pode ter centenas ou milhares de unidades de processamento.

A propriedade mais importante das redes neurais é aprender com o seu ambiente e, com isso, melhorar seu desempenho. Isso é feito por meio de um processo iterativo de ajustes aplicado a seus pesos, o treinamento. O aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas. (Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC USP), 2019).

A Figura 3 mostra como o cérebro humano pode ser visto tendo em mente um sistema de três estágios. A rede neural (nervosa) representa o cérebro, centro do sistema, que recebe continuamente informação, percebe-a e toma decisões apropriadas. Os receptores convertem estímulos do corpo humano ou do ambiente externo em sinais que transmitem informações para a rede neural (cérebro). Os atuadores convertem os sinais gerados pela rede neural em respostas (saídas do sistema). (HAYKIN, 2001).

Figura 3: Representação em diagrama de blocos do sistema nervoso.

Fonte: HAIKYN, 2001

Redes neurais oferecem o potencial de prover paralelismo massivo, adaptação e novas soluções algorítmicas para problemas de reconhecimento de voz. Estudos iniciais mostram que redes de múltiplas camadas (MLP - *Multi-layer Perceptron*) com atrasos podem gerar discriminações excelentes entre pequenos segmentos de palavras, consoantes e vogais difíceis de serem discriminadas. A performance para esses casos de vocábulos pequenos excederem várias vezes os métodos mais convencionais. (LIPPMANN, 2008).

Redes neurais podem ser poderosos classificadores de sinais de voz. Um pequeno conjunto de palavras pode ser reconhecido mesmo com algoritmos com poucas camadas e simplificados. (GEVAERT; TSENOV; MLADENOV, 2010)

Tendo definido de onde veio o estudo das redes neurais, suas aplicações no reconhecimento de voz, e no que elas são baseadas, faz-se necessário explicar a modelagem matemática e computacional.

É importante que se entenda, inicialmente, o Perceptron Simples, para se ter um entendimento básico do equacionamento de um neurônio, para que depois possa ser entendido o modelo MLP com *backpropagation* de maneira mais rápida.

3.1.3.2 Perceptron Simples

A operação de uma unidade de processamento, proposta por McCullock e Pitts em 1943, pode ser resumida da seguinte maneira:

• Sinais são apresentados à entrada;

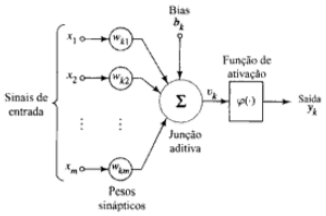
• Cada sinal é multiplicado por um número, ou peso, que indica a sua influência na saída da unidade;

• É feita a soma ponderada dos sinais que produz um nível de atividade;

• Se este nível de atividade exceder um certo limite (*threshold*) a unidade produz uma determinada resposta de saída. (MCCULLOCK; PITTS, 1943)

Esse resumo pode ser visto de forma esquematizada na Figura 4, na qual X = (x_1,x_2,..,x_m) representa o vetor com os sinais de entrada; W = (w_{k1},w_{k2},...,w_{km}) representa o vetor com os pesos associados para cada elemento do vetor de entrada; o bloco de somatório (combinador linear) representa a soma da multiplicação de cada elemento do vetor com seu respectivo peso (combinação linear entre X e W mais o elemento de *bias* (uma constante que ajuda o modelo a se adequar aos dados recebidos, deslocando a fronteira de decisão) b_k); a função $\varphi(\cdot)$ representa uma equação que determina qual a resposta y_k correspondente ao vetor de entrada.

Figura 4: Esquema de unidade McCullock-Pitts.



Fonte: HAYKIN, 2001

Vale lembrar que *k* representa o neurônio em questão e *m* o terminal de entrada da sinapse à qual o peso se refere.

Dessa forma, pode-se iniciar o equacionamento do modelo de um neurônio utilizando as notações anteriormente explicadas. Todo o equacionamento foi baseado em (HAYKIN, 2001).

Em termos matemáticos, podemos descrever um neurônio *k* escrevendo o seguinte par de equações:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \quad (8)$$

e

$$y_k = \varphi(u_k + b_k) \quad (9),$$

onde u_k representa a saída do combinador linear.

Outro detalhe importante está na multiplicação entre os vetores *w* e *x*. Seguindo (BOLDRINI, 1986): Só se pode efetuar o produto de duas matrizes A_{mxn} e B_{lxp} se o número de colunas da primeira for igual ao número de linhas da segunda, isto é, n = l.

Tendo isso em mente, para realizar a multiplicação entre *w* e *x*, é necessário transpor a matriz *w*, para que a dimensão seja adequada à multiplicação. Desse modo a equação 8 fica sendo:

$$u_k = w_j^T x \ (10)$$

As equações 8, 9 e 10 representam, em geral, a saída de um neurônio em uma rede, a pergunta a ser respondida agora é: Como que a partir delas ocorre o aprendizado da rede?

Aprendizagem é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre. (MENDEL; MCCLAREN, 1970).

Com isso, se entende o equacionamento de um neurônio, e se vê a necessidade de estabelecer equações que prevejam o aprendizado da rede. Dessa forma, pode ser dado o estudo da rede MLP e do algoritmo de aprendizado *backpropagation*.

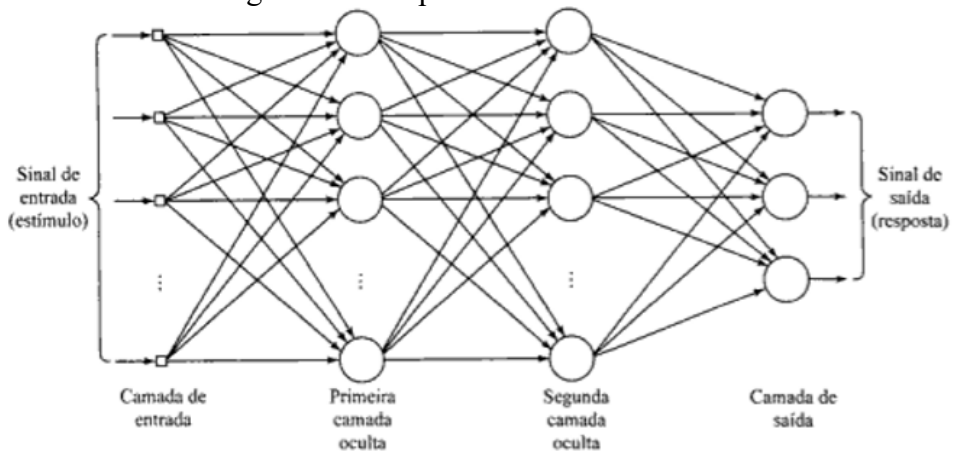
3.1.3.3 Perceptron Múltiplas Camadas

A principal motivação para se utilizar redes de múltiplas camadas vêm de que os problemas linearmente separáveis representam uma classe muito restrita, e que a maioria dos sistemas encontrados teriam resultados menores que a performance ótima quando utilizando um classificador linear como o Perceptron Simples. (BISHOP, 1995).

Tipicamente, a rede MLP consiste de um conjunto de unidades sensoriais (nós de fonte) que constituem a camada de entrada, uma ou mais camadas ocultas de nós computacionais e uma camada de saída de nós computacionais. O sinal de entrada se propaga para frente através da rede, camada por camada. (HAYKIN, 2001).

A Figura 5 representa a arquitetura de um perceptron de múltiplas camadas com duas camadas ocultas e uma camada de saída. Um detalhe importante é o de que a rede é totalmente conectada, ou seja, um neurônio em qualquer camada da rede está conectado a todos os nós/neurônios da camada anterior. O fluxo de sinal da rede progride para frente, da esquerda para a direita e de camada em camada. (HAYKIN, 2001).

Figura 5: Exemplo de uma rede MLP.



Fonte: HAYKIN, 2001

As redes MLP têm sido aplicadas com sucesso para resolver diversos problemas difíceis, através do seu treinamento de forma supervisionada com um algoritmo muito popular conhecido como algoritimo de retropropagação de erro (*error back-propagation*). Este algoritmo é baseado na regra de aprendizagem por correção de erro. (HAYKIN, 2001).

Basicamente, a aprendizagem por retropropagação de erro consiste de dois passos através das diferentes camadas da rede: um passo para frente, a propagação, e um passo para trás, a retropropagação.

Na propagação, o vetor de entrada é aplicado aos nós sensoriais da rede e seu efeito se propaga através da rede, camada por camada, produzindo um sinal de saída. Nessa fase, os pesos sinápticos da rede são todos fixos.

Na retropropagação, a resposta real da rede é subtraída de uma resposta desejada para produzir um sinal de erro. Esse sinal de erro é propagado para trás através da rede, contra a direção das conexões sinápticas. Os pesos sinápticos são ajustados para fazer que a resposta real da rede se mova para mais perto da resposta desejada, em um sentido estatístico.

Em resumo, na retropropagação, os pesos sinápticos são todos ajustados de acordo com uma regra de correção de erro. (HAYKIN, 2001).

Para finalizar o primeiro contato com as redes MLP, pode-se resumir 3 características importantes à essas redes: o modelo de cada neurônio inclui uma função de ativação não linear, diferenciável em qualquer ponto, utilizada para que a relação de entrada-saída da rede não seja reduzida àquela de um perceptron simples; a rede contém uma ou mais camadas de neurônios ocultos, que não são parte da entrada ou da saída da rede. Estes neurônios capacitam a rede a aprender tarefas complexas extraindo progressivamente as características mais significativas dos vetores de entrada; a rede exibe um alto grau de conectividade, determinado pelas sinapses da rede. (HAYKIN, 2001).

O equacionamento da propagação e retropropagação será realizado tendo como base (BARRETO, 2017).

A etapa de propagação envolve o cálculo das ativações e saídas de todos os neurônios da camada escondida e de todos os neurônios da camada de saída. Desse modo, após a apresentação de um vetor de entrada **x**, na iteração *t*, o primeiro passo é calcular as ativações dos neurônios da camada escondida:

$$u_i(t) = \sum_{j=0}^{p} w_{ij}(t)x_j(t) = \boldsymbol{w}_i^T(t)\boldsymbol{x}(t),\ i=1,\ldots,q \quad (11)$$

Em que T indica o vetor (ou matriz) transposto e q indica o número de neurônios da camada escondida. Em seguida, as saídas correspondentes são calculadas como:

$$z_i(t) = \varphi\big(u_i(t)\big) = \varphi_i\big(\sum_{j=0}^{p} w_{ij}(t)x_j(t)\big) = \varphi_i(\boldsymbol{w}_i^T(t)\boldsymbol{x}(t)), \quad (12)$$

Tal que a função de ativação $\varphi(\cdot)$ assume geralmente uma das seguintes formas:

$$\varphi\big(u_i(t)\big) = \frac{1}{1 + \exp\left[-u_i(t)\right]}, (Logística) \quad (13)$$

$$\varphi\big(u_i(t)\big) = \frac{1 - \exp\left[-u_i(t)\right]}{1 + \exp\left[-u_i(t)\right]}, (Tangente\ Hiperbólica) \quad (14)$$

A Equação 13 é utilizada em casos onde se deseja as saídas apenas como números positivos, e a Equação 14 em aplicações cujas saídas devem ser entre -1 e 1. (UFOP – Universidade Federal de Ouro Preto, 2016).

O segundo passo consiste em repetir as operações das equações 11 e 12 para os neurônios da camada de saída:

$$u_k(t) = \sum_{i=0}^{q} m_{ki}(t)z_i(t), \text{ k=}1,\ldots,M \quad (15)$$

Em que M é o número de neurônios de saída e $\boldsymbol{m}_{ki}(t)$ é o vetor de pesos associados a cada neurônio k da camada de saída. Note que as saídas dos neurônios da camada escondida, $\boldsymbol{z_i}(t)$, fazem o papel de entrada para os neurônios da camada de saída.

Em seguida, as saídas dos neurônios da camada de saída são calculadas como:

$$\boldsymbol{y}_k(t) = \varphi\big(u_k(t)\big) = \varphi_k\left(\sum_{i=0}^{q} \boldsymbol{m}_{ki}(t)\boldsymbol{z}_i(t)\right) \quad (16)$$

Tal que a função de ativação φ_k assume geralmente uma das formas definidas nas Equações 13 e 14.

Nisso, finaliza-se o processo de propagação e se inicia o estudo da retropropagação.

Após os cálculos das ativações e saídas na propagação, o primeiro passo consiste em calcular os gradientes locais δ_k dos neurônios da camada de saída:

$$\delta_k(t) = e_k(t)\varphi'\big(u_k(t)\big), k = 1, \ldots, M \quad (17)$$

Em que $e_k(t)$ é o erro entre a saída desejada $d_k(t)$ para o neurônio k e a saída gerada por ele, $y_k(t)$:

$$e_k(t) = d_k(t) - y_k(t), k = 1, \dots, M \quad (18)$$

A derivada $\varphi'\big(u_k(t)\big)$ assume diferentes formas, dependendo da escolha da função de ativação. Assim temos as seguintes possibilidades:

$$\varphi'\big(u_k(t)\big) = \frac{d\varphi(u_k(t))}{du_k(t)} = y_k(t)[1 - y_k(t)], \text{ se } \varphi\big(u_k(t)\big) \text{ é a função logística. (19)}$$

$$\varphi'\big(u_k(t)\big) = \frac{d\varphi(u_k(t))}{du_k(t)} = 1 - y_k^2(t), \text{ se } \varphi\big(u_k(t)\big) \text{ é a tangente hiperbólica. (20)}$$

O próximo passo consiste em calcular os gradiantes locais dos neurônios da camada escondida:

$$\delta_i(t) = \varphi'\big(u_i(t)\big) \sum_{k=1}^{n} m_{ki}(t)\delta_k(t)\,, i = 1, \dots, q (21)$$

Tal que a derivada $\varphi'\big(u_i(t)\big)$ pode ser calculada por:

$$\varphi'\big(u_i(t)\big) = \frac{d\varphi(u_i(t))}{du_i(t)} = y_i(t)[1 - y_i(t)], \text{ se } \varphi\big(u_i(t)\big) \text{ é a função logística. (22)}$$

$$\varphi'\big(u_i(t)\big) = \frac{d\varphi(u_i(t))}{du_i(t)} = \frac{1}{2}[1 - y_i^2(t)], \text{ se } \varphi\big(u_i(t)\big) \text{ é a tangente hiperbólica. (23)}$$

Agora é preciso realizar o processo de atualização ou ajuste dos parâmetros (pesos sinápticos e limiares) da rede MLP com uma camada escondida. Assim, para a camada escondida tem-se que a regra de atualização dos pesos, w_{ij}, é dada por:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

$$= w_{ij}(t) + \alpha\delta_i(t)x_j(t) \quad (24)$$

Em que α é a taxa de aprendizagem. E para cada camada de saída a regra de atualização dos pesos, m_{ki}, é dada por:

$$m_{ki}(t+1) = m_{ki}(t) + \Delta m_{ki}(t)$$

$$= m_{ki}(t) + \alpha\delta_k(t)z_i(t) \quad (25)$$

Com isso, foi discutido os métodos de aprendizagem das redes, as arquiteturas das redes e os princípios básicos por trás das redes neurais. Desse modo, se pode ter uma boa noção de como o *Google Cloud Speech-to-Text* reconhece o que o usuário fala.

3.1.4 *Escolha da API*

Buscando uma API que fosse simples de utilizar e que reconhecesse facilmente a língua nativa (português), mas que pudesse ser facilmente adaptado para outras línguas e outras situações, se decidiu por utilizar o *Cloud Speech-to-Text*, por ser a opção que juntava a comodidade de se trabalhar com um sistema mundialmente conhecido e com a flexibilidade para operar em diversas funções.

3.2 *MIT AppInventor*

O *MIT AppInventor* é um ambiente de programação didático lúdico-visual que permite criar aplicativos totalmente funcionais para *smartphones* e *tablets*. Usuários novatos conseguem desenvolver um aplicativo simples com facilidade. A ferramenta baseada em blocos facilita a criação de aplicativos complexos em tempo significamente menor que outros ambientes tradicionais de programação. (MIT, 2019).

Diante disso, procurando desenvolver uma interface simples de programar e que fosse amigável ao usuário, se selecionou o *MIT AppInventor*.

Toda a seção de explicação da ferramenta e dos blocos utilizados no trabalho será dada com base no que está disponibilizado de tutoriais e apresentações no site do *MIT AppInventor*.

Há duas etapas principais para a elaboração de um aplicativo no *AppInventor*, o *AppInventor Designer* e o *AppInventor Blocks Editor*. Essas etapas serão explicadas a seguir:

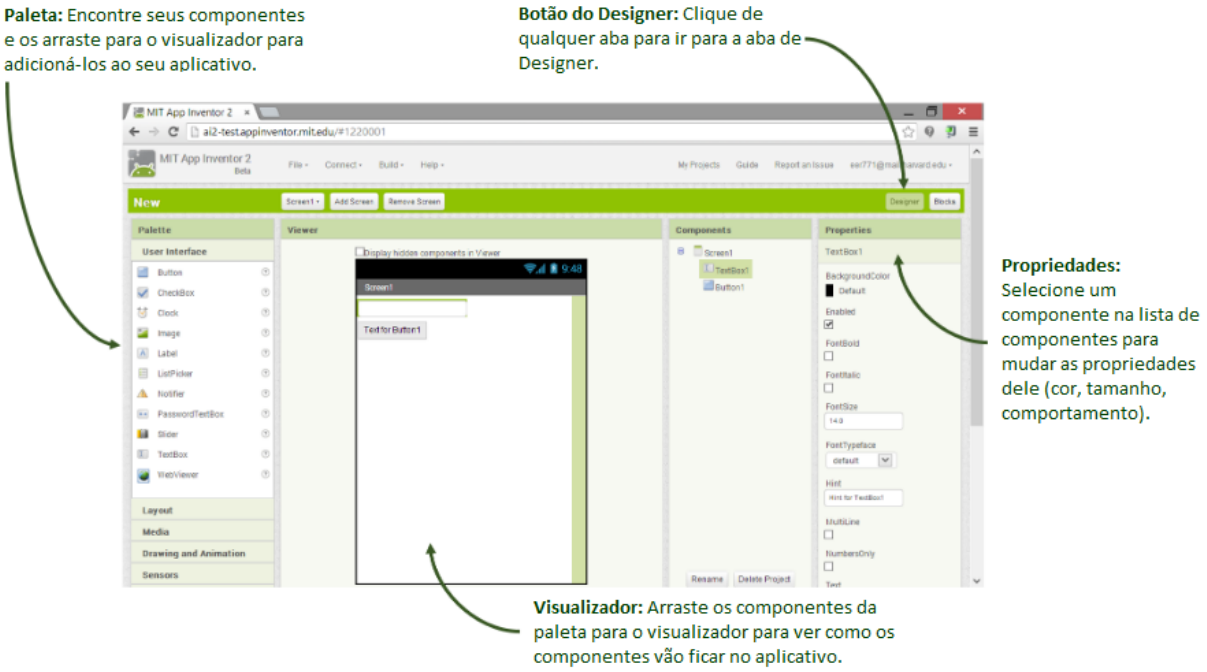
3.2.1 *App Inventor Designer*

É o ambiente que contém os componentes que podem ser colocados no aplicativo.

Esse ambiente permite desenhar a interface colocando componentes que ficarão na tela ou que funcionarão sem que o usuário os veja diretamente.

A Figura 6 mostra como é o ambiente ao inicializar um novo projeto no *App Inventor*, além de conter um breve resumos das funcionalidades. Neste texto será explicado sobre os elementos básicos e mais especificamente sobre os elementos utilizados no projeto.

Figura 6: Ambiente novo de criação de aplicativo no App Inventor



Fonte: Próprio Autor.

A ferramenta mais importante e a que será o foco do texto é a paleta com os elementos, representada na Figura 7. Mais especificamente, serão tratados dos elementos nas categorias “Interface de Usuário”, “Organização”, “Mídia” e “Conectividade”.

Figura 7: Paleta App Inventor

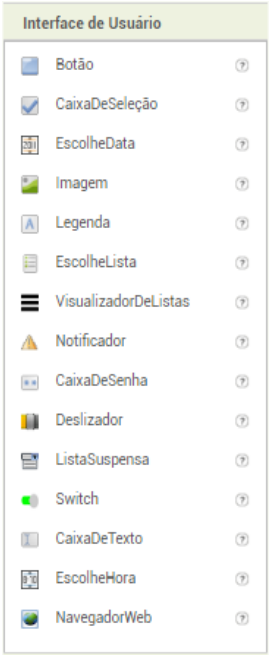
Fonte: Próprio Autor.

Além da paleta, vale destacar a parte de propriedades, na qual é possível selecionar componentes da lista de componentes para alterar suas propriedades (cor, tamanho,

comportamento, entre outras) e o visualizador no qual se colocam os componentes da paleta para ver como irá ficar o aplicativo.

A Figura 8 mostra os elementos dentro da categoria de “Interface de Usuário”, destes será discorrido sobre os elementos “Botão”, “Escolhe Lista” e “Legenda”.

Figura 8: Elementos dentro de “Interface de Usuário”



Fonte: Próprio Autor.

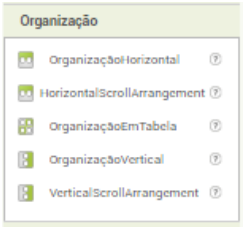
“Botão” é um botão com a habilidade de detectar cliques. Muitos aspectos da aparência dele podem ser mudados, assim como se ele é clicável ou não.

“Escolhe Lista” é um botão que, quando clicado, mostra uma lista de textos na qual o usuário pode escolher um deles. Ao mudar a propriedade de mostrar a lista para verdadeira, fará a lista ser procurável. Outras propriedades envolvem a aparência do botão e se ele pode ser ou não clicado.

“Legenda” é um componente usado para mostrar texto. A legenda é mostrada por meio da propriedade de texto. Outras propriedades da legenda controlam a aparência e a localização do texto.

A Figura 9 mostra os elementos dentro da categoria de “Organização”, destes será discorrido apenas o elemento “Organização Em Tabela”.

Figura 9: Elementos dentro de “Organização”

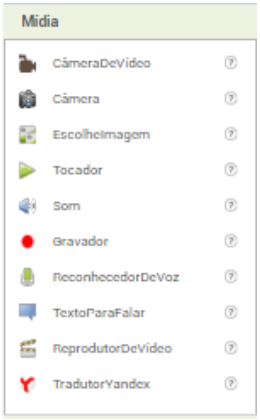


Fonte: Próprio Autor.

“Organização Em Tabela” é um componente utilizado para mostrar um grupo de componentes em um formato tabular. Esse componente é um meio de formatar a localização dos outros elementos escolhidos para o projeto.

A Figura 10 mostra os elementos dentro da categoria de “Mídia”, destes será discorrido os elementos “Reconhecedor De Voz” e “Texto Para Falar”, os elementos fundamentais para o projeto desenvolvido, já que são esses os que chamam as funções relacionadas ao *Google Cloud Speech-to-Text*, sistema utilizado para reconhecimento de voz.

Figura 10: Elementos dentro de “Mídia”



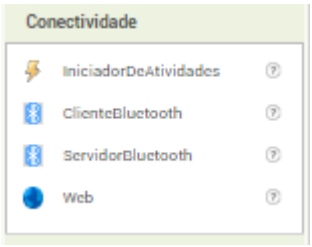
Fonte: Próprio Autor.

“Reconhecedor De Voz” é um componente utilizado para ouvir o usuário falando e converter a linguagem falada em texto utilizando a ferramenta de reconhecimento de voz do Android (no caso, o *Google Cloud Speech-to-Text*).

“Texto Para Falar” é um componente utilizado para fazer que o aplicativo fale um texto.

A Figura 11 mostra os elementos dentro da categoria de “Conectividade”, destes será discorrido apenas o elemento “Cliente Bluetooth”.

Figura 11: Elementos dentro de “Conectividade”



Fonte: Próprio Autor.

“Cliente Bluetooth” é um componente utilizado para controlar o serviço de *Bluetooth* do celular.

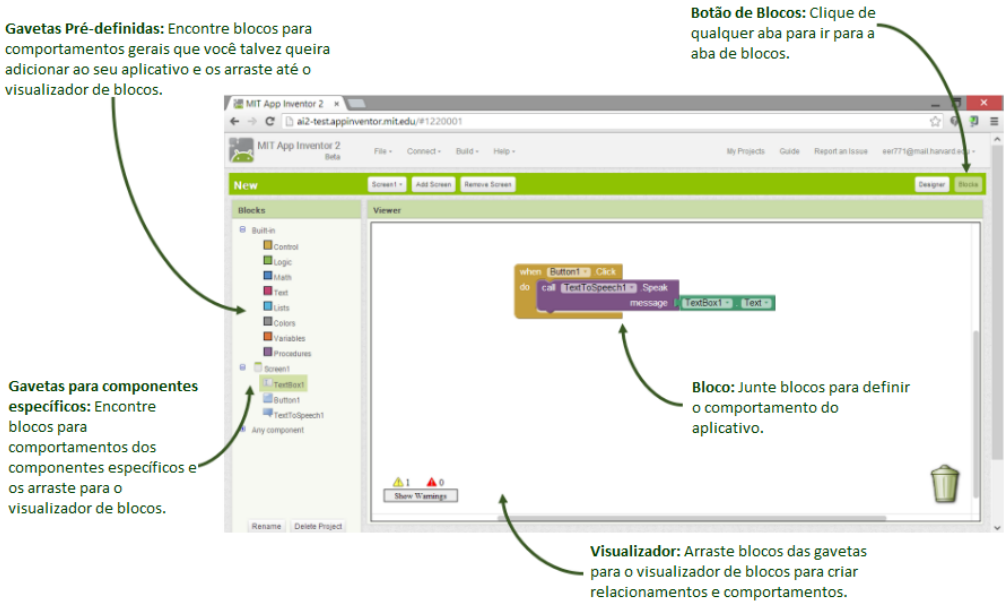
Com isso, se tem um entendimento básico do que cada um dos elementos utilizados do *App Inventor Designer* na execução do projeto.

3.2.2 *App Inventor Blocks Editor*

É o ambiente em que se programa o comportamento do aplicativo pela junção de blocos.

A Figura 12 mostra como é o ambiente ao realizar a programação dos blocos no *App Inventor*, além de conter um breve resumos das funcionalidades. Neste texto serão explicadas as possibilidades de rotinas a serem aplicadas pelo *App Inventor*. Se dará foco nas instruções mais comumente utilizadas para a criação de aplicativos móveis.

Figura 12: Ambiente novo de programação de aplicativo no App Inventor



Fonte: Próprio Autor.

Para exemplificar os diferentes tipos de comando disponíveis na forma de blocos, será feito um breve resumo do que cada opção vista na Figura 13 tem para oferecer ao usuário.

Figura 13: Opções de blocos no App Inventor.

Fonte: Próprio Autor.

Para os blocos internos, têm-se o seguinte:

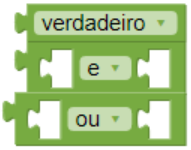
"Controle": Fornece funções condicionais do tipo: para tal condição, fazer atividade definida pelo usuário; se, então, senão; enquanto, fazer; e outras gerais. Em sumo, contém os blocos de condições e de *loops*. A Figura 14 mostra exemplos de blocos do tipo "Controle".

Figura 14: Opções de blocos de "Controle".

Fonte: Próprio Autor.

"Lógica": Possui funções do tipo verdadeiro ou falso, e/ou e comparativas. A Figura 15 exemplifica blocos do tipo "Lógica".

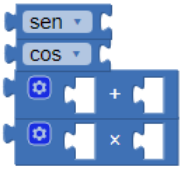
Figura 15: Opções de blocos de "Lógica"



Fonte: Próprio Autor.

"Matemática": Têm funções matemáticas como seno, cosseno, soma, multiplicação, entre outras. A Figura 16 exemplifica blocos do tipo "Matemática".

Figura 16: Opções de blocos de "Matemática"



Fonte: Próprio Autor.

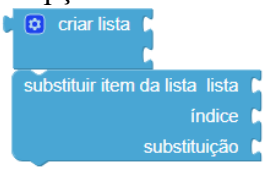
"Texto": Fornece funções que envolvem *strings* e textos em geral. A Figura 17 exemplifica blocos do tipo "Texto".

Figura 17: Opções de blocos de "Texto"

Fonte: Próprio Autor.

"Listas": Possui funções que envolvem a criação, e modificações de listas do tipo da parte de "Escolhe Lista" do *App Inventor Designer*. A Figura 18 exemplifica blocos do tipo "Listas".

Figura 18: Opções de blocos de "Lista"



Fonte: Próprio Autor.

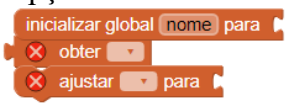
"Cores": Contém funções que envolvem a cor dos elementos do aplicativo. A Figura 19 exemplifica blocos do tipo "Cores".

Figura 19: Opções de blocos de "Cores"

Fonte: Próprio Autor.

"Variáveis": Possui funções que envolvem a inicialização, obtenção e ajuste de variáveis. A Figura 20 exemplifica blocos do tipo "Variáveis".

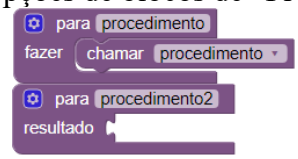
Figura 20: Opções de blocos de “Variáveis”



Fonte: Próprio Autor.

“Procedimentos”: Fornece funções que chamam procedimentos e relacionam procedimentos com rotinas. A Figura 21 exemplifica blocos do tipo “Procedimentos”.

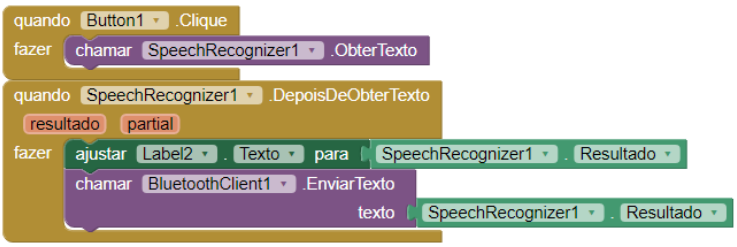
Figura 21: Opções de blocos de “Procedimentos”



Fonte: Próprio Autor.

Para as opções contidas em “Screen1” e em “Qualquer componente” se têm funções específicas para cada elemento escolhido para o projeto no *App Inventor Designer*. A Figura 22 contém alguns exemplos de blocos que estão nessas seções, contudo, um aprofundamento destes blocos e como utilizá-los será feito ao mostrar as rotinas envolvidas na criação do aplicativo.

Figura 22: Exemplo de blocos criados nas seções “Screen1” e “Qualquer componente”



Fonte: Próprio Autor.

A rotina mostrada na Figura 22 mostra o quão intuitivo é criar uma instrução no *App Inventor*. Essa rotina, por exemplo, faz que ao botão ser clicado, o *Google Cloud Speech-to-Text* seja chamado para pedir que o usuário fale, após o texto ser obtido, se mostra na Legenda o texto obtido e, por fim, a *string* obtida é enviada para o cliente de *Bluetooth* conectado.

Com isso, se tem conhecimento básico das funções do *App Inventor* e como se dá a criação de um aplicativo por meio dele.

3.3 *Bluetooth*

Nessa seção, será dada uma breve explicação de como se dá uma comunicação *Bluetooth* e uma apresentação do módulo utilizado.

De maneira simples, *Bluetooth* é um meio pelo qual dispositivos se comunicam sem fio em distâncias curtas (até 100m, apenas em condições ideiais, em condições normais, em até 10m). (HUANG; RUDOLPH, 2005).

Mais detalhadamente, o *Bluetooth* é uma rede a qual trabalha no padrão de uma rede pessoal sem fio (WPAN - *Wireless personal area network*) que opera na região de 2,4 GHz usando um método chamado de espectro de difusão em frequência variável (FHSS - *Frequency-hopping spread spectrum*). Por ele, um dispositivo mestre consegue se comunicar com até sete dispositivos dentro de uma rede *Bluetooth*, na qual os papéis de mestre e escravo podem ser mudados. (LINSKELL; DEWSBURY, 2019).

Por conta dessas características e da facilidade de acesso à dispositivos de comunicação *Bluetooth* se escolheu esse método para realizar a comunicação entre o aplicativo e o processador.

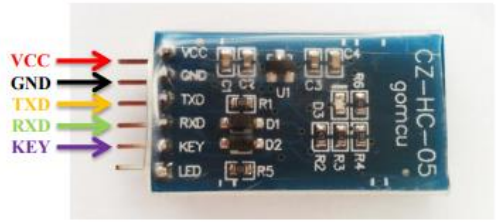
O módulo *Bluetooth* utilizado para o projeto foi o HC-05, isso se deu pela dificuldade de se encontrar módulos em Fortaleza, portanto, se utilizou o módulo que se tinha disponível.

O HC-05 é um módulo que permite utilizar facilmente o protocolo de comunicação serial para conexão sem fio. Essa comunicação faz que esse seja fácil de se conectar com controladores (como o Arduino que será utilizado) ou com computadores. (Manual do Usuário – Módulo Bluetooth HC-05, 2019).

A Figura 23 mostra a definição dos pinos do módulo Bluetooth, na qual: VCC possui a função de conectar o módulo em 5V; GND possui a função de conectar o módulo ao terra; TXD tem a função de se conectar ao ponto de recebimento de sinal do controlador, e representa o pino que envia o sinal para o controlador; RXD é o pino que conecta ao pino de envio de sinal do controlador, e representa o pino que recebe o sinal enviado pelo controlador; KEY representa um pino no qual se o sinal de entrada é 0, o módulo está pareando ou em modo de comunicação, se o sinal de entrada é 1, o módulo entra no modo AT que servem para

configurar, operar e obter dados do módulo. (Manual do Usuário – Módulo Bluetooth HC-05, 2019).

Figura 23: Definição de pinos do Módulo HC-05



Fonte: (Manual do Usuário – Módulo Bluetooth HC-05, 2019).

Com isso, se tem uma explicação básica sobre o que é e como funciona uma comunicação Bluetooth, além de saber qual módulo será utilizado para a comunicação e as explicações da pinagem deste.

3.4 *Arduino*

Nessa seção, será explicado o que é um *Arduino*, qual placa foi escolhida para o projeto e as funções lógicas utilizadas no projeto.

Arduino é uma plataforma eletrônica de arquitetura aberta baseada em *hardware* e *software* facéis de serem utilizados. É indicada para qualquer um que deseje fazer projetos interativos. A placa recebe os dados do ambiente por meio de vários tipos de sensores, e interage com o seu redor controlando atuadores em geral. Para dizer ao *Arduino* o que fazer, se deve escrever em código na linguagem de programação dele (C++) e usando o ambiente de desenvolvimento do *Arduino*. (Arduino A, 2019).

Com base nessa definição, já se entende porque foi escolhida essa plataforma para uso, já que é de fácil acesso e de programação simples, como será visto.

Há diversas placas *Arduino* no mercado, se preferiu utilizar a placa Uno porque essa placa já atendia as necessidades do projeto e se tinha fácil acesso a ela na universidade.

3.4.1 *Arduino Uno*

É uma placa microcontrolada baseada no ATmega328P, ela possui 14 pinos digitais, que podem ser utilizados como entrada ou saída, e dos quais 6 podem ser usados como saídas PWM; 6 entradas analógicas; um cristal de 16Mhz, uma conexão USB, uma entrada de

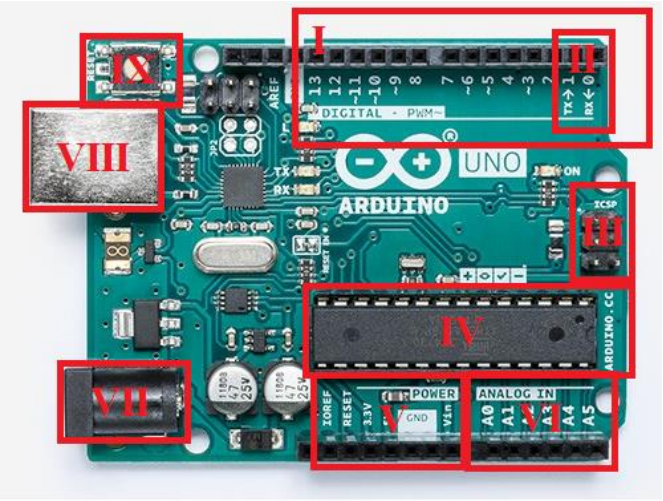
alimentação, um cabeçote de ICSP (*In Circuit Serial Programming* - Programação Serial em Circuito) e um botão de *reset*. É a primeira placa em uma série de placas *Arduino* USB e o modelo referência para a plataforma. (Arduino B, 2019).

Além disso, alguns pinos tem funções específicas, destes pinos, vale citar apenas os pinos de comunicação serial RX e TX, já que são os únicos desses que serão utilizados no projeto.

Esses pinos são utilizados para receber (RX) e transmitir (TX) informação serial. Eles são conectados aos pinos correspondentes do chip serial da placa. (Arduino B, 2019).

A Figura 24 representa uma placa Arduino Uno, com suas características ressaltadas pelas marcações em vermelho: I corresponde aos pinos digitais, nos quais os pinos PWM são representados com um til (~); II mostra os pinos seriais RX e TX; III é o cabeçote de ICSP; IV é a placa do controlador; V representa os pinos de alimentação; VI são as entradas analógicas; VII a entrada de alimentação; VIII a entrada USB; e IX o botão de *reset*.

Figura 24: Placa Arduino Uno



Fonte: Próprio Autor.

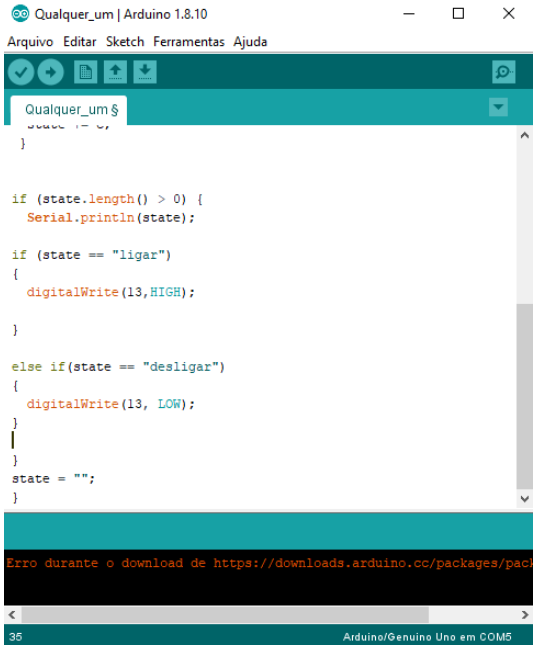
3.4.2 *Ambiente de Desenvolvimento*

Conforme dito anteriormente, existe um ambiente de desenvolvimento no qual se programa o comportamento desejado para o *Arduino*.

Essa plataforma é representada pela IDE (*Integrated Development Environment* - Ambiente de Desenvolvimento Integrado) disponibilizada no próprio site do *Arduino*, e é um modo simples de escrever código e gravar esse código na placa.

A Figura 25 mostra como é o ambiente de desenvolvimento, exemplificando no espaço com o fundo branco onde se é escrita o código da programação, no espaço em preto onde são mostrados os avisos e o status de compilação, além do espaço em verde, o qual contém, respectivamente as funções “Verificar”, “Carregar”, “Novo”, “Abrir” e “Salvar”.

Figura 25: Ambiente de desenvolvimento Arduino



Fonte: Próprio Autor.

Sabendo agora como se dá o ambiente de programação do Arduino, pode-se inicializar as discussões sobre as bibliotecas, rotinas e comandos utilizados.

3.4.2.1 Bibliotecas

A única biblioteca fora das funções nativas do Arduino utilizada é a “SoftwareSerial” desenvolvida para permitir comunicação serial em outros pinos digitais do Arduino, utilizando *software* para replicar a funcionalidade. (Arduino C, 2019).

3.4.2.2 Rotinas

No projeto não foi criada nenhuma rotina original, se utilizando apenas das rotinas tradicionais da programação em Arduino: setup() e loop().

A rotina setup() consiste em uma função que é chamada quando o código é inicializado, dentro dela são inicializadas variáveis, modos de pinos, bibliotecas, etc. Essa rotina só roda uma vez a cada *reset* da placa. (Arduino C, 2019)

A rotina loop() é lida após a setup() e faz *loops* consecutivos, o que permite ao programa mudar e responder. É utilizada pra controlar ativamente a placa. (Arduino C, 2019)

3.4.2.3 Comandos

Aqui serão listados todos os comandos diferentes utilizados durante a programação do projeto, para que durante a explicação das rotinas se tenha um entendimento melhor do que cada uma está realizando. Todas as definições foram retiradas do site do *Arduino*. (Arduino C, 2019)

#include: utilizado para incluir bibliotecas externas ao *sketch*.

SoftwareSerial: utilizado para criar um objeto da biblioteca SoftwareSerial cujo nome deve ser realizado da seguinte maneira: *SoftwareSerial* Nome (rxPin, txPin), em que rxPin é o pino que vai receber os dados e txPin o pino que vai transmitir os dados.

String: constrói uma instância da classe String.

void: palavra utilizada em declarações de funções. Indica que não é esperado que a função retorne alguma informação.

begin: indica a velocidade para a comunicação serial.

pinMode: configura o pino especificado para funcionar como entrada ou saída.

while: um *loop* irá se repetir continuamente, e indefinidamente, até que a expressão dentro dos parênteses se torne falsa.

available: pega o número de *bytes* (caracteres) disponível para a leitura de uma porta serial.

delay: pausa o programa por uma quantidade especificada de tempo (em milissegundos).

char: tipo de dado usado para armazenar um caractere.

read: retorna um caractere que foi lido pelo pino RX da porta serial do *software*.

length: retorna o tamanho da *string* em caracteres.

if: checa uma condição e executa o comando a seguir, ou um bloco de comandos delimitados por chaves, se a condição for verdadeira.

printIn: Escreve informação para o pino de transmissão da porta serial do *software*.

digitalWrite: aciona um valor HIGH ou LOW em um pino digital, em que HIGH corresponde ao “1” lógico e LOW ao “0” lógico.

Com isso, se tem noção do que se trata a plataforma *Arduino*, porquê se escolheu a placa Uno e quais as funcionalidades dessa placa, o ambiente de desenvolvimento e as funções lógicas utilizadas na programação do microprocessador para o projeto.

3.5 Considerações Finais

Este capítulo abordou os conceitos e aspectos que envolvem desde a criação do aplicativo, passando pelo método de reconhecimento de voz, a comunicação *Bluetooth* e o microprocessador utilizado para processar as informações e realizar os comandos.

Tendo isso em mente, se pode inicializar o estudo do projeto em si, apresentando como se deu a criação do aplicativo com as ferramentas explicadas, além do processamento da *string* enviada e a apresentação do protótipo criado.

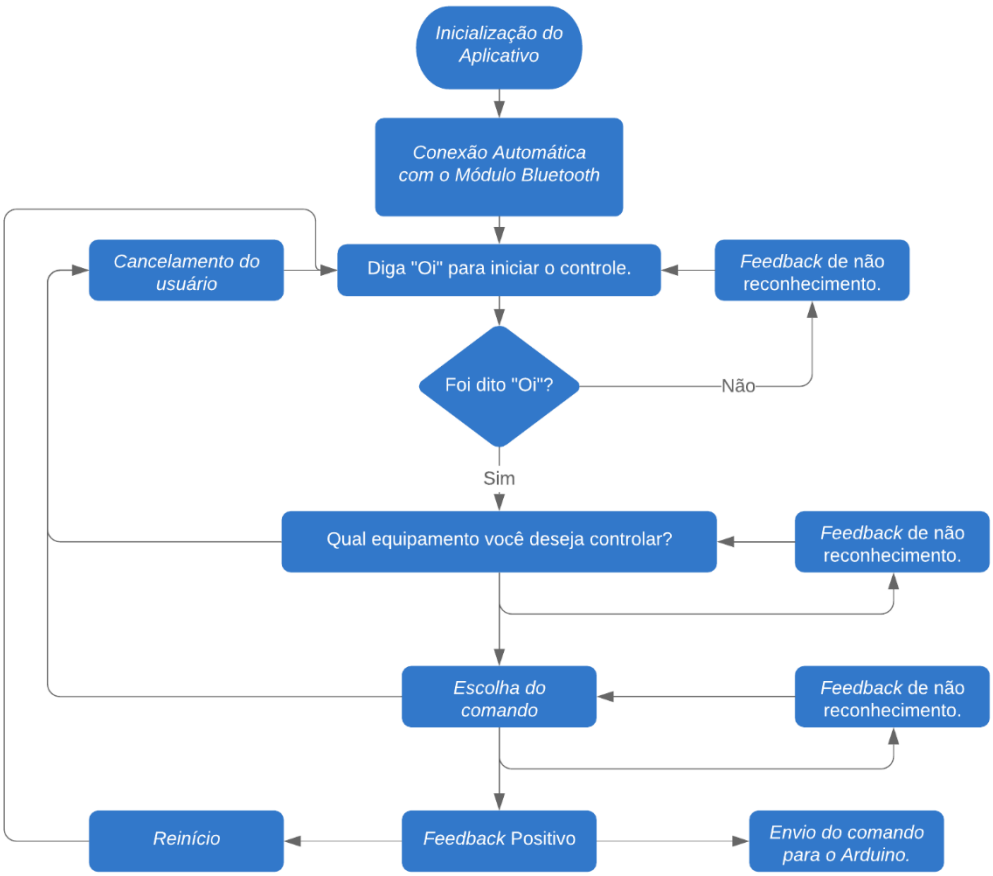
4 SISTEMA DE RECONHECIMENTO DE VOZ PARA AUTOMAÇÃO RESIDENCIAL UTILIZANDO *APPINVENTOR* E *ARDUINO*.

Neste capítulo é apresentado o modelo do projeto, seguindo as etapas, iniciando com o projeto do aplicativo, junto ao fluxograma do *AppInventor* com a explicação e o código em anexo; partindo para o do Arduino, também com as explicações e código em anexo e terminando mostrando como ficou o protótipo.

4.1 Aplicativo *AppInventor*

A Figura 26 apresentada o fluxograma da sequência lógica criada no *AppInventor*. Nessa seção, será discutido sobre cada elemento do fluxograma, exemplificando quais equipamentos podem ser controlados, além dos comandos existentes e as rotinas de programação existentes. Para as informações do código programado no *AppInventor*, checar o código exibido no ANEXO A.

Figura 26: Fluxograma geral do aplicativo



Fonte: Próprio Autor.

Ao iniciar o aplicativo, o usuário recebe uma confirmação sonora de “Iniciando”, após o som ser exibido, é realizada a conexão entre o aparelho com o aplicativo instalado e o módulo *Bluetooth*.

Ao essa conexão ser estabelecida, é dado um comando de voz ao usuário “Diga “Oi” para iniciar o controle”. É realizado um teste para verificar se o usuário disse “Oi” ou outra informação, caso não tenha dito “Oi”, é informado que não foi reconhecido e se repete novamente a frase inicial; caso tenha dito “Oi”, se é perguntado ao usuário qual o equipamento que ele deseja controlar.

Os equipamentos disponibilizados para ser controlados foram selecionados se considerando um quarto simples. Com isso, se foi disponibilizado lâmpada, porta e ventilador.

O usuário deve selecionar dentre esses equipamentos qual deseja controlar, caso o que for reconhecido não esteja dentre as opções disponíveis, o sistema informa que não entendeu e pede para selecionar novamente o equipamento desejado; caso o usuário deseje cancelar a operação, se retorna para a frase inicial; e, por fim, caso reconheça algum dos equipamentos listados, será perguntado, dependendo do equipamento, o que se quer que seja realizado.

Para lâmpada, é feita a pergunta “Acender ou apagar a lâmpada?”, caso o usuário diga acender, as lâmpadas são acesas e é dado um *feedback* sonoro; caso o usuário diga apagar, as lâmpadas são apagadas e, também, é dado um *feedback* sonoro; caso o usuário deseje cancelar a operação, se retorna a frase inicial; caso não seja dito nenhuma das informações anteriores se é informado que o sistema não entendeu e é refeita a pergunta “Acender ou apagar a lâmpada?”. Após isso, é enviada a *string* referente ao comando para o *Arduino* e o sistema é reiniciado.

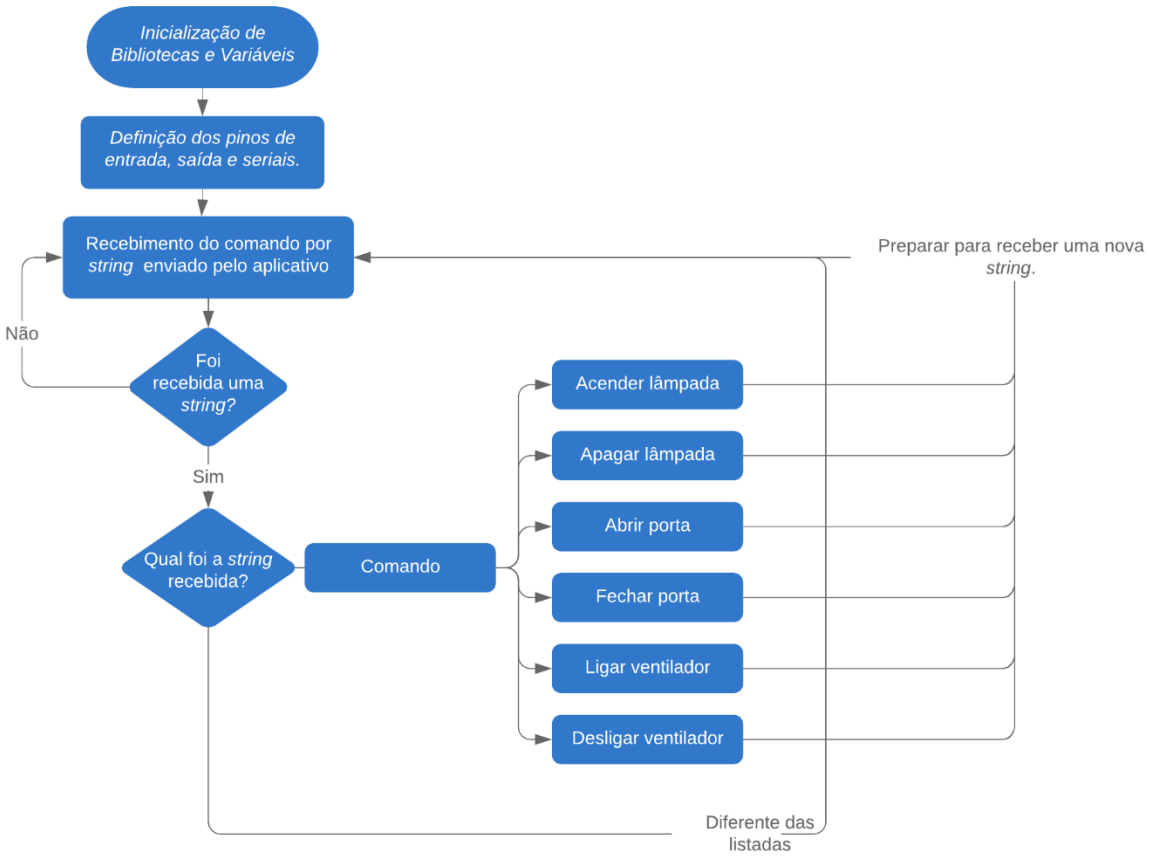
Para porta, a pergunta feita é “Abrir ou fechar a porta?”; e para ventilador “Ligar ou desligar o ventilador?”. Os demais procedimentos são similares aos descritos no parágrafo anterior.

Com isso, se tem um entendimento de como funciona o programa do aplicativo, bastando agora explicar como o comando enviado por *string* é processado no *Arduino*.

4.2 Recebimento da *String* e Acionamento por Arduino

A código lido pelo *Arduino* segue a sequência lógica apresentada no fluxograma da Figura 27. Nessa seção, será discutido sobre cada elemento do fluxograma, exemplificando como foi programado cada uma das seções do fluxograma.

Figura 27: Fluxograma geral do código lido pelo Arduino



Fonte: Próprio Autor.

Inicialmente foi incluída a biblioteca para as funções seriais, além de definir quais pinos são de saída, de entrada e os seriais.

Após isso, há a leitura da *string* enviada pelo aplicativo para o módulo *Bluetooth*. Nisso, é realizado um comparador que verifica se a *string* tem mais que 0 caracteres, o que verifica se foi enviada ou não uma *string*. Caso tenha sido recebida uma *string*, se passa para um comparador que verifica qual a *string* recebida, caso contrário, se retorna ao estado anterior até que seja recebida uma *string*.

No bloco comparador que verifica qual a *string* recebida, se têm 6 resultados possíveis: “Acender” e “Apagar”, referentes à lâmpada; “Abrir” e “Fechar”, referentes à porta; “Ligar” e “Desligar” referentes ao ventilador. Se a *string* recebida for alguma das citadas, é realizado o comando relacionado ao equipamento; caso a *string* seja diferentes dessas, se retorna ao bloco de recebimento da *string*; após o comando ter sido realizado, se retorna para o estado de receber a *string* enviada pelo aplicativo.

Para mais detalhes, o código está disponibilizado e comentado no ANEXO B.

4.3 Fluxograma Geral do Projeto

No ANEXO B, na Figura 36, há uma junção dos dois fluxogramas anteriormente apresentados nas Figuras 26 e 27. Com a análise do fluxograma, se consegue ter uma noção de como se dá o projeto em sua completude, desde a inicialização do aplicativo até o acionamento da lâmpada. No fluxograma geral são mostradas quais partes do projeto estão no aplicativo e quais estão no Arduino, além da função que une as duas plataformas.

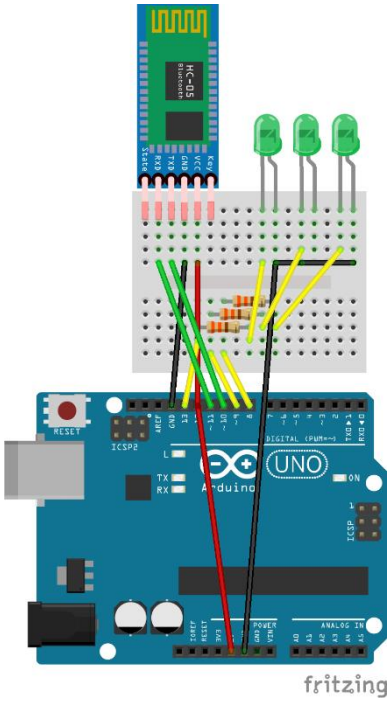
4.4 Esquemático da Montagem do Projeto e Protótipo

A montagem do projeto foi realizada se utilizando de uma placa *Arduino* Uno, conjuntos de jumpers, um módulo *Bluetooth* HC-05, um resistor de 330kΩ, um LED verde e um *smartphone* com sistema *Android*.

O esquemático da montagem é exemplificado na Figura 28, na qual o pino 13 do Arduino representa a saída resultante do comparador, para acender ou apagar a lâmpada; o pino 8 representa a saída resultante do comparador, para abrir ou fechar a porta; o pino 9 representa a saída resultante do comparador, para ligar ou desligar o ventilador; os pinos 10 e 11 representam, respectivamente, os pinos de TX e RX; os outros pinos utilizados correspondem apenas aos pinos de alimentação de 5V e GND. O smartphone é externo ao circuito.

A lâmpada, a porta e o ventilador foram simulados utilizando LEDs.

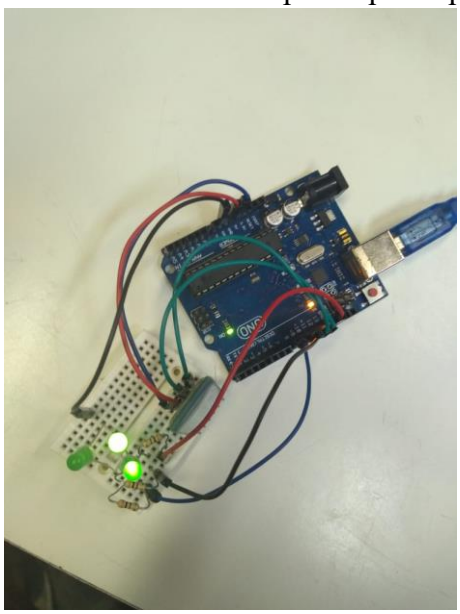
Figura 28: Esquemático de montagem do circuito do projeto.



Fonte: Próprio Autor.

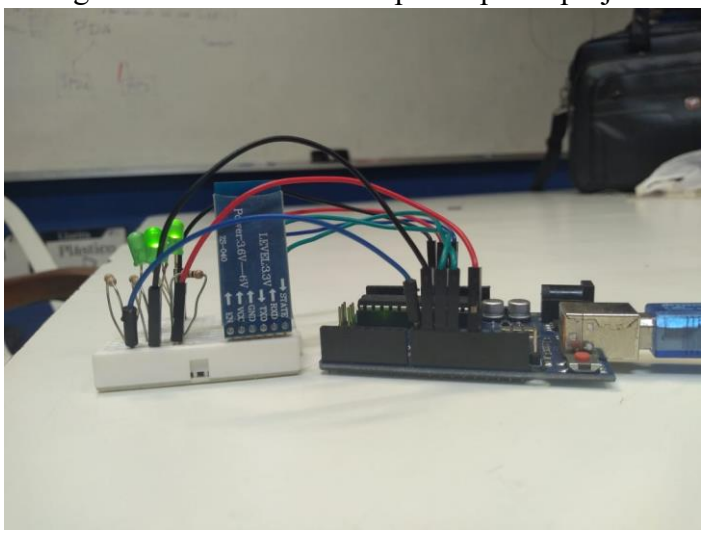
Com isso, se fez a montagem do protótipo com base no circuito da Figura 28. A Figura 29 representa uma vista de cima da montagem, e a Figura 30 representa uma vista lateral do projeto. Com essas duas Figuras é possível visualizar o que foi dito anteriormente sobre a definição dos pinos.

Figura 29: Visão de cima do protótipo do projeto.



Fonte: Próprio Autor.

Figura 30: Visão lateral do protótipo do projeto.



Fonte: Próprio Autor.

4.5 Considerações Finais

Este capítulo se apresentou o modelo do projeto, apresentando os fluxogramas da lógica de programação tanto do *AppInventor* e do *Arduino*. Por fim, foi mostrado o circuito a ser montado e o protótipo do projeto.

Com isso, se consegue partir para a parte das análises dos resultados atingidos pelo protótipo, verificando a viabilidade, efetividade e reprodutibilidade do projeto.

5 ANÁLISE QUANTITATIVA E QUALITATIVA DO APLICATIVO DE RECONHECIMENTO DE VOZ PARA AUTOMAÇÃO RESIDENCIAL.

Neste capítulo, são mostrados os resultados obtidos pelo projeto, por meio de demonstrações quantitativas, no qual serão apresentadas métricas para mostrar o quão efetivo é o protótipo. Por fim, demonstrações qualitativas baseadas nos resultados demonstrados, mostrando que a solução é viável e de fácil solução e implementação, sendo citadas as vantagens e desvantagens do sistema e comparando com outras soluções disponíveis no mercado.

5.1 Análise Quantitativa

Essa análise é realizada pois se sabe que classificadores erram, e é importante saber o quanto eles erram e em quais condições esses erros são cometidos com mais frequência.

Alguns dos maiores fatores que influenciam o reconhecimento correto ou não são: o quão quieto é um ambiente (quanto mais quieto mais alta a eficiência), a qualidade dos microfones e *hardware* de áudio (quanto melhores mais alta a eficiência), sotaques e vozes atípicas podem atrapalhar o reconhecimento e quanto mais simples a gramática maior a eficiência (*Sun Microsystems,* 1998).

A maioria dos erros caem nas seguintes categorias: rejeição; reconhecimento errado e disparamento errado. O erro de rejeição acontece quando o usuário fala, mas o reconhecedor não entende o que foi dito, ou seja, não é profuzida uma saída do que foi reconhecido; o erro de reconhecimento errado é quando se dá um retorno diferente do que foi dito; por fim, o erro de disparamento errado se dá ao reconhecedor retornar algo sem o usuário ter falado nada.

Pensando nisso, para a análise quantitativa, foram realizados testes com usuários femininos e masculinos, para testar o reconhecimento com diferentes timbres de voz. Nesse teste foram realizadas mudanças em duas variáveis, sendo estas distâncias (0m e 1m) e ruído (entre 70dB e 80dB (ambiente silencioso) e entre 80 e 90dB (ambiente com conversação moderada)). O ruído foi medido com auxílio do aplicativo “Sound Meter” para *Android*.

Os testantes foram orientados a falar os comandos de forma imperativa, até por isso os comandos foram definidos como verbos no imperativo.

Para cada usuário, se foi repetido o teste 10 vezes. A verificação foi realizada utilizando apenas dois comandos em cada condição “Ligar” e “Desligar”, o resultado foi marcado como positivo se foi reconhecida a *string* correta e se o LED do protótipo atendeu ao comando; caso contrário, o resultado foi marcado como negativo.

Com isso, se fará uma análise de cada um dos cenários, iniciando com as duas distâncias nos casos sem ruído e finalizando com as duas distâncias nos casos com ruído.

Os resultados para cada caso são demonstrados nas Figuras 31, 32, 33 e 34, com uma demonstração do número de acertos e erros no total de testes e a percentagem de efetividade do protótipo.

5.1.1 *Casos sem Ruído.*

Para os casos sem ruído, os usuários foram chamados para um ambiente silencioso em que não houvesse tanta interferência de pessoas externas, nem animais para interferir no reconhecimento de voz. Nesses casos foram realizados testes a 0m e a 1m do receptor de voz (*smartphone*).

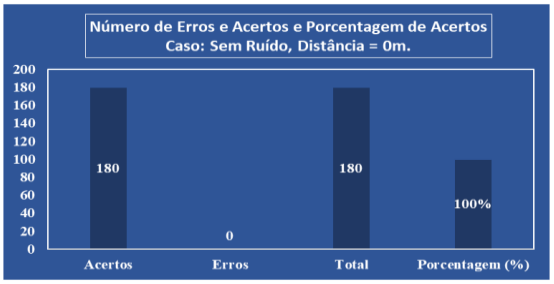
5.1.1.1 Distância 0m

Esse teste foi realizado com o usuário com o celular na mão, o mais próximo o possível da boca.

Para esse caso, dentro das situações sem ruído, não houve erros por parte do sistema para nenhum dos comandos, havendo apenas uma certa demora nos casos em que a conexão do *smartphone* com a internet estava fraca, esse problema é geral para todos os casos analisados na seção de análise quantitativa.

A Figura 31 representa os resultados para esse caso.

Figura 31: Resultados Caso Sem Ruído, Distância 0m



Fonte: Próprio Autor.

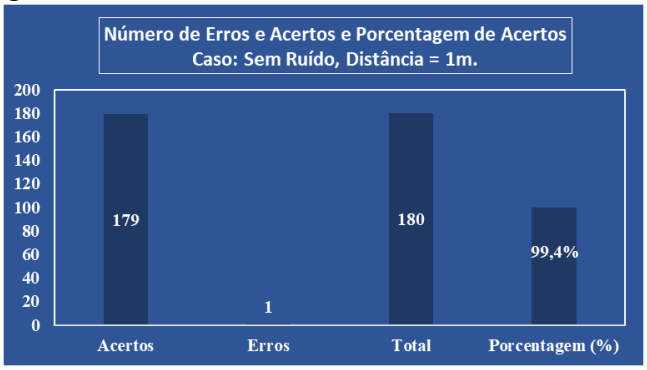
5.1.1.2 Distância 1m

Esse teste foi realizado com o celular em uma distância de 1m do usuário, sendo esta distância aferida com o auxílio de uma trena.

Para esse caso, dentro das situações sem ruído, novamente, não houve erros por parte do sistema.

A Figura 32 representa os resultados para esse caso.

Figura 32: Resultados Caso Sem Ruído, Distância 1m



Fonte: Próprio Autor.

5.1.2 *Casos com Ruído*

Para os casos com ruído, os usuários foram chamados para um ambiente em que houvesse conversa/música/vento, entre outros elementos que fossem capazes de elevar o ruído.

O teste se verificou impossível de ser realizado em condições nas quais houvesse um interlocutor muito próximo ao receptor de áudio, mesmo que a voz do usuário fosse imperativa, o sistema ainda reconhecia a fala desse interlocutor, o que impedia o reconhecimento da *string* correta.

Portanto, o teste foi realizado em um ambiente em que houvesse conversas paralelas ou música, mas em que o causador do ruído não estivesse tão próximo do receptor de voz.

Novamente, foram realizados testes a 0m e a 1m do receptor de voz (*smartphone*).

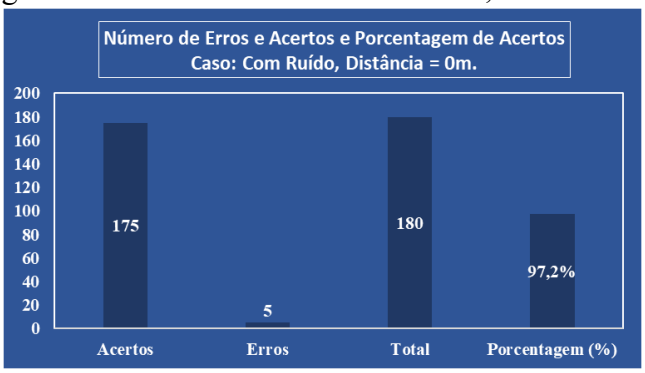
5.1.2.1 Distância 0m

Esse teste foi realizado com o usuário com o celular na mão, o mais próximo o possível da boca.

Para esse caso, dentro das situações com ruído, houve alguns erros, normalmente quando o usuário não utilizava a voz de modo imperativo, mas nada que comprometesse a utilização do sistema, já que a taxa de acerto permaneceu bastante alta, conforme verificado na Figura 38.

A Figura 33 representa os resultados para esse caso.

Figura 33: Resultados Caso Com Ruído, Distância 0m



Fonte: Próprio Autor.

5.1.2.2 Distância 1m

Esse teste foi realizado com o celular em uma distância de 1m do usuário, sendo esta distância aferida com o auxílio de uma trena. Para esse caso, dentro das situações com ruído, houve alguns erros, normalmente com o reconhecimento errado da *string* ou com nenhuma *string* reconhecida.

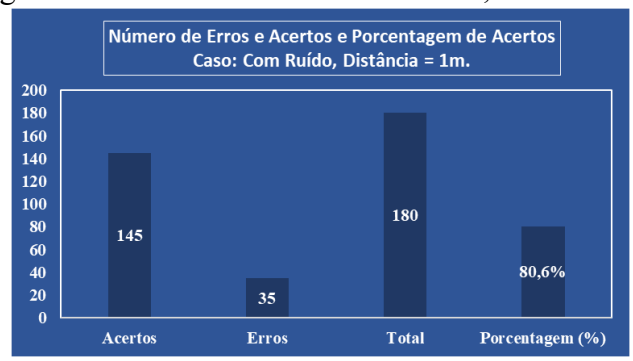
Nos casos de reconhecimento errado da *string*, esse normalmente acontecia quando o ruído era muito elevado, e acabava sendo reconhecida alguma fala externa.

Nos casos em que não se reconhecia nenhuma *string*, normalmente era mostrada na tela do *smartphone* a *string* inicial, mas depois o sistema entendia que não era o que tinha sido falado e ficava esperando por uma nova entrada que, por conta do ruído não era reconhecida.

Em alguns casos, também, após alguns segundos sem reconhecer uma *string*, o sistema informava que nada foi reconhecido e pedia para tentar novamente.

A Figura 34 representa os resultados para esse caso.

Figura 34: Resultados Caso Com Ruído, Distância 1m

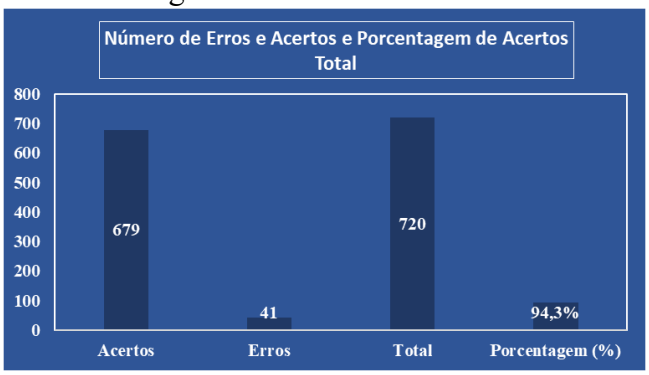


Fonte: Próprio Autor.

5.1.3 *Resultados Quantitativos Totais*

Por fim, a Figura 35 apresenta o resultado total de todas as condições testadas, a partir dela, e analisando também as Figuras 31 a 34, pode-se tirar algumas conclusões sobre o protótipo criado.

Figura 35: Resultados Totais



Fonte: Próprio Autor.

Os resultados mostrados levam a conclusão de que os resultados são muito positivos para os casos sem ruído, representando uma taxa de acerto muito próxima aos 100%.

Para os casos com ruído, os resultados dão uma queda, com alguns erros no caso de 0m, mas ainda se mostrando bem viável, com taxa de acerto acima de 95%; já no caso de 1m, têm-se resultados que não se mostram favoráveis a implementação nessas condições, já que a taxa de acerto ficou por volta de 80%.

Dito isso, o protótipo se mostrou bastante eficiente para a maioria dos casos propostos, e, mesmo considerando o caso de 1m com ruído, a taxa de acerto total ficou por volta de 95%, demonstrando que o projeto é viável.

5.2 Análise Qualitativa

A solução proposta teve êxito em reconhecer a voz do usuário, transmitir a informação reconhecida por *Bluetooth* e realizar um acionamento com base no que foi reconhecido, no caso uma lâmpada. Isso se utilizando de um circuito simples e de uma interface fácil de ser utilizada, sem muita complexidade.

Conforme foi visto durante a elaboração do trabalho, todas as tecnologias que envolvem o projeto realizado são de fácil acesso e de simples execução, apesar de uma boa necessidade de embasamento teórico para se entender as tecnologias por trás das ferramentas utilizadas.

Um dos pontos mais positivos desse projeto é o fato de que o aplicativo desenvolvido funciona no sistema *Android*, o qual representava, no período de outubro de 2018 a outubro de 2019, 76,67% dos sistemas operacionais de *smartphones* no mundo. Observando apenas o Brasil esse número é ainda maior, representando 87,86% dos sistemas operacionais para o mesmo período (Statcounter, 2019). Portanto, há um fácil acesso dos usuários ao aplicativo, já que a plataforma na qual este funciona tem ampla ocupação no mercado.

Outro ponto extremamente positivo do projeto é que este utiliza tecnologias livres. Todas as ferramentas estudadas são *open source*: a *Google Cloud Speech-to-Text API*, O *MIT App Inventor* e o *Arduino*. Além disso, todas essas ferramentas possuem documentações e tutoriais disponíveis e fáceis de serem acessados nos respectivos *websites*.

Por conta desses dois pontos citados, é fácil se desenvolver e aprimorar o projeto realizado, já que há discussões diariamente em fórums como o *GitHub* sobre diferentes métodos de lidar com problemas que envolvem esses instrumentos.

Mais um fato que valida o modelo para a automação com reconhecimento de voz proposta é o baixo custo envolvido no projeto. Considerando que o *smartphone* não faz parte da proposta e é de responsabilidade do usuário, têm-se o seguinte: O *Google Cloud Speech-to-Text* é gratuito para reconhecimento de até 60 minutos, ou seja para uma solução em que o comando é dado por poucas palavras, como a proposta, a utilização dele é gratuita; o *MIT App Inventor* é também um ambiente de programação gratuito, ou seja, a criação do aplicativo também não é um custo.

Logo, em uma solução residencial, os gastos necessários para a aplicação da solução envolvem a placa *Arduino* Uno, na qual o sistema para programação também é disponibilizado gratuitamente, e o módulo *Bluetooth*. Fazendo uma breve pesquisa no *Mercado Livre* se encontrou os preços mais baixos e mais altos para esses componentes, verificando, portanto, os custos mínimos e máximos do projeto.

A placa *Arduino* Uno mais barata encontrada foi de R$ 23,00 e a mais cara de R$ 63,49. No caso do módulo *Bluetooth* HC-05 o mais barato encontrado foi de R$ 18,50 e o mais caro de R$ 77,70. (Mercado Livre, 2019).

Desse modo, o projeto mais barato sairia por R$ 42,50 e o mais caro por R$ 141,19. Mostrando que, além da eficiência do projeto, é um projeto de baixo custo, principalmente se souber procurar os componentes para encontrar o mais barato.

A questão custo é ainda mais um fator a ser analisado quando se observam soluções de reconhecimento de voz e automação existentes. Como exemplo comparativo será utilizado o *Google Home*, dispositivo interativo da *Google* que consiste de um assistente de voz para automação residencial, com diversas funções pré-definidas. Consultando também no *Mercado Livre*, o *Google Home* mais barato encontrado foi de R$ 159,00. Ou seja, mesmo o mais caro dos protótipos possíveis para a proposta sugerida é mais barato que a solução existente da *Google*. Obviamente, não se foram propostas as mesmas soluções que são fornecidas pelo *Google Home*, mas, conforme dito anteriormente, a continuidade e evolução do projeto é mais uma questão de programação e despendiamento de tempo do que custos financeiros.

5.3 Considerações Finais

As análises realizadas mostram que o projeto consiste de uma solução eficiente para o problema proposto de automação residencial por reconhecimento de voz. O protótipo desenvolvido demonstrou um ótimo índice de acerto, além de ter se mostrado, especialmente quando comparado com outras soluções existentes um projeto de baixo custo e com amplas possibilidades para desenvolvimento de novos projetos e evolução do protótipo.

A condição em que o projeto não se apresentou ideal foi em situações nas quais o usuário esteja distante do receptor de voz e em que haja ruído, se tornando, inclusive, inviável em situações nas quais houvesse um interlocutor causando ruído alto próximo ao receptor.

6 CONCLUSÃO E TRABALHOS FUTUROS

O presente trabalho apresentou uma solução para a inclusão digital de deficientes por meio de um sistema de baixo custo para automação residencial por meio de reconhecimento de voz.

O teste de conceito para que usuários com a visão totalmente comprometida fossem capazes de usufruir do produto foi realizado com efetividade, com a verificação da voz realizada de forma contínua, e o pareamento com o módulo realizado de forma automática, com o retorno para o usuário em cada uma das etapas, dando segurança para o deficiente.

Por meio de uma arquitetura simples, utilizando tecnologias livres, se desenvolveu um sistema que permitiu o acionamento de uma lâmpada. As ferramentas trabalhadas foram o *MIT AppInventor, Google Cloud Speech-to-Text* e *Arduino*.

Os objetivos propostos inicialmente para o trabalho foram todos bem resolvidos, já que foi criado um protótipo simples e fácil de ser manuseado, além de poder ser acessado por vários usuários, já que opera no sistema operacional *Android*.

A criação do aplicativo pelo *MIT AppInventor* se mostrou fácil e com diversos espaços para mudança e adaptação, sobretudo na parte estética do projeto.

Além disso, o embasamento teórico realizado para entender como o *Google Cloud Speech-to-Text* transforma a voz em informação textual foi bastante importante para que se possa propor novas soluções e se aprofundar nos algoritmos da documentação.

Conseguiu se perceber que um modelo baseado em comunicação *Bluetooth*, apesar de não ser o modelo mais sofisticado, é suficiente para resolver a proposta, especialmente em distâncias curtas.

Por fim, a utilização de um microprocessador *Arduino* foi essencial para a realização dos comandos de automação, e, também, por interligar o aplicativo com o objeto a ser controlado.

Em suma, se foi capaz de estudar os meios de reconhecimento de voz já existentes no mercado, criando um aplicativo capaz de reconhecer a voz e, por meio dela, criar uma *string*

que corresponda ao que foi falado, utilizando essa *string* para ativar ou desativar um comando conforme a *string* recebida por comunicação *Bluetooth* entre o celular e o *Arduino*.

6.1 Trabalhos Futuros

Além de novas funcionalidades como comandos, testes com métodos de reconhecimento de voz alternativos às redes neurais artificiais também estão previstos.

Uma discussão da performance entre modelos com Cadeia de Markov e com Redes Neurais pode ser realizada.

A possibilidade de tentar aprimorar o filtro de ruído para melhorar o resultado nas situações ruidosas também é possível como trabalho futuro.

A inicialização do aplicativo para plataformas móveis baseada em *Android* juntamente com o sistema operacional asseguraria a acessibilidade do sistema desde o momento da aquisição do celular/*tablet*.

O desenvolvimento da plataforma para o sistema operacional iOS seria de interesse aos usuários de produtos *Apple*.

A possibilidade de utilizar outros processadores mais baratos, linguagens de programação que permitam convergência mais rápidas, a possibilidade de se trabalhar com comunicação *Wi-fi* em vez de *Bluetooth* consistem em mais trabalhos futuros com base nesse projeto e a viabilidade de se trabalhar com o projeto com uma base de dados *offline* em vez da utilização da *Google Cloud*.

REFERÊNCIAS

Amazon. **Amazon Echo.** Disponível em: <https://www.amazon.com.br/b/ref=pd_sl_4ed0qgx41e_e_nodl?ie=UTF8&node=19877613011&ext=5311-29768&tag=hydrbrgk-20&hvpos=1t1&hvnetw=g&hvrand=13645018442554994141&hvpone=&hvptwo=&hvqmt=b&hvdev=m&hvdvcmdl=&hvlocint=&hvlocphy=1001773&hvtargid>. Acesso em 27/11/2019.

ANTON, Howard; RORRES, Chris. **Álgebra linear com aplicações**. Porto Alegre: Bookman, 2001.

Apple. **Aplicativo Casa.** Disponível em: <https://www.apple.com/br/ios/home/>. Acesso em 27/11/2019.

Arduino A. **What is Arduino?.** Disponível em: <https://www.arduino.cc/>. Acesso em 17/11/2019.

Arduino B. **Arduino UNO Rev3.** Disponível em: <https://create.arduino.cc/projecthub/products/arduino-uno-rev3>. Acesso em 17/11/2019.

Arduino C. **Language Reference.** Disponível em: <https://www.arduino.cc/reference/en/>. Acesso em 18/11/2019.

BARRETO, Guilherme. **Perceptron Multicamadas e o Algoritmo de Retropropagação do Erro.** Notas de aula da disciplina de Inteligência Computacional Aplicada, 2017. Disponível em <https://drive.google.com/open?id=1FrpFGpZvhD1CQXQCc2ZXFj0J7Y5oYrkB>. Acesso em 05/11/2019.

BIDARRA, Jorge e Diógenes; RODRIGUES, Carlos Eduardo. "XLUPA - Uma lente de aumento digital inteligente para pessoas com baixa visão". **ATIID 2005 - III Seminário e II Oficina "Acessibilidade, TI e Inclusão Digital"** , USP/Faculdade de Saúde Pública de São Paulo, 5 e 6 de Setembro de 2005. Disponível em <http://www.prodam.sp.gov.br/multimidia/midia/cd_atiid/conteudo/ATIID2005/MR3/02/XLupa-LenteAumentoDigital.pdf>. Acesso em 27/11/2019.

BISHOP, Christopher M. et al. **Neural networks for pattern recognition**. Oxford university press, 1995.

BOLDRINI, José. Luiz e outros. **Álgebra Linear, 3a ed., Harbra, São Paulo**, 1986.

BRAGA, P. de L. Reconhecimento de voz dependente de locutor utilizando Redes Neurais Artificiais. 2006. **Trabalho de Conclusão de Curso-Engenharia da Computação, Universidade de Pernambuco, Recife**, 2006. Disponível em <https://drive.google.com/open?id=1tTFTwmX4lslmIRGsKNY74FNNZenhPk8S>. Acesso em 27/11/2019.

CAPARICA, Monte. VOICE RECOGNITION USING A FUZZY MULTIPLE ATTRIBUTE MODEL. **Revista Nacional de Investigação Operacional**, v. 21, p. 1, 2001.

CIPRIANO, Jose Luis Gomez. Desenvolvimento de arquitetura para sistemas de reconhecimento automático de voz baseados em modelos ocultos de Markov. 2001. Disponível em<https://lume.ufrgs.br/bitstream/handle/10183/2633/000373873.pdf?sequence=1&isAllowed=y>. Acesso em 27/11/2019.

CORRÊA, Daniel Ruggeri. Automação e segurança residencial de baixo custo utilizando aplicativo Android, Bluetooth e reconhecimento de voz, em conjunto com Arduino. 2013. Disponível em: <https://drive.google.com/open?id=1SNbxUrhwL9E3ZGu_N6EnlAgPa2hmmfdO>. Acesso em 27/11/2019.

FU, King Sun; MENDEL, Jerry M. (Ed.). **Adaptive, Learning, and Pattern Recognition Systems: Theory and Applications**. Academic Press, 1970.

GEVAERT, Wouter; TSENOV, Georgi; MLADENOV, Valeri. Neural networks used for speech recognition. **Journal of Automatic control**, v. 20, n. 1, p. 1-7, 2010.

Google. **Cloud Speech-to-Text.** Disponível em: <https://cloud.google.com/speech-to-text/>. Acesso em 29/10/2019.

Google Store. **Google Nest Mini.** Disponível em: <https://store.google.com/product/google_nest_mini>. Acesso em 27/11/2019.

GM Electronic. **HC-05 Bluetooth Module User's Manual V1.0.** Disponível: <https://www.gme.cz/data/attachments/dsh.772-148.1.pdf>. Acesso em 17/11/2019.

HAYKIN, Simon. **Redes neurais: princípios e prática**. Bookman Editora, 2007.

HUANG, Albert; RUDOLPH, Larry. Bluetooth for programmers. **Massachusetts Institute of Technology, Cambridge**, 2005. Disponível em <http://people.csail.mit.edu/rudolph/Teaching/Articles/BTBook.pdf >. Acesso em 27/11/2019.

ICMC USP. **Redes neurais artificiais.** Disponível em: <http://conteudo.icmc.usp.br/pessoas/andre/research/neural/index.htm#intro>. Acesso em: 30/10/2019.

LEE, Akinobu. **The Julius book. Edition 1.0.3.** 2010. Disponível em <https://julius.osdn.jp/juliusbook/en/>. Acesso em 27/11/2019.

LIPPMANN, Richard P. Review of neural networks for speech recognition. **Neural computation**, v. 1, n. 1, p. 1-38, 1989.

LINSKELL, Jeremy; DEWSBURY, Guy. Assisted Living. In: **Handbook of Electronic Assistive Technology**. Academic Press, 2019. p. 215-258.

MARANGONI, Josemar Barone; PRECIPITO, Waldemar Barilli. Reconhecimento e sintetização de voz usando Java Speech. **Revista Científica Eletrônica de Sistemas de Informação (ISSN 1807-1872). Ano**, v. 2, 2017.

MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115-133, 1943.

Medium Corporation. **A Serverless Audio Transcription Pipeline.** Disponível em: <https://medium.com/@jlaham/serverless-audio-transcription-515cd7f67e9c>. Acesso em 29/10/2019.

Mercado Livre. **Eletrônicos, Áudio e Vídeo.** Disponível em: <https://home.mercadolivre.com.br/eletronicos/>. Acesso em 19/11/2019.

MIT. **MIT App Inventor.** Disponível em: <https://appinventor.mit.edu/>. Acesso em 06/11/2019.

MOREIRA, Nícolas de Araújo. Proposta de um front-end em Java para sintetizador de voz baseado no MBROLA. 2015. Disponível em <http://www.repositorio.ufc.br/bitstream/riufc/21091/1/2015_dis_namoreira.pdf>. Acesso em 27/11/2019.

OMS. **World report on vision.** Disponível em: <www.who.int/publications-detail/world-report-on-vision>. Acesso em 22/10/2019.

PILOTI, Jason Scalco. Sistema de automação residencial: acessibilidade no controle doméstico. 2016. Disponível em: <https://repositorio.ucs.br/xmlui/bitstream/handle/11338/1364/TCC%20Jason%20Scalco%20Piloti.pdf?sequence=1&isAllowed=y >. Acesso em 27/11/2019.

REINALDI, Leticia Ramos; DE CAMARGO JÚNIOR, Cláudio Rosa; CALAZANS, Angelica Toffano Seidel. Acessibilidade para pessoas com deficiência visual como fator de inclusão digital. **Universitas: Gestão e TI**, v. 1, n. 2, 2011.

ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, v. 65, n. 6, p. 386, 1958.

SANTOS, Jader Gustavo de Campos. Acessibilidade em aplicações desktop utilizando ferramentas Java. 2001. Disponível em <https://docplayer.com.br/amp/2571511-Universidade-tecnologica-federal-do-parana-campus-cornelio-procopio-especializacao-em-tecnologia-java-jader-gustavo-de-campos-santos.html>. Acesso em 27/11/2019.

Statcounter. **Mobile Operating System Market Share Worldwide.** Disponível em: <https://gs.statcounter.com/os-market-share/mobile/worldwide>. Acesso em 19/11/2019.

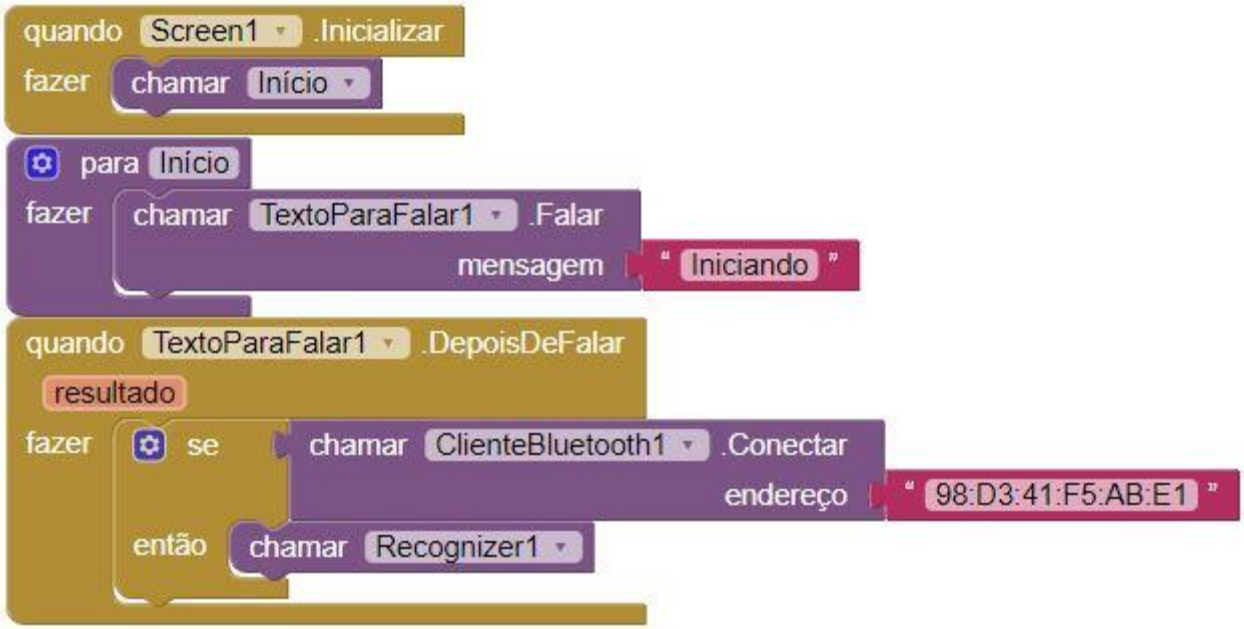
Sun Microsystems. **GNOME 2.0 Desktop: Developing With the Accessibility Framework**, Janeiro de 2003. Disponível em <https://drive.google.com/open?id=1Ks4QZv6YMUdcCXsZJ_zvnkHUAwnOWHmW>. Acesso em 27/11/2019.

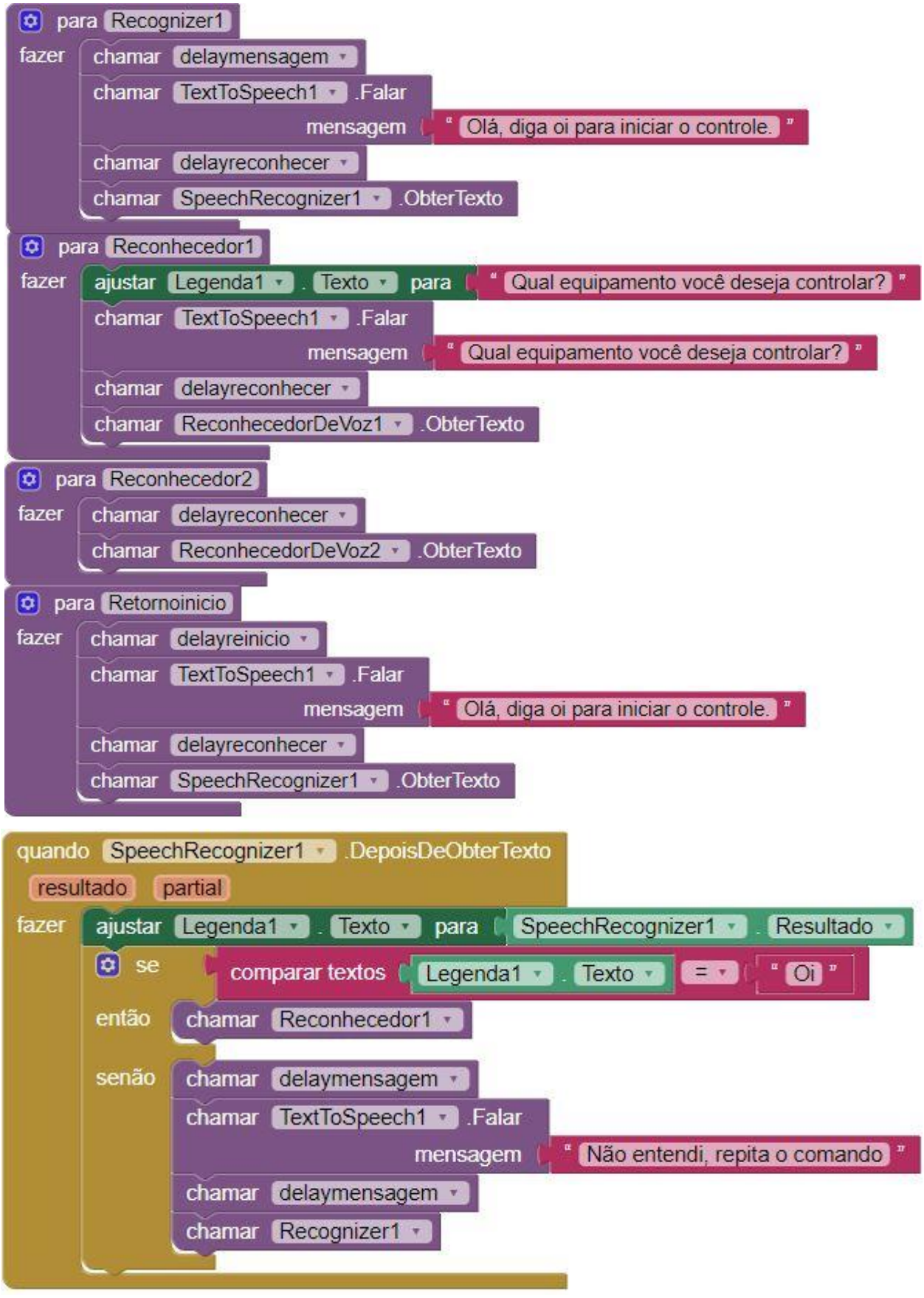
Sun Microsystems. **Java TM Speech API Programmer's Guide. Versão 1.0.** Palo Alto, Outubro de 1998. Disponível em <https://drive.google.com/open?id=1G3NU7sGdC_6q8rC7f5h7jZdnU53B7m4s>. Acesso em 27/11/2019.

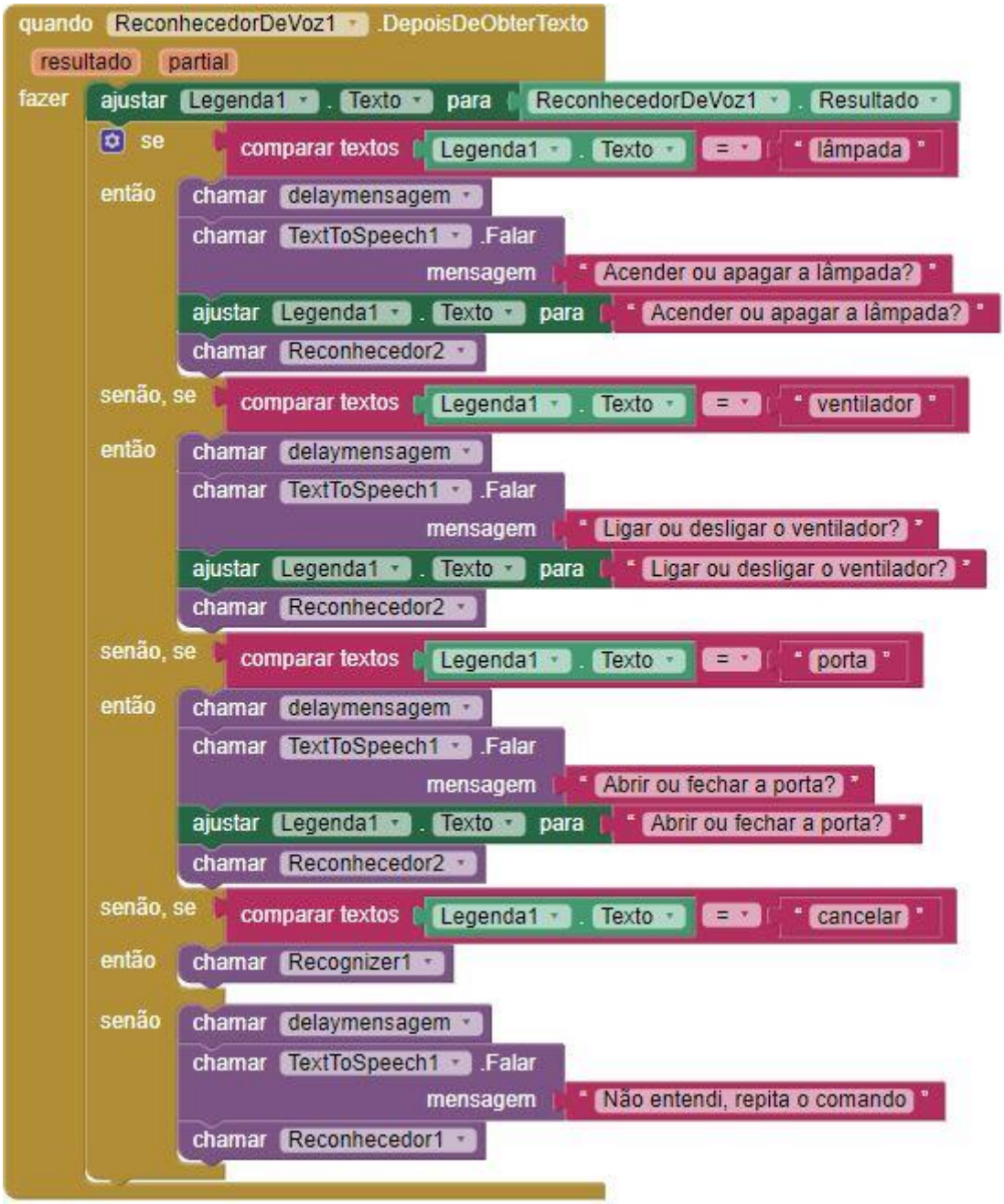
WEBB, Andrew R. **Statistical pattern recognition**. John Wiley & Sons, 2003.

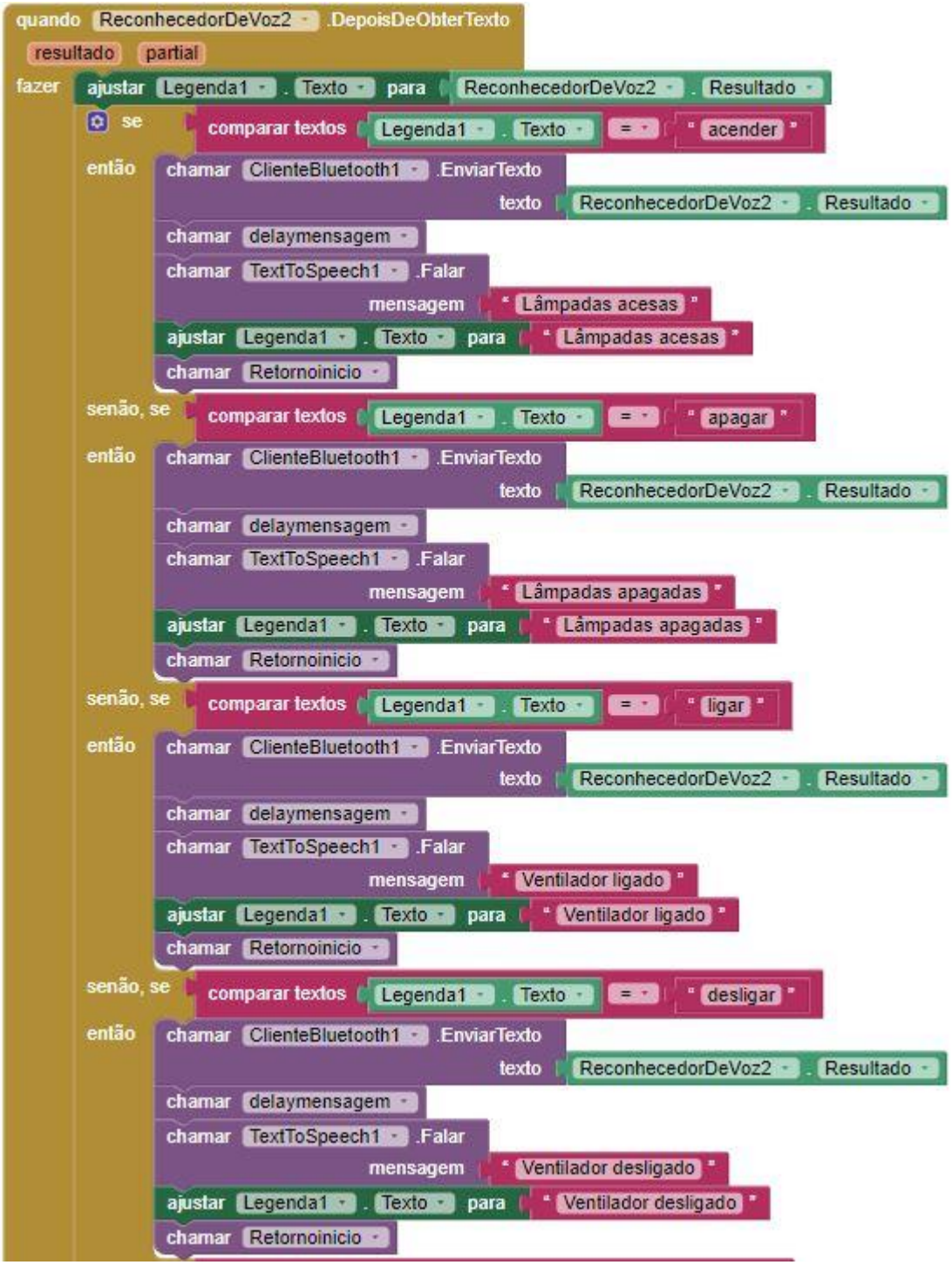
UFOP. **Redes neurais artificiais.** Disponível em: <http://www.decom.ufop.br/imobilis/redes-neurais-funcoes-de-ativacao/>. Acesso em: 13/12/2019.

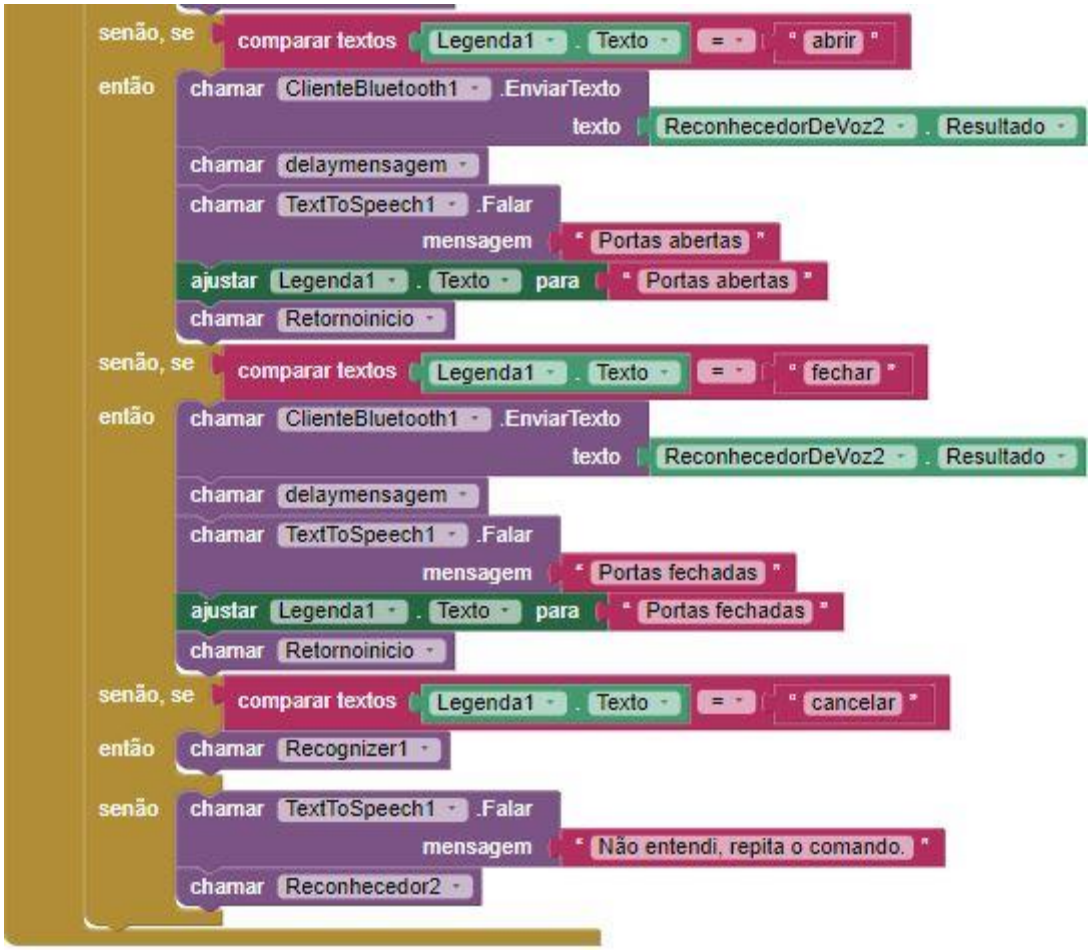
ANEXO A – CÓDIGOS *APP INVENTOR*.




para Recognizer1
fazer
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Olá, diga oi para iniciar o controle. "
chamar delayreconhecer
chamar SpeechRecognizer1 .ObterTexto
para Reconhecedor1
fazer
ajustar Legenda1 . Texto para " Qual equipamento você deseja controlar? "
chamar TextToSpeech1 .Falar
mensagem " Qual equipamento você deseja controlar? "
chamar delayreconhecer
chamar ReconhecedorDeVoz1 .ObterTexto
para Reconhecedor2
fazer
chamar delayreconhecer
chamar ReconhecedorDeVoz2 .ObterTexto
para Retornoinicio
fazer
chamar delayreinicio
chamar TextToSpeech1 .Falar
mensagem " Olá, diga oi para iniciar o controle. "
chamar delayreconhecer
chamar SpeechRecognizer1 .ObterTexto
quando SpeechRecognizer1 .DepoisDeObterTexto
resultado partial
fazer
ajustar Legenda1 . Texto para SpeechRecognizer1 . Resultado
se
comparar textos Legenda1 . Texto = " Oi "
então
chamar Reconhecedor1
senão
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Não entendi, repita o comando "
chamar delaymensagem
chamar Recognizer1


quando ReconhecedorDeVoz1 .DepoisDeObterTexto
resultado partial
fazer ajustar Legenda1 . Texto para ReconhecedorDeVoz1 . Resultado
se comparar textos Legenda1 . Texto = " lâmpada "
então chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Acender ou apagar a lâmpada? "
ajustar Legenda1 . Texto para " Acender ou apagar a lâmpada? "
chamar Reconhecedor2
senão, se comparar textos Legenda1 . Texto = " ventilador "
então chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Ligar ou desligar o ventilador? "
ajustar Legenda1 . Texto para " Ligar ou desligar o ventilador? "
chamar Reconhecedor2
senão, se comparar textos Legenda1 . Texto = " porta "
então chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Abrir ou fechar a porta? "
ajustar Legenda1 . Texto para " Abrir ou fechar a porta? "
chamar Reconhecedor2
senão, se comparar textos Legenda1 . Texto = " cancelar "
então chamar Recognizer1
senão chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Não entendi, repita o comando "
chamar Reconhecedor1


quando ReconhecedorDeVoz2 .DepoisDeObterTexto
resultado partial
fazer ajustar Legenda1 . Texto para ReconhecedorDeVoz2 . Resultado
se comparar textos Legenda1 . Texto = " acender "
então chamar ClienteBluetooth1 .EnviarTexto
texto ReconhecedorDeVoz2 . Resultado
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Lâmpadas acesas "
ajustar Legenda1 . Texto para " Lâmpadas acesas "
chamar Retornoinicio
senão, se comparar textos Legenda1 . Texto = " apagar "
então chamar ClienteBluetooth1 .EnviarTexto
texto ReconhecedorDeVoz2 . Resultado
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Lâmpadas apagadas "
ajustar Legenda1 . Texto para " Lâmpadas apagadas "
chamar Retornoinicio
senão, se comparar textos Legenda1 . Texto = " ligar "
então chamar ClienteBluetooth1 .EnviarTexto
texto ReconhecedorDeVoz2 . Resultado
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Ventilador ligado "
ajustar Legenda1 . Texto para " Ventilador ligado "
chamar Retornoinicio
senão, se comparar textos Legenda1 . Texto = " desligar "
então chamar ClienteBluetooth1 .EnviarTexto
texto ReconhecedorDeVoz2 . Resultado
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Ventilador desligado "
ajustar Legenda1 . Texto para " Ventilador desligado "
chamar Retornoinicio


senão, se
comparar textos Legenda1 . Texto = " abrir "
então
chamar ClienteBluetooth1 .EnviarTexto
texto ReconhecedorDeVoz2 . Resultado
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Portas abertas "
ajustar Legenda1 . Texto para " Portas abertas "
chamar Retornoinicio
senão, se
comparar textos Legenda1 . Texto = " fechar "
então
chamar ClienteBluetooth1 .EnviarTexto
texto ReconhecedorDeVoz2 . Resultado
chamar delaymensagem
chamar TextToSpeech1 .Falar
mensagem " Portas fechadas "
ajustar Legenda1 . Texto para " Portas fechadas "
chamar Retornoinicio
senão, se
comparar textos Legenda1 . Texto = " cancelar "
então
chamar Recognizer1
senão
chamar TextToSpeech1 .Falar
mensagem " Não entendi, repita o comando. "
chamar Reconhecedor2

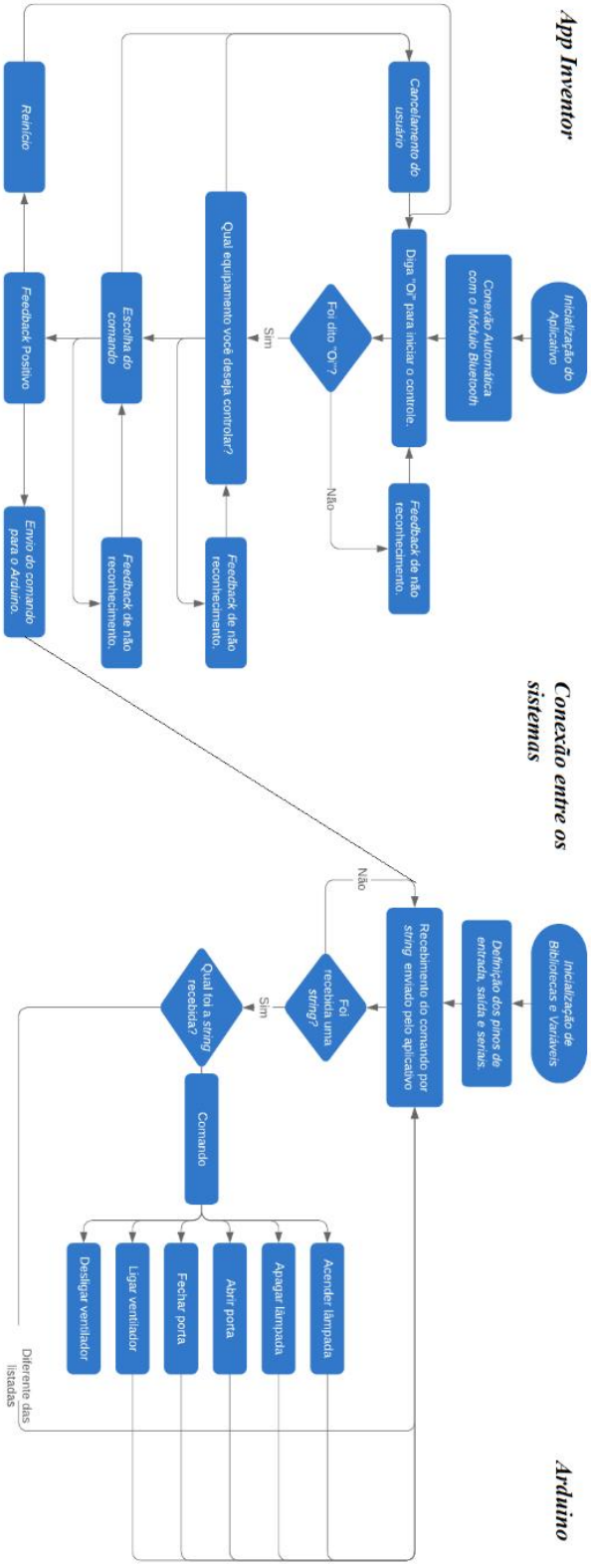
ANEXO B – CÓDIGOS *ARDUINO*.

```
#include <SoftwareSerial.h>
SoftwareSerial BT(10, 11);
String state;
void setup() {
BT.begin(9600);
  Serial.begin(9600);
    pinMode(13, OUTPUT);
    pinMode(8, OUTPUT);
    pinMode(9, OUTPUT);
}
void loop() {
 while (BT.available()){
  delay(10);
  char c = BT.read();
  state += c;
 }
if (state.length() > 0) {
  Serial.println(state);
if (state == "acender")
{
  digitalWrite(8,HIGH);
  }
```

```
else if(state == "apagar")
{
  digitalWrite(8, LOW);
}
else if(state == "abrir")
{
  digitalWrite(9,HIGH);
}
else if(state == "fechar")
{
  digitalWrite(9,LOW);
}
else if(state == "ligar")
{
  digitalWrite(13,HIGH);
}
else if(state == "desligar")
{
  digitalWrite(13,LOW);
}
}
state = "";}
```

ANEXO C – FLUXOGRAMA TOTAL DO PROGRAMA

Figura 36: Fluxograma completo do Projeto.



Fonte: Próprio Autor