UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ENGENHERIA DE SÃO CARLOS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Sistema de Descrição de Imagens com Comandos por Voz para Auxílio a Deficientes Visuais

Autor: Arian Fernandes Bertonha

Orientador: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2018

Arian Fernandes Bertonha

Sistema de Descrição de Imagens com Comandos por Voz para Auxílio a Deficientes Visuais

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Carlos

Curso de Engenharia Elétrica – Ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luís Linhari Rodrigues

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Fernandes Bertonha, Arian

F547s Sistema de Descrição de Imagens com Comandos por Voz para Auxílio a Deficientes Visuais / Arian Fernandes Bertonha; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2018.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. Raspberry Pi. 2. computação em nuvem. 3. comandos por voz. 4. classificação de imagens. 5. tecnologia assistiva. 6. deficientes visuais. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Arian Fernandes Bertonha

Titulo: "Sistema de descrição de imagens com comandos por voz para auxilio a deficientes visuais"

Trabalho de Conclusão de Curso defendido e aprovado em 20/06/19,

com NOTA 10,0 (DEZ, ZERO), pela Comissão Julgadora:

Prof. Associado Evandro Luis Linhari Rodrigues - Orientador - SEL/EESC/USP

Prof. Associado Adilson Gonzaga - SEL/EESC/USP

Dr. Alex Antonio Affonso - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

DEDICATÓRIA

Este trabalho de conclusão de curso é dedicado à minha mãe, ao meu pai, minha irmã, minha noiva e toda a minha família, que sempre estiveram comigo.

Arian Fernandes Bertonha

AGRADECIMENTOS

Quero agradecer primeiramente à Deus, e agradeço à minha família, por acreditar em mim e sempre me apoiar.

Mãe, Tania Mara Fernandes Bertonha, seu cuidado e dedicação me deram a esperança para seguir em frente.

Pai, Laerte João Bertonha, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada, um exemplo a ser seguido.

Minha irmã, Ariane Fernandes Bertonha Delezuk, pela amizade, companheirismo e amor.

À Ariane Carolina da Rocha, minha noiva, a pessoa que escolhi para partilhar minha vida. Obrigado pelo carinho, a paciência e por sua capacidade de me trazer paz na correria de cada semestre.

Ao professor Evandro Luís Linhari Rodrigues, um professor que admiro, que me ensinou boa parte do que sei, pela sua paciência na orientação e incentivo que tornaram possível a conclusão desta monografia.

À Universidade de São Paulo, por abrir as portas para minha formação, onde eu pude ter as bases para desenvolver o presente projeto.

A todos aqueles que de alguma forma estiveram e estão próximos a mim, fazendo valer a pena essa conquista.

Arian Fernandes Bertonha

“A percepção do desconhecido é a mais fascinante das experiências. O homem que não tem os olhos abertos para o misterioso passará pela vida sem ver nada.”

Albert Einstein

RESUMO

Atualmente o Brasil está com mais de 200 milhões de habitantes, e sua população que possui qualquer tipo de deficiencia ultrapassa os 22%. Existindo 1,1 milhão de cegos e cerca de 4 milhões de deficientes visuais sérios. A tecnologia tem avançado com o passar do tempo, em 2016 as estimativas apontaram que 4 em cada 10 pessoas já possuíam acesso à Internet. O presente projeto propõe o desenvolvimento de um sistema acessível e portátil que possa descrever o ambiente e seja controlado a partir de comandos por voz para o auxilio de pessoas que possuem deficiência visual, utilizando uma Raspberry Pi. A funcionalidade de detecção de imagens mostrou-se eficiente, assim como os comandos por voz, apesar da necessidade constante de conexão com a internet, demonstrando ser promissor nessa área de aplicação pouco desenvolvida.

Palavras-Chave: Raspberry Pi, computação em nuvem, comandos por voz, classificação de imagens, tecnologia assistiva, deficientes visuais.

ABSTRACT

Currently Brazil has more than 200 million inhabitants, and its population that has any type of deficiency exceeds 22%. There were 1.1 million blind people and about 4 million visually impaired people. Technology has progressed over time, by 2016 estimates indicated that 4 out of 10 people already had access to the Internet. The present project proposes the development of an accessible and portable system that can describe the environment and be controlled from voice commands to the aid of people who are visually impaired using a Raspberry Pi. The image detection functionality proved to be efficient, as well as the voice commands, in spite of the constant need to connect to the internet, proving to be promising in this area of poorly developed application.

Keywords: Raspberry Pi, voice commands, cloud computing, image classification, assistive technology, visually impaired.

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS

ITU	International Telecommunication Union – União Internacional de Telecomunicações
OMS	Organização Mundial da Saúde
AT	Assistive Technology – Tecnologia Assistiva
NIST	National Institute of Standards and Technology – Instituto Nacional de Padrões e Tecnologia dos EUA
IOT	Internet of Things – Internet das Coisas
API	Application Programming Interface – Interface de Programação de Aplicativos
SR	Speech Recognition – Sistema de Reconhecimento de Fala
ASR	Automatic Speech Recognition – Reconhecimento Automático de Fala
HMM	Hidden Markov Model – Modelo Oculto de Markov
PCM	Pulse Code Modulation – Modulador de Código de Pulso
LVCSR	Large Vocabulary Continuous Speech Recognition – Reconhecimento Contínuo de Fala para Vocabulário Extenso
LM	Language Model – Modelos de Linguagem
STT	Speech-To-Text – Fala para Texto
TTS	Text-To-Speech – Texto para Fala
ML	Machine Learning – Aprendizado de Máquina
NPL	Natural Language Processing – Processamento de Linguagem Natural
BOW	Bag of Words – Saco de Palavras
CV	Computational Vision – Visão Computacional
OCR	Optical Character Recognition – Reconhecimento Ótico de Caracter
REST	Representational State Transfer – Transferência de Estado Representacional
SBC	Single Board Computer – Computador de Placa Única
ARM	Acorn RISC Machine
NLTK	Natural Language Toolkit – Kit de Ferramentas para Linguagem Natural
ORD-BMS	Object Relational Database Management System – Sistema de Gerenciamento de Banco de Dados Orientado a Objeto
DNS	Domain Name Service – Serviços de Nome de Domínio

TCP	Transmission Control Protocol – Protocolo de Controle de Transmissão
UDP	User Datagram Protocol – Protocolo de Datagrama do Usuário
GCSR	Google Cloud Speech Recognition
GCV	Google Cloud Vision

SUMÁRIO

1. INTRODUÇÃO

Em 2016 o Banco Mundial publicou seu relatório anual sobre o Desenvolvimento Mundial: Dividendos Digitais cujo tema principal foi a relação entre os avanços tecnológicos e a inclusão digital. Os resultados mostram que o desenvolvimento tecnológico pode facilitar a participação social, econômica e cívica de pessoas com algum tipo de deficiência. Mas para que isso seja possível é imperativo que as novas tecnologias se tornem acessíveis, caso contrário, os novos desenvolvimentos se tornarão uma outra fonte de marginalização e exclusão para pessoas com deficiência [1]. Esse capítulo faz uma introdução ao leitor do cenário tecnológico atual, do panorama mundial de pessoas com deficiência visual, as motivações e objetivos do desenvolvimento de uma aplicação voltada à engenharia assistiva para este tipo de deficiência, e, por fim, irá propor um sistema que aplique tecnologias modernas para facilitar a vida deste público-alvo.

Desde os primórdios o ser humano vive em constante adaptação, buscando novas tecnologias que possam auxiliar seu desenvolvimento e proporcionar facilidades e conforto. Existem cada vez mais aparelhos eletrônicos para facilitar as tarefas cotidianas. Em meados de 2016 os aparelhos celulares atingiram a marca de um dispositivo por pessoa no mundo, e 4 em cada 10 pessoas já possuíam acesso à Internet [1] [2]. Além disso, de acordo com a União Internacional de Telecomunicações (ITU, *International Telecommunication Union*) 85% da população mundial já tem acesso à eletricidade [3]. É inegável que com "o aumento do número de dispositivos eletrônicos, combinado com a conectividade e o valor informativo da Internet, as tecnologias digitais estão assumindo um papel fundamental no cotidiano das pessoas" [1].

Porém, esta revolução não se encontra disponível para todos. Embora a tecnologia atual possibilite meios alternativos de comunicação, como interfaces controladas por gestos e reconhecimento por voz, pessoas com deficiência viram poucos e tardios avanços para a sua inclusão. O desenvolvimento de tecnologias assistivas nem sempre é acessível aos usuários que a necessitam, sua realização exige esforços ativos para realinhar e moldar os fatores sociais, econômicos, pessoais e de infraestrutura [1].

O Brasil, com mais de 200 milhões de habitantes, possui uma população de pessoas com deficiência que ultrapassa os 22%. De acordo com a Organização Mundial de Saúde, 10% da população tem algum tipo de deficiência e 80% vivem em países em desenvolvimento. Na América Latina e Caribe esse número corresponderia a 85 milhões de pessoas [4].

Mais especificamente, pesquisas realizadas pela Organização Mundial da Saúde (OMS) indicam que, se iniciativas de alcance mundial e regional não forem tomadas, em 2020 existirão no mundo 75 milhões de pessoas cegas e mais de 225 milhões de portadores de baixa visão, ou seja, incapazes de desempenhar grande número de tarefas do dia a dia, devido à deficiência visual. No Brasil, os cálculos apontam para a existência de 1,1 milhão de cegos (0,6% da população estimada) e cerca de 4 milhões de deficientes visuais sérios [4].

1.1. Motivação

A primeira motivação para a elaboração deste Trabalho de Conclusão de Curso é o aprendizado de uma linguagem de programação poderosa, versátil e representativa no cenário atual, como Python, que atualmente se faz necessário a um profissional de engenharia elétrica. A segunda é o estudo de métodos de programação e aplicações que estão trilhando os avanços tecnológicos. A terceira é o desenvolvimento de uma aplicação voltada a engenharia assistiva, contribuindo, assim, para facilitar a vida de pessoas com deficiência visual e consequentemente promover independência e inclusão.

1.2. Objetivo

Pensando nisso, o presente projeto se propõe a desenvolver um sistema acessível e portátil que possa descrever o ambiente e seja controlado a partir de comandos por voz para o auxílio de pessoas com deficiências visuais.

1.3. Justificativa

Levando-se em consideração as tendências tecnológicas nesta área de aplicação da engenharia assistiva, foi natural seguir o movimento de projetos e pesquisas já desenvolvidos que se tornaram referência no atual estado da arte.

Existe uma necessidade significativa de projetos de sistemas que utilizem as tecnologias disponíveis no contexto atual de desenvolvimento para beneficiar as pessoas com deficiência, gerando soluções de acessibilidade de baixo custo.

1.4. Organização do Trabalho

No presente trabalho serão apresentadas, nas seções a seguir, as principais soluções voltadas para o auxílio a pessoas com deficiência visual, as ferramentas atualmente utilizadas para o desenvolvimento de um sistema de reconhecimento por voz, ferramentas para descrição de imagens, bem como os materiais e métodos utilizados na montagem do sistema. Mais especificamente, serão demonstrados os recursos de ambiente utilizados, como a Raspberry Pi 3, o sistema operacional Raspbian, os motores de tradução de texto para voz e de voz para texto, o classificador de imagens, o reconhecedor de interlocutor e os métodos de classificação de texto. A partir dos resultados será analisado o sistema implementado, apresentando seu desempenho. Por fim, uma breve conclusão será apresentada, com o propósito de referenciar os objetivos alcançados no decorrer da evolução deste projeto, além de dissertar sobre os próximos passos no desenvolvimento do presente projeto.

2. EMBASAMENTO TEÓRICO

Antes de entender o funcionamento e a integração do sistema, é de extrema importância o estudo dos principais conceitos que fundamentam a execução do projeto. As seções contidas neste capítulo discorrem sobre os conceitos utilizados no processo de desenvolvimento do projeto: conceito de tecnologia assistiva e dispositivo '*wearable*', reconhecimento da fala, síntese de voz, reconhecimento de interlocutor, interpretação de comandos através da classificação de textos, classificação de imagens e síntese de voz. Estes fundamentos serão utilizados como suporte para compreensão dos próximos capítulos.

2.1. Tecnologia Assistiva

Este projeto propõe a implementação de um sistema com o conceito de tecnologia assistiva, que é definido de acordo com o Comitê de Ajudas Técnicas [5], instituído pela PORTARIA N° 142, DE 16 DE NOVEMBRO DE 2006: "Tecnologia Assistiva é uma área do conhecimento, de característica interdisciplinar, que engloba produtos, recursos, metodologias, estratégias, práticas e serviços que objetivam promover a funcionalidade, relacionada à atividade e participação de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social".

Figura 2.1: Tecnologia assistiva empregada no sistema eSight.
Fonte: https://www.esighteyewear.com/

A tecnologia assistiva (*Assistive Technology* - AT) descreve qualquer dispositivo ou outra solução usada por pessoas com deficiências para auxiliar na execução de tarefas ou atividades. Embora potencialmente útil, a disponibilidade de AT apropriada para pessoas com deficiências pode ser problemática devido a altos custos, fontes de financiamento limitadas e falta de conhecimento sobre quais soluções podem existir para suas necessidades [1].

2.2. Dispositivos Wearable

A tecnologia *'wearable'* é um termo genérico para dispositivos eletrônicos que podem ser usados no corpo, seja como acessórios ou como parte de uma roupa. Existem muitos tipos de tecnologia *'wearable'*, mas alguns dos dispositivos mais populares são rastreadores de atividade e *'smartwatches'*. Uma das principais características da tecnologia *'wearable'* é a sua capacidade de se conectar à Internet, permitindo que dados sejam trocados entre uma rede e o dispositivo [6]. Um dos requisitos deste projeto é a implementação de um dispositivo prático que garanta sua usabilidade no dia a dia de pessoas com deficiência visual, com o conceito "*wearable*". A **Figura 2.1** apresenta um dispositivo *'wearable'* utilizado por pessoas com certo grau de deficiência visual.

2.3. Computação em Nuvem

O Instituto Nacional de Padrões e Tecnologia dos EUA (*National Institute of Standards and Technology* - NIST) define a computação em nuvem como "um modelo que permite acesso sob demanda, via rede, a um conjunto compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e liberados, com esforço mínimo de gerenciamento ou interação com o provedor de serviços" [7]. A definição lista cinco características essenciais da computação em nuvem: autoatendimento sob demanda, amplo acesso à rede, compartilhamento de recursos, escalabilidade e serviço medido por quantidade de acessos ou dados transferidos. O NIST também lista três "modelos de serviço" (*'software'*, plataforma e infraestrutura) e quatro "modelos de acesso" (privado, comunitário, público e híbrido) que juntos categorizam as formas de fornecer serviços em nuvem [7].

Em suma, o conceito de computação em nuvem (em inglês, *Cloud Computing*) refere-se à utilização da memória, da capacidade de armazenamento, do cálculo de computadores e servidores compartilhados, interligados por meio da Internet, seguindo o princípio da computação em grade [8].

O armazenamento ou processamento de dados é feito em serviços que poderão ser acessados de qualquer lugar do mundo, a qualquer hora, não havendo necessidade de instalação de programas. O acesso a programas, serviços e arquivos é remoto, através da Internet - daí a alusão à nuvem [8]. O uso desse modelo (ambiente) em muitos projetos é mais viável do que o uso de unidades físicas, como em aplicações IOT que não dispõem de hardwares tão poderosos para executar todo o processamento [8].

A Google Cloud API é uma REST API que emprega o conceito de computação em nuvem e oferece soluções modernas para problemas como o reconhecimento de fala, a classificação de imagens e síntese de voz, que os motores de processamento de comandos por voz do Android utilizam. Seu uso é limitado, mas inclui pacotes gratuitos que atendem aos requisitos funcionais deste projeto.

Figura 2.2: Diagrama ilustrativo da computação em nuvem.
Fonte: https://pt.wikipedia.org/wiki/Computa%C3%A7%C3%A3o_em_nuvem

2.4. Estrutura do Diálogo

O diálogo, através de um determinado padrão linguístico é a forma de comunicação mais utilizada entre os seres humanos [9]. As duas ações mais importantes de um diálogo são a emissão e recepção de enunciados que contêm informações. Para que haja comunicação é necessário que o destinatário da informação a receba e compreenda. Neste contexto, o comunicador tem que ser claro durante a transmissão da mensagem principalmente para evitar a confusão de conceitos e ideias. Assim, fica evidente a importância do entendimento da estrutura do diálogo para o estudo desta monografia.

Graças à capacidade do cérebro humano de interpretar informações extremamente complexas, os seres humanos captam facilmente, em uma mensagem falada, várias informações que sugerem que o discurso é construído com palavras, e cada palavra é composta por fonemas. Porém, a fala é um processo dinâmico sem partes claramente distintas. Por isso, o processo de representação fonética não é determinístico; muitas variáveis devem ser levadas em conta antes de se inferir o que foi dito [10]. Os computadores ainda não têm o potencial de reconhecer através da fala todas as informações como nós seres humanos fazemos. Todas as descrições modernas da fala são, em certa medida, probabilísticas. Isso significa que não existem limites certos entre unidades fonéticas ou entre palavras.

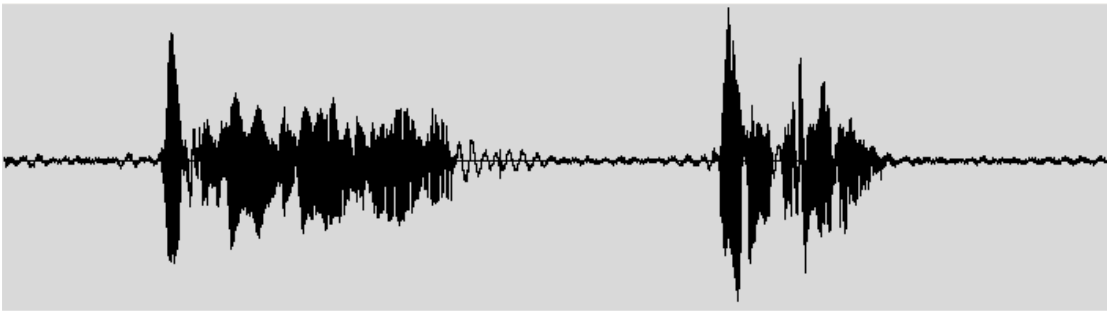

Figura 2.3: Exemplo da forma de onda da fala.

A partir do exemplo presente na **Figura 2.3**, observa-se que o diálogo é um fluxo contínuo de áudio em que os estados estáveis se misturam com estados dinamicamente alterados. Nesta sequência de estados, pode-se definir classes de sons ou fonemas mais ou menos semelhantes. As propriedades acústicas de uma forma de onda correspondente a um fonema podem variar bastante dependendo de muitos fatores: contexto do fonema, tipo de voz, sotaque e assim por diante.

Isso faz com que os fonemas sejam diferentes da sua representação "canônica". Assim, as transições entre as palavras podem ser mais informativas do que regiões estáveis, ou seja, muitas vezes é mais proveitoso analisar subseções dos fonemas e observar como sua plasticidade responde ao contexto da enunciação. Essa divisão dos fonemas é chamada de senones - palavra do inglês que significa secção de um fonema em “n” partes [9].

2.5. Sistemas de Reconhecimento de Fala

O sistema de reconhecimento de fala (*Speech Recognition* - SR) é a tecnologia que permite o reconhecimento e a tradução da linguagem falada em texto. É também conhecido como "reconhecimento automático de fala" (*Automatic Speech Recognition* - ASR), “reconhecimento de fala do computador” ou apenas “fala para texto” (*Speech-To-Text* - STT).

2.6. Sistema de Fala para Texto

O motor de voz para texto (STT) é um tipo de *‘software’* que efetivamente transcreve conteúdo de áudio em palavras escritas. Os softwares STT vêm melhorando sua precisão e evoluindo em funcionalidades gerais para desempenhar um papel maior nas comunicações modernas em plataformas digitais. Tanto o modelo acústico e quanto o modelo de linguagem são partes importantes de algoritmos modernos de reconhecimento de fala com base estatística. O modelo Oculto de Markov (*Hidden Markov Model* - HMM) é amplamente utilizado em muitos sistemas.

O processo de reconhecimento de fala basicamente permitiu que o ser humano fale com o computador. Este processo funciona fundamentalmente como um pipeline que converte o áudio digital do Modulador de Código de Pulso (*Pulse Code Modulation* - PCM) de uma placa de som para um discurso reconhecido. A forma de onda obtida através da fala é dividida em pequenas amostras separadas por intervalos de silêncio, e então um algoritmo de reconhecimento faz a estimativa do que está sendo dito em cada amostra obtida. Para fazer isso os softwares selecionam todas as combinações possíveis de palavras e tenta combiná-las comparando-as com o áudio. O diagrama abaixo exemplifica o funcionamento de um STT, que contém unidade de extração, unidade de modelo acústico, unidade de modelo de linguagem e unidade de pesquisa ou decodificadora.

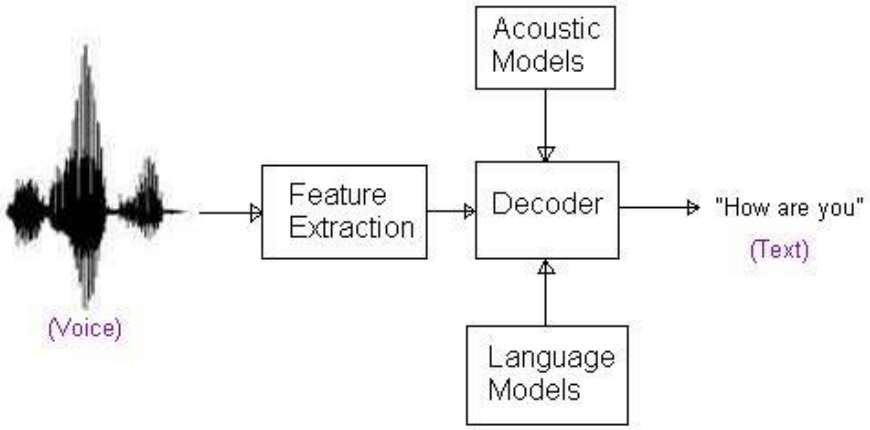

Figura 2.4: Funcionamento de um STT.
Fonte: http://www.rfwireless-world.com/Terminology/automatic-speech-recognition-system.html

2.7. Modelo Acústico

O modelo acústico gera um modelo matemático, a partir das características extraídas da fala, que representa um sinal de fala original. Isso ocorre de tal forma que os segmentos da fala desconhecidos possam ser mapeados de modo correto, determinando a palavra correspondente. O modelo acústico representa a sua pronúncia em um formato legível para a máquina. O modelo acústico usa o fato de as palavras faladas serem compostas de sons, assim como as palavras escritas serem compostos por letras. Usando esse conhecimento, é possível segmentar as palavras em sons (representados pela pronúncia) e agrupá-los de novo no reconhecimento. Estes blocos de construção chamam-se “fonemas”.

2.8. Modelo de Linguagem

É utilizado principalmente em sistemas de reconhecimento contínuo para grandes vocabulários que possuem uma extensa gramática, comumente conhecido na literatura como LVCSR (*Large Vocabulary Continuous Speech Recognition*). Utilizar informações acústicas não é a maneira mais fácil de obter um bom desempenho do sistema, pois as palavras não são necessariamente ditas de forma isolada. Uma palavra poderia ser seguida por qualquer outra o que dificultaria o trabalho do reconhecedor. Parte desse problema é resolvido utilizando-se Modelos de Linguagem (*Language Model* - LM). O LM tem como objetivo estimar de forma mais confiável possível a probabilidade de ocorrência de uma determinada sequência de palavras.

Na criação de modelos de linguagem são várias as características que os diferenciam, tais como: vocabulário, definido pela quantidade de palavras existentes no modelo de linguagem e tipo de dados utilizados para treino. Como o modelo de linguagem busca facilitar o trabalho do decodificador, restringindo o número de possíveis palavras que podem seguir uma dada palavra, o mesmo influencia diretamente no seu desempenho.

2.9. Sistema de Texto para Fala

O sistema Texto Para Fala (*Text-To-Speech* - TTS), é uma forma de síntese de fala que converte texto em saída de voz falada. O sistema TTS foi desenvolvido pela primeira vez para auxiliar os deficientes visuais, oferecendo uma voz falada gerada por computador capaz de "ler" o texto para o usuário. Os sistemas TTS formam frases baseadas em grafemas e fonemas de um idioma, utilizando um dicionário fonético e um modelo probabilístico, assim sempre pronunciará um determinado conjunto de fonemas da mesma forma.

Figura 2.5: Funcionamento de um TTS.
Fonte: https://www.igorsramos.com/watson-text-to-speech-caching/

2.10. Distância de Levenshtein

A distância Levenshtein ou distância de edição entre duas *'strings'* (duas sequências de caracteres) é dada pelo número mínimo de operações necessárias para transformar uma *'string'* em outra. Entendemos por "operações" a inserção, deleção ou substituição de um carácter. É muito útil para aplicações que precisam determinar quão semelhantes duas *'strings'* são, como é por exemplo o caso com os verificadores ortográficos.

2.11. Reconhecimento de Interlocutor

O Reconhecimento de Interlocutor é a identificação de uma pessoa a partir de características da voz (biometria de voz) [11]. Existe uma diferença entre o reconhecimento de interlocutor (reconhecendo quem está falando) e o reconhecimento de fala (reconhecendo o que está sendo dito).

Reconhecer o interlocutor pode simplificar a tarefa de traduzir a fala em sistemas que foram treinados em vozes de pessoas específicas ou pode ser usado para autenticar ou verificar a identidade de um interlocutor como parte de um processo de segurança.

2.12. Aprendizado de Máquina

O Aprendizado de Máquina (*Machine Learning* - ML) ou aprendizagem automática é um subcampo da ciência da computação [12] que evoluiu do estudo de reconhecimento de padrões e da teoria do aprendizado computacional em inteligência artificial [12]. Em 1959, Arthur Samuel definiu aprendizado de máquina como o "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados" [13]. O aprendizado automático explora o estudo e construção de algoritmos que podem aprender de seus erros e fazer previsões sobre dados [14]. Tais algoritmos operam construindo um modelo a partir de entradas amostrais a fim de fazer previsões ou decisões guiadas pelos dados em vez de simplesmente seguindo inflexíveis e estáticas instruções programadas.

2.13. Processamento de Linguagem Natural

O Processamento de Linguagem Natural é uma gama teoricamente motivada de técnicas computacionais para analisar e representar textos que ocorrem naturalmente em um ou mais níveis de análise linguística com a finalidade de obter processamento de linguagem semelhante à de um ser humano.

O Processamento de Linguagem Natural (*Natural Language Processing* - NLP) é uma vasta área da ciência da computação que se preocupa com a interação entre computadores e linguagem humana. A modelagem de idiomas é uma tarefa fundamental na inteligência artificial e na NLP. Um modelo de linguagem é formalizado como uma distribuição de probabilidade sobre uma sequência de palavras. Recentemente, modelos de aprendizagem profunda alcançaram resultados notáveis em reconhecimento de fala [15] e visão computacional [16].

A classificação de texto desempenha um papel importante em muitos aplicativos que realizam NLP, como filtragem de spam, categorização de e-mail, recuperação de informações, pesquisa na web e classificação de documentos [17] [18], nos quais é necessário atribuir categorias predefinidas a uma sequência de texto.

Um método popular e comum para representar textos é o saco de palavras (*Bag Of Words* - BOW). No entanto, o método *'Bag of Words'* perde a ordem das palavras e ignora a semântica. Para o simples fim de categorizar um comando de voz, neste trabalho será utilizado o método BOW.

2.14. Visão Computacional

Visão computacional (*Computational Vision* - CV) representa um campo de pesquisa complexo que apresentou um enorme desenvolvimento nas últimas décadas. A Visão Computacional pode ser definida como a ciência e tecnologia das máquinas capazes de coletar e analisar imagens (ou vídeos) com o objetivo de extrair informações dos dados visuais processados. Por essas razões, a CV explora vários métodos compartilhados com campos de pesquisa de processamento de sinais, e seus algoritmos fazem uso de disciplinas matemáticas e de engenharia, como óptica, geometria, teoria da probabilidade, otimização de estatísticas, física e biologia.

Como as câmeras estão se tornando hardware de computador padrão e recurso necessário de dispositivos móveis, a CV está passando de uma ferramenta de nicho para uma ferramenta cada vez mais comum para uma ampla gama de aplicações. Alguns deles incluem controle visual (robôs industriais ou veículos autônomos), detecção de eventos (pessoas ou contagem de objetos), análise de imagens (reconhecimento facial, inspeção industrial ou análise de imagens médicas), gerenciamento de informações (indexação de bancos de dados) e entrada de dados para dispositivos de interação humano-computador. A CV também é uma ferramenta poderosa para o desenvolvimento de soluções de tecnologia assistiva (AT) para pessoas com deficiências visuais. Os aplicativos baseados em CV podem melhorar a mobilidade e a orientação das pessoas, bem como o reconhecimento de objetos, o acesso a serviços locais e a interação social.

2.15. Reconhecimento Óptico de Caractere

O reconhecimento óptico de caractere (*Optical Character Recognition* - OCR) é uma tecnologia que permite reconhecer automaticamente caracteres através de um mecanismo óptico. No caso dos seres humanos, nossos olhos são mecanismos ópticos. A imagem vista pelos olhos é uma entrada de dados para o cérebro.

A capacidade de entender esses dados visuais varia em cada pessoa de acordo com muitos fatores [19]. O OCR é uma tecnologia que funciona como a capacidade humana de leitura.

O OCR pode reconhecer texto manuscrito e impresso. Mas o seu desempenho é diretamente dependente da qualidade das imagens de entrada. Esta tecnologia é projetada para processar imagens que consistem quase inteiramente em texto, com pouca interferência não textual obtida da imagem capturada pela câmera móvel.

Ao aplicar uma imagem com uma página de texto, por exemplo, como entrada para um sistema que possua tal função, ocorre primeiramente uma confirmação da orientação do texto, seguida de uma segmentação em preto e branco dos pixels, uma divisão em linhas de texto, e por último em símbolos individuais. Ao fim desse processo, um algoritmo treinado para reconhecer tal símbolo é aplicado. Se o sistema tiver sido previamente treinado para reconhecer o elemento em questão, calcula-se uma probabilidade de sua interpretação estar correta. Os símbolos reconhecidos são, então, agrupados em palavras e em sentenças e a informação completa é retornada em ordem [19].

2.16. Classificação de Imagens

Para que um sistema seja capaz de classificar uma imagem, e posteriormente promover uma descrição, que é o caso deste projeto, é necessário que o mesmo esteja previamente treinado com imagens. Um processo como esse, envolve busca e comparação de imagens. J.R. Parker [19] sugere em seu livro um método para se realizar buscas de imagens, inserindo imagens como entrada.

Primeiramente, assume-se que existe um conjunto de imagens, previamente rotuladas e devidamente agrupadas de acordo com seu conteúdo. Então, o que ocorre é uma análise computacional da imagem para extração de dados em busca de padrões. Esses dados são comparados com os dados das imagens cujas características já são conhecidas e baseado em seu nível de similaridade, ocorre o agrupamento da imagem que é classificada de forma a refletir seu conteúdo.

Vários padrões podem ser extraídos de uma imagem para servir de critério de comparação, como a coloração. Através da contagem de pixels com cada cor presente na imagem é possível criar um histograma da distribuição dessas cores.

A comparação pode ser, então, realizada sobre a similaridade entre histogramas, como por exemplo os histogramas de padrão binário local.

2.17. REST API

REST significa *'Representational State Transfer'*. Em português, Transferência de Estado Representacional, trata-se de uma abstração da arquitetura Web. Resumidamente, o REST consiste em princípios/regras/*'constraints'* que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas, desta forma, as aplicações são capazes de se comunicar de forma padronizada com esforço mínimo.

2.18. Estado da Arte

Alguns projetos foram utilizados para basear o desenvolvimento do presente projeto. Dessa forma, o projeto procurou seguir a linha de produtos bem-sucedidos no mercado na área de auxílio a pessoas com deficiência visual.

2.18.1. Projetos para Auxílio a Deficientes Visuais

O aplicativo TapTapSee, é um exemplo de projeto que possui funcionalidades semelhantes à proposta feita pelo presente trabalho, fotografando objetos e identificando-os em voz alta. O aplicativo é vinculado com contas de usuário, como Facebook e Twitter para compartilhamento das identificações [20].

Há também o aplicativo móvel Be My Eyes [21], um sistema de auxilio em rede que conecta usuários cegos a voluntários por meio de vídeo chamadas e áudios. Foi realizado uma avaliação do aplicativo por Mauro Avila et al. [22], com 15 homens e 15 mulheres através de redes sociais, com idade entre 36 e 65 anos. Os entrevistados consideraram que o aplicativo útil para leitura de textos, na localização de objetos, assistência a compra, entre outras atividades do dia a dia.

Um produto que merece destaque na área de engenharia assistiva para pessoas que possuem um certo grau de deficiência visual é o eSight, um dispositivo médico *'wearable'* que foi projetado para melhorar a visão de quem possui limitações visuais ou cegueira legal. O empresário Conrad Lewis, engenheiro eletricista canadense, fundou a eSigth em 2006, com suas duas irmãs legalmente cegas.

Sua motivação era construir um dispositivo que fosse capaz de fazer com que suas irmãs conseguissem enxergar, trabalhar e fazer as atividades cotidianas apesar da visão limitada [24] [25].

2.18.2. Cenário no Brasil

No Brasil também há alguns trabalhos que merecem destaque nesta área de aplicação. O aplicativo CPqD Alcance, faz a adaptação de uma grande parte das funcionalidades nativas de um celular em uma interface que apresenta maior usabilidade. Este aplicativo pode ser usado não somente por pessoas com deficiência visual, como também por pessoas idosas e pessoas iletradas.

Na Universidade de São Paulo, há pesquisas voltadas para o auxílio a deficientes visuais que merecem destaque, por exemplo o projeto GuideMe [24], coordenado pelo professor Francisco Monaco, do ICMC-USP. Este sistema consiste em um dispositivo pequeno ajustável à roupa contendo uma câmera e sensores de ultrassom com o intuito de identificar rostos de pessoas conhecidas, por meio de algoritmos de processamento de imagem, além de reconhecer obstáculos em um percurso, que para avisar ao usuário, emite sons cuja a intensidade é baseada na direção e proximidade em relação aos obstáculos.

3. PLANEJAMENTO DO PROJETO

Antes de apresentar a implementação do sistema, é importante discorrer sobre seu planejamento. Vale ressaltar, que as atividades de um projeto precisam ser calcadas de um planejamento realista e factível, que contemple os objetivos, metas, restrições e recursos envolvidos. Contudo, não deve ser encarado "como trilho, mas sim como trilha", auxiliando na redução de incertezas e na previsão de ações corretivas.

Este capítulo será destinado a apresentar a análise de requisitos e o planejamento de testes do sistema, que são fundamentais no processo de desenvolvimento de um *'software'*.

3.1. Análise de Requisitos

Definir o escopo do projeto é uma tarefa importante durante o desenvolvimento de um sistema. Nesta seção serão apresentados os requisitos funcionais e não-funcionais.

3.1.1. Requisitos Funcionais

Um requisito funcional define uma função de um sistema de *'software'* ou seu componente. O requisito funcional representa o quê o *'software'* faz, em termos de tarefas e serviços.

- O sistema deve permitir que o usuário execute comandos de voz para operar suas funcionalidades.
- O sistema deve oferecer ao usuário as funcionalidades de extração de informações visuais de textos impressos ou escritos à mão, características visuais de objetos e do ambiente ao seu redor.
- O sistema deve registrar todas as informações extraídas.
- O sistema deve retornar ao usuário dados em forma de som e sintetizados em voz.
- O sistema deve ser capaz de estabelecer conexão contínua com a internet através de um *'hotspot'*.

3.1.2. Requisitos Não-Funcionais

São os requisitos relacionados ao uso da aplicação em termos de desempenho, usabilidade. Estes requisitos dizem respeito a como as funcionalidades serão entregues ao usuário do *'software'*.

- O sistema deve ser acessível, não ultrapassando o valor R$ 500,00.
- O sistema deve garantir sua usabilidade por pessoas com deficiência visual.
- O sistema deve possuir uma fonte de energia que o atenda ininterruptamente durante no mínimo 1 hora.
- O sistema deve garantir a possibilidade de utilização de suas funções ininterruptamente, respeitando os possíveis limites da fonte de energia.
- O sistema deve exibir resultados de extração de informações com confiabilidade acima de 80%.
- O sistema deve possuir uma latência máxima de 10 segundos, para garantir a sua usabilidade.
- O sistema deve constituir um dispositivo portátil.

3.1.3. Planejamento de Testes

É a base para o início de qualquer teste de *'software'*.

- É importante considerar a realização dos seguintes testes a fim de garantir a funcionalidade e performance do sistema.
- Verificar a acurácia do reconhecimento de voz do Google Cloud Speech API.
- Avaliar a capacidade de o sistema responder a uma palavra-chave para os comandos por voz.
- Verificar a funcionalidade dos comandos de voz.
- Verificar a capacidade do sistema de detectar um interlocutor.

- Verificar a relação entre resolução de captura de imagens textuais e o tempo entre solicitação e retorno do resultado, para definir a resolução adequada que retorna resultados confiáveis que possuam acerto superior a 80%, no menor tempo.
- Realizar testes quanto à extração de textos danificados, rotacionados ou escritos à mão em letra de forma e cursiva.
- Medir o consumo de energia do sistema para validar o uso da fonte de energia escolhida.
- Percorrer todas as funcionalidades do sistema para verificar simultaneamente seu funcionamento e o atendimento aos requisitos.

4. MATERIAL E MÉTODOS

Neste capítulo serão descritos e especificados os hardwares e softwares empregados e a forma como foram utilizados para implementar o sistema.

Na execução foram utilizados o Google Cloud Speech API como STT, o Google Cloud Text-To-Speech API como TTS, o Google Cloud Vision API como classificador de imagens. Além disso, foram utilizadas as bibliotecas em Python: PyAudio e Speech Recognition para a aplicação STT e PiWho como reconhecedor de interlocutor. Para implementar a RESTful API com PostgreSQL como banco de dados utilizou-se Flask, Flask-Restful, SQLAlchemy e Marshmallow. Outra biblioteca em Python utilizada foi Natural Language Toolkit para o classificador de texto. Para a realização de testes do sistema, foi utilizado a biblioteca Python Levenshtein para calcular as distâncias das strings faladas e reconhecidas. Para a captura de imagens com a webcam, utilizou-se o pacote fswebcam, e para a reprodução de áudio omxplayer.

Com exceção das APIs da Google Cloud que exigem alguns dados bancários para utilização, mesmo que seja em sua versão gratuita, devido às políticas de franquia de dados da Google, os softwares escolhidos são *'open-source'*. É necessária conexão constante com a internet para a realização do processo de fala para texto (STT), texto para fala (TTS) e classificação de imagens que são realizados em nuvem (Google Cloud Computing). Como plataforma foi utilizada o '*Single Board Computer'* Raspberry Pi 3, com distribuição própria Raspbian baseada no OS Debian. Além disso, foi utilizada uma Webcam USB com microfone integrado fones de ouvido para saída de áudio analógica, um Powerbank e um cabo USB/micro USB para alimentação.

4.1. Material

Nesta seção serão discutidas as escolhas de ambiente, *'hardware'* e *'software'* do sistema.

4.1.1. Raspberry PI 3 E Raspbian

A plataforma escolhida para a implementação do projeto foi uma Raspberry Pi 3 (**Figura 4.1**), que consiste em um *'Single Board Computer'* (SBC) compacto, baseado em ARM, criado pela Fundação Raspberry Pi, com uma documentação ampla e comunidade ativa.

O minicomputador executa o sistema operacional GNU/Linux baseado em Debian denominado Raspbian. Um sistema operacional é o conjunto de programas básicos e utilitários que fazem o *'hardware'* funcionar, no entanto, este sistema operacional utiliza softwares livres e de código aberto e fornece mais do que um sistema operacional puro: contendo mais de 35.000 pacotes e *'software'* pré-compilados. Por ser gratuito e otimizado para o hardware da Raspberry Pi, o Raspbian foi a escolha natural de OS para a plataforma, utilizou-se a versão Raspbian Stretch Lite 4.14.

Figura 4.1: Raspberry Pi 3 modelo B.
Fonte: https://www.raspberrypi.org

A versão 3 da Raspberry Pi possui suporte WiFi/Bluetooth integrado e um processador de 64 bits. Seu processador, quando comparado com outros SBCs, possui um bom desempenho, constituído de uma CPU de núcleo quádruplo de 1.2GHz, ARMv8. As especificações da Raspberry Pi 3 podem ser vistas na **Tabela 4.1**.

Tabela 4.1: Especificações técnicas Raspberry PI 3.

Processador	Broadcom BCM2837
Arquitetura	ARMv8
Núcleo	4 Núcleos Cortex-A53 64-bit
Clock CPU	1.2 GHz
GPU	VideoCore IV
RAM	1 GB
Wireless	802.11n
Bluetooth	4.1
Consumo Médio	800 mA
Preço Médio	R$ 200,00

4.1.2. Powerbank

Figura 4.2: Powerbank de 10000 mAh Kimaster E88.
Fonte: http://kimaster.com.br

Para a implementação de um dispositivo prático é necessário o uso de uma fonte de alimentação portátil, utilizou-se então o *'Powerbank'* de 10000 mAh Kimaster E88 (**Figura 4.2**). As especificações do *'Powerbank'* se encontram na **Tabela 4.2**.

Tabela 4.2: Especificações técnicas Powerbank.

Entrada	5 V/2 A
Saída	5 V / 2,4 A
Capacidade real de carga	3,7 V / 10000 mAh
Preço Médio	R$ 100,00

4.1.3. Webcam USB

Figura 4.3: Webcam USB Logitech C270 HD.
Fonte: https://assets.logitech.com/

Foi utilizada uma *'Webcam'* USB Logitech C270 HD (**Figura 4.3**) para captura de imagem e de som. As especificações da *'Webcam'* USB utilizada encontram-se na **Tabela 4.3**.

Tabela 4.3: Especificações técnicas Webcam USB.

Alimentação	5 V (USB)
Interface	USB 2.0
Resolução Interpolada	3 Megapixels
Resolução de Imagem	1280 x 720
Preço Médio	R$ 80,00

4.1.4. Fone de Ouvido

Para a saída de áudio analógica, utilizou-se um fone de ouvido intra-auricular ASUS, cujas especificações se encontram a seguir na **Tabela 4.4**.

Tabela 4.4: Especificações técnicas Fone de Ouvido.

Potência	20 mW
Impedância	16 Ohms
Frequência	11 a 20000 Hz
Preço Médio	R$ 50,00

4.1.5. Google Cloud Speech API

A API Speech-To-Text do Google Cloud foi utilizada no processo de conversão de fala para texto. Esta ferramenta realiza reconhecimento automático de fala (ASR) através de uma interface RESTful que pode ser acessada por meio de solicitações HTTP POST comuns. A aplicação reconhece 120 idiomas e variantes, incluindo o Português Brasileiro, modelados em uma rede neural de aprendizagem profunda. A API processa streaming em tempo real ou áudio pré-gravado, usando a tecnologia de aprendizado de máquina do Google.

O processamento é realizado fornecendo o arquivo de áudio e a descrição do seu formato através de um POST. A API retorna, então, o texto com melhor correspondência com a precisão do reconhecimento. Opcionalmente, pode ser solicitado o retorno de várias alternativas de reconhecimento além da melhor correspondência, cada uma com a respectiva precisão estimada.

O arquivo a ser reconhecido pode ser fornecido incluindo-se o sinal de áudio no conteúdo da solicitação de HTTP (codificada com Base64) ou fornecendo a URI do arquivo no armazenamento em nuvem da Google. Os formatos suportados são áudio bruto e formato FLAC, MP3 e AAC não são aceitos.

Para melhorar a precisão do sistema, palavras ou frases podem ser anexadas à solicitação como texto. Isso é particularmente útil no caso de sinais de áudio com ruído ou quando palavras incomuns e específicas estão presentes. A plataforma oferece clientes para linguagens de programação comuns (por exemplo, Python, Java, iOS, Node.js), tanto para solicitações em HTTP quanto em tempo real (com respostas assíncronas).

4.1.6. Google Cloud Text-To-Speech API

A API Text-to-Speech do Google Cloud foi utilizada no processo de síntese de voz. A ferramenta permite sintetizar voz natural com 32 vozes diferentes em 12 idiomas e variantes, incluindo o Português Brasileiro. A aplicação utiliza modelos de aprendizado de máquina para gerar vozes que imitam a fala e sons humanos. A API devolve um arquivo de áudio a partir de uma solicitação HTTP em uma interface REST contendo o texto a ser falado. A aplicação permite adicionar pausas, números, formatação de data e hora e outras instruções de pronúncia, ajuste da velocidade de fala que pode ser quatro vezes mais rápida ou lenta que a taxa normal, afinação com 40 semitons diferentes e controle de volume de -96db a 16db.

4.1.7. Google Cloud Vision API

A API Vision do Google Cloud foi utilizada no processo de classificação de imagens. Ela fornece uma interface RESTful que analisa rapidamente o conteúdo da imagem. Essa interface esconde a complexidade dos modelos de aprendizado de máquina e dos algoritmos de processamento de imagem utilizados. Estes modelos definem a precisão geral do sistema, especialmente no que diz respeito à detecção de objetos e estão em constante evolução.

Mais detalhadamente, a API permite classificar imagens a partir dos recursos a seguir:

- **LABEL_DETECTION:** executa a análise de conteúdo em toda a imagem e fornece rótulos relevantes (por exemplo, palavras-chave e categorias);
- **TEXT_DETECTION:** realiza o reconhecimento óptico de caracteres (OCR) e fornece o texto extraído, se houver;
- **FACE_DETECTION:** detecta rostos, fornece probabilidade emocional e afins;
- **LANDMARK_DETECTION:** detecta marcos geográficos;
- **LOGO_DETECTION:** detecta logotipos de empresas e marcas famosas;
- **SAFE_SEARCH_DETECTION:** determina a probabilidade de que uma imagem possa conter violência ou nudez.

A API permite realizar uma única chamada que retorne todos os recursos de uma só vez, com um único upload, porém o resultado é obtido mais rapidamente quando os recursos são executados individualmente. A API aceita imagens codificadas em base64 e imagens presentes na Google Cloud Storage como entrada.

4.1.8. Python

Python é uma linguagem de programação de alto nível, interpretativa e orientada a objetos. Essa linguagem caracteriza-se, especialmente, por ser concisa e clara facilitando sua utilização em diversas aplicações. A linguagem de programação utilizada foi Python versão 3.6.3.

4.1.9. Speech Recognition Library

Speech Recognition é uma biblioteca para Python que realiza reconhecimento de fala, com suporte para vários mecanismos e APIs, on-line e off-line. A biblioteca é capaz de ajustar o nível de ruído ambiente para melhor funcionamento, gravar arquivos de áudio ou enviar o áudio diretamente para os APIs de reconhecimento de fala, no caso do presente projeto a Google Cloud Speech API.

4.1.10. PiWho

PiWho é uma biblioteca gratuita e open source para Python baseada no framework MARF. Foi utilizada neste projeto para o treinamento e reconhecimento de interlocutor. Com esta biblioteca é possível que apenas os interlocutores confiáveis, que possuem permissões, executem comandos no sistema.

4.1.11. Natural Language Toolkit

Para a classificação do texto, obtido a partir do método STT, em um comando foi utilizado o Natural Language Toolkit (NLTK), uma plataforma usada para criar programas para análise de texto. O NLTK destina-se a apoiar a pesquisa e o ensino em Processamento de Linguagem Natural (NLP) ou áreas relacionadas, incluindo linguística empírica, ciência cognitiva, inteligência artificial, recuperação de informação e aprendizado de máquina. A plataforma suporta funcionalidades de classificação, tokenização, *'stemming'*, *'tagging'*, análise e raciocínio semântico.

4.1.12. Flask

O Flask foi o framework escolhido para a implementação da RESTful API que armazenará no banco de dados, os dados gerados pelo sistema. Este microframework web escrito em Python baseado nas bibliotecas WSGI Werkzeug e Jinja2, tem a flexibilidade da linguagem de programação Python e provê um modelo simples para desenvolvimento web. Uma vez importando no Python, o Flask pode ser usado para economizar tempo construindo aplicações web.

É chamado de microframework porque mantêm um núcleo simples, mas extensível. Não há camada de abstração do banco de dados, validação de formulários, ou qualquer outro componente. Para isso, o framework conta com inúmeras bibliotecas de terceiros que provêm as funcionalidades que se deseja obter. Assim, Flask suporta extensões capazes de adicionar tais funcionalidades na aplicação final. A vasta coleção de bibliotecas para frameworks web em Python simplificam o desenvolvimento e tornam a curva de aprendizado mais suave.

4.1.13. PostgreSQL

O PostgreSQL, normalmente chamado de Postgres, é um sistema de gerenciamento de banco de dados do tipo objeto-relacional (ORDBMS) com ênfase em extensibilidade e em padrões de conformidade. Como um servidor de banco de dados, sua principal função é armazenar dados de forma segura, apoiando as melhores práticas, permitindo a recuperação dos dados a pedido de outras aplicações de software. Ele pode lidar com cargas de trabalho que vão desde pequenas aplicações a aplicações de grande porte voltadas para a Internet, utilizada de forma simultânea por vários usuários. Tendo isso em vista, o PostgreSQL foi escolhido como gerenciador de banco de dados do sistema.

4.2. Métodos

Nessa sessão serão discutidas particularidades da organização do sistema implementado. Será explicado como instalar todos os softwares e APIs utilizados e também como comandos falados são de fato ouvidos e interpretados pelo sistema e resultam em um comando capaz de capturar uma imagem do ambiente e descrevê-lo.

4.2.1. Instalação e Inicialização do Sistema Operacional

Inicialmente realizou-se a formatação do cartão SD de 16 GB a ser utilizado na plataforma com o software SD Formatter:

Format Type: Full (OverWrite)
Format Size Adjustment: ON
Click: Format

Posteriormente, foi feito o Download da imagem do sistema operacional disponibilizada diretamente no site da Raspberry (*https://www.raspberrypi.org/downloads/raspbian/*), bem como a sua gravação no cartão com Win32 Disk Imager:

Image File: 2018-18-04-raspbian-jessie-lite
Click: Write

Ao iniciar a Raspberry foi realizado o login e então a configuração da senha de super usuário:

```
Login: pi
Password: raspberry
$ sudo passwd
```

E então realizou-se a configuração de localização, data e hora e a expansão do cartão SD (Expand Filesystem) através da interface de configuração da RPi:

```
$ sudo raspi-config
```

Com a reinicialização do sistema, é aconselhável realizar um update de todos os pacotes e softwares instalados:

```
$ apt-get update
$ apt-get upgrade
$ apt-get dist-upgrade
```

Posteriormente realizou-se a conexão WiFi editando-se o arquivo wpa_supplicant.conf localizado em /etc/wpa_supplicant/wpa_supplicant.conf como a seguir:

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
network={
    ssid="The_ESSID "
    psk="The_wifi_password"
}
```

No caso deste projeto, a conexão WiFi usada é um hotspot que utiliza os dados móveis de um smartphone. O sistema necessita conexão com a internet, para que ele seja portátil e móvel foi necessário configurar um hotspot no smartphone de acordo com a conexão configurada acima.

Foi configurado o acesso via SSH, por questões de segurança utilizou-se a porta 22000:

```
$ sudo raspi-config
5 Interfacing Options: Enable SSH
$ sudo nano /etc/ssh/sshd_config
Port 22000
```

Para garantir que o sistema possua uma conexão estável e segura também é necessário definir o “DNS Name Servers”:

```
$ sudo nano /etc/network/interfaces

source-directory /etc/network/interfaces.d

auto lo
iface lo inet loopback

iface eth0 inet manual
dns-nameservers 8.8.8.8 8.8.4.4

allow-hotplug wlan0
iface wlan0 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
dns-nameservers 8.8.8.8 8.8.4.4

allow-hotplug wlan1
iface wlan1 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
dns-nameservers 8.8.8.8 8.8.4.4
```

Além disso, para obter acesso as bibliotecas e pacotes do Raspbian, deve-se alterar o “sources.list”:

```
$ sudo nano /etc/apt/sources.list
deb http://archive.raspbian.org/raspbian/ stretch main contrib non-free rpi
```

Para configurar o funcionamento do teclado no padrão de teclados brasileiros, realizou-se:

```
$ sudo nano /etc/default/keyboard
XKBMODEL="abnt2"
XKBLAYOUT="br"
XKBVARIANT=""
XKBOPTIONS="lv3:alt_switch,compose:rctrl"
```

Finalmente, com a finalidade de facilitar o acesso remoto, eliminando a dependência de um IP estático instalou-se o No-IP, fazendo com que ele inicie automaticamente com o sistema:

```
$ cd ~
$ mkdir noip
$ cd noip
$ wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
$ tar vzxf noip-duc-linux.tar.gz
$ cd noip-2.1.9-1
$ sudo make
$ sudo make install

$ sudo nano /etc/init.d/noip
#! /bin/sh
# /etc/init.d/noip

### BEGIN INIT INFO
# Provides:          noip
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Simple script to start a program at boot
### END INIT INFO
# Carry out specific functions when asked to by the system
case "$1" in
  start)
    echo "Iniciando No-IP"
    # run application you want to start
    /usr/local/bin/noip2
  ;;
  stop)
    echo "Parando No-IP"
    # kill application you want to stop
    killall noip2
  ;;
  *)
    echo "Usage: /etc/init.d/noip {inicio|pausa}"
    exit 1
  ;;
esac
exit 0

$ sudo chmod 755 /etc/init.d/noip
$ sudo /etc/init.d/noip start
$ sudo update-rc.d noip defaults
```

Com todos os passos anteriores realizados, é iniciada a instalação de softwares e pacotes necessários para a implementação do projeto.

4.2.2. Instalação de Pacotes, Programas e Bibliotecas

Um Ambiente Virtual permite manter as dependências exigidas por diferentes projetos em locais separados, criando ambientes virtuais para Python. Em outras palavras, o virtualenv é uma ferramenta para criar ambientes isolados do Python. O virtualenv cria uma pasta que contém todos os executáveis necessários para usar os pacotes que um projeto Python precisaria.

As bibliotecas virtualenv e virtualenvwrapper (um conjunto de extensões para a ferramenta virtualenv) foram utilizados como uma maneira fácil de se criar ambientes isolados do Python usando como base a própria instalação do sistema para que fosse possível testar os programas desenvolvidos, de forma isolada, sem a necessidade de replicar cópias de Python.

Assim, cada projeto pode ter suas próprias dependências, independentemente de quais dependências cada outro projeto possui.

Processo de instalação e configuração do Virtualenv e do Virtualenvwrapper:

```
$ pip3 install virtualenv
$ mkdir ~/virtualenvs
$ virtualenv ~/virtualenvs/project –no-site-packages
$ pip3 install virtualenvwrapper

$ nano ~/.bashrc
VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
export WORKON_HOME=~/.virtualenvs

$ mkdir ~/.virtualenvs $ source ~/.bashrc
$ source ~/.bashrc

$ mkvirtualenv project
$ workon project
```

Para sair do virtualenv:

```
$ deactivate
```

Instalando pip para Python 3:

```
$ sudo apt-get install python3-pip
```

Instalação do Flask:

```
$ sudo pip3 install flask flask_restful flask_script flask_migrate
$ sudo pip3 install marshmallow flask_marshmallow
$ sudo pip3 install flask_sqlalchemy marshmallow-sqlalchemy psycopg2
```

Instalando o PyAudio:

```
$ sudo apt install portaudio19-dev python-all-dev
$ sudo pip3 install pyaudio

$ sudo nano /usr/share/alsa/alsa.conf
defaults.ctl.card 1
defaults.pcm.card 1
```

Instalação do Speech Recognition:

```
$ pip3 install SpeechRecognition
$ sudo apt-get install flac
```

Instalando o Google Cloud Python Client:

```
$ sudo pip3 install google-cloud
```

Instalação do Google Text-to-Speech:

```
$ sudo pip3 install gTTs
```

Instalando do Natural Language Toolkit:

```
$ sudo pip3 install nltk pyyaml
```

Instalação do fswebcam, omxplayer:

```
$ sudo apt-get install fswebcam omxplayer swig libpulse-dev
```

Instalando o PiWho:

```
$ sudo pip3 install piwho
```

Instalação do Python-Levenshtein:

```
$ sudo pip3 install python-Levenshtein
```

Processo de instalação e configuração do PostgreSQL:

```
$ apt-get install postgresql libpq-dev postgresql-client postgresql-client-common
$ su postgres
$ createuser pi -P --interactive

$ psql
> create database DB;
```

4.2.3. Configurações da Google Cloud

O primeiro passo para a utilização das ferramentas do Google Coud é a configuração de uma conta. A conta pode ser criada utilizando um *'login'* do Google ou do Gmail. O Google oferece um ano gratuitamente, desde que seja adicionado um cartão de crédito para faturamento. Em caso de cobranças, só são realizadas caso o dono da conta autorize. Depois de criar a conta, é possível configurar um projeto e ativar o faturamento.

Inicialmente, na página de projetos do Google Cloud é necessário criar um novo projeto. Posteriormente é necessário ativar o faturamento da sua conta (não haverá cobranças). Então, é necessário ativar as APIs que serão utilizadas no projeto. Finalmente basta obter uma chave JSON e vinculá-la ao uso das APIs, para autenticar seu uso.

Após salvar o JSON na Raspberry pi, deve-se criar uma variável de ambiente contendo o caminho para o arquivo chave:

GOOGLE_APPLICATION_CREDENTIALS.

4.3. Descrição do Sistema

Uma vez que a inicialização da Raspberry Pi 3 é realizada utilizando o SO Raspbian, bem como a instalação de todos os softwares necessários, foi feito as configurações necessárias para a utilização da Google Cloud API, como descrito no item **4.2.3** da seção anterior. Além dos componentes e softwares já citados, como descrito anteriormente (**Seção 4.1**), foi necessário uma *'Webcam'* USB com microfone integrado, já que a RPi não possui entradas analógicas de áudio para captura de áudio e imagens. Para a saída de áudio foi utilizado um fone de ouvido conectado a saída analógica de áudio. Para a alimentação do sistema, utilizou-se um cabo USB/micro USB e um *'Powerbank'*. A **Figura 4.1** abaixo ilustra o *'hardware'* utilizado.

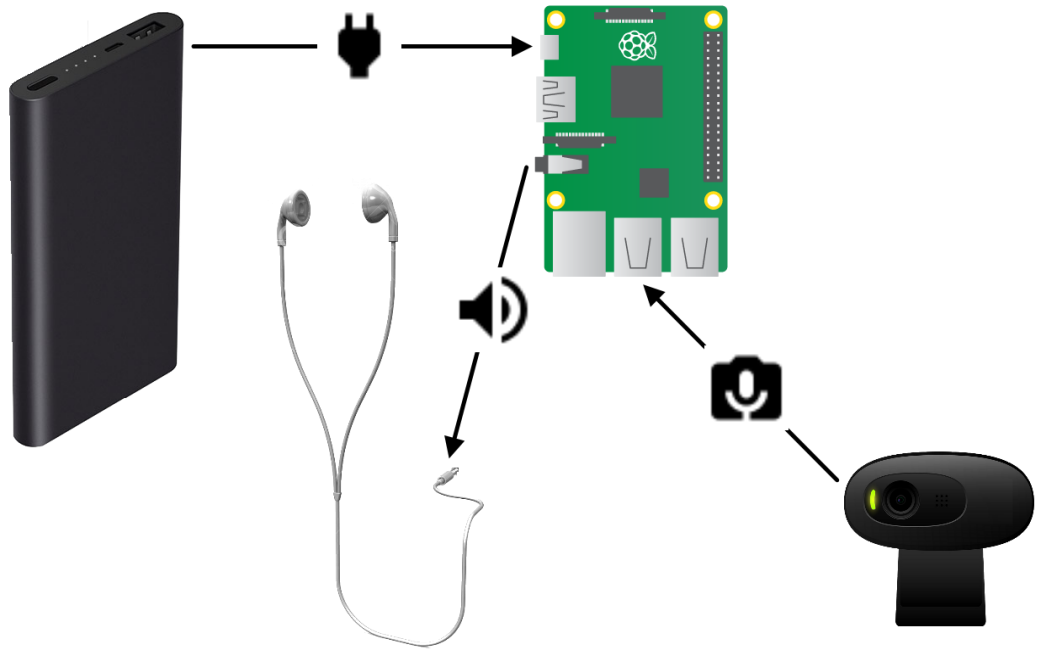

Figura 4.4: Diagrama de entradas e saída do sistema.

Durante o funcionamento do sistema, para evitar que o mesmo tente executar comandos toda vez que alguém falar no alcance do microfone, é necessário estabelecer uma palavra chave para iniciar os comandos. O código base apresenta um "loop" base que verifica constantemente se a palavra chave foi dita ou não.

O programa executa o reconhecimento de interlocutor através do framework PiWho, a fim de realizar uma verificação de segurança, para garantir que somente quem tenha permissão possa executar os comandos (o cadastro de permissões deve ser realizado manualmente). Com a autenticação efetuada, a verificação consiste na gravação periódica do som ambiente, que é pré-processado e enviado ao Google Cloud Speech Recognition API para verificação de fala, e então é execução o processo STT. O texto retornado é processado pelo classificador de textos para determinar a ocorrência da palavra-chave.

Caso a verificação falhe, o sistema retorna ao estado de espera pela chamada da palavra-chave. Se a verificação for válida, para evitar chamadas excessivas à API o sistema espera que o comando tenha sido dito concomitantemente com a palavra-chave, então para o mesmo texto retornado do processo STT é feita a verificação de um comando válido, com o uso do classificador de textos. Com um comando detectado, o programa captura uma imagem do ambiente, realiza um pré-processamento e a envia para o Google Cloud Vision que executa a classificação da imagem correspondente ao comando. Há comandos para classificar: rótulos, textos, faces, logos e ponto de referência.

Assim que a API de CV retorna, um som indicando sucesso é reproduzido validando o processo, então realiza-se o processamento e tradução da resposta. A partir disso é gerada uma resposta através do processo TTS realizado pela Google Cloud Text-to-Speech. Posteriormente o programa retorna ao "loop". No caso de nenhum comando válido seja passado ao sistema, o mesmo retorna ao seu estado de "loop", aguardando a chamada. Para evitar a interferência do ruído ambiente, inicialmente é executado um ajuste de ruído a fim de reduzir a taxa de SNR durante as gravações.

Quando um comando é executado com sucesso, o programa chama a RESTful API e grava os dados referentes à execução no banco de dados. Estes dados foram usados para a análise dos resultados que será apresentada no **Capítulo 5**.

O código implementado para cadastrar as vozes, para autenticação, realiza o treinamento da voz associando-a ao nome do respectivo interlocutor, que é passado por argumento. Sua lógica apresenta um "loop" que verifica através do processo STT via GCSR que a frase dita durante o treinamento corresponda a palavra-chave pré-estabelecida. Após a validação da verificação da palavra-chave, o sistema grava um arquivo de áudio com a voz correspondente e a associa com o respectivo nome.

A acurácia do processo de reconhecimento de interlocutor depende da quantidade de amostras salvas para cada pessoa cadastrada. O treinamento de novas vozes pode ser chamado a partir da execução principal do programa como um comando.

Todo o código do programa desenvolvido se encontra no **Apêndice**. Diagrama da execução principal do programa:

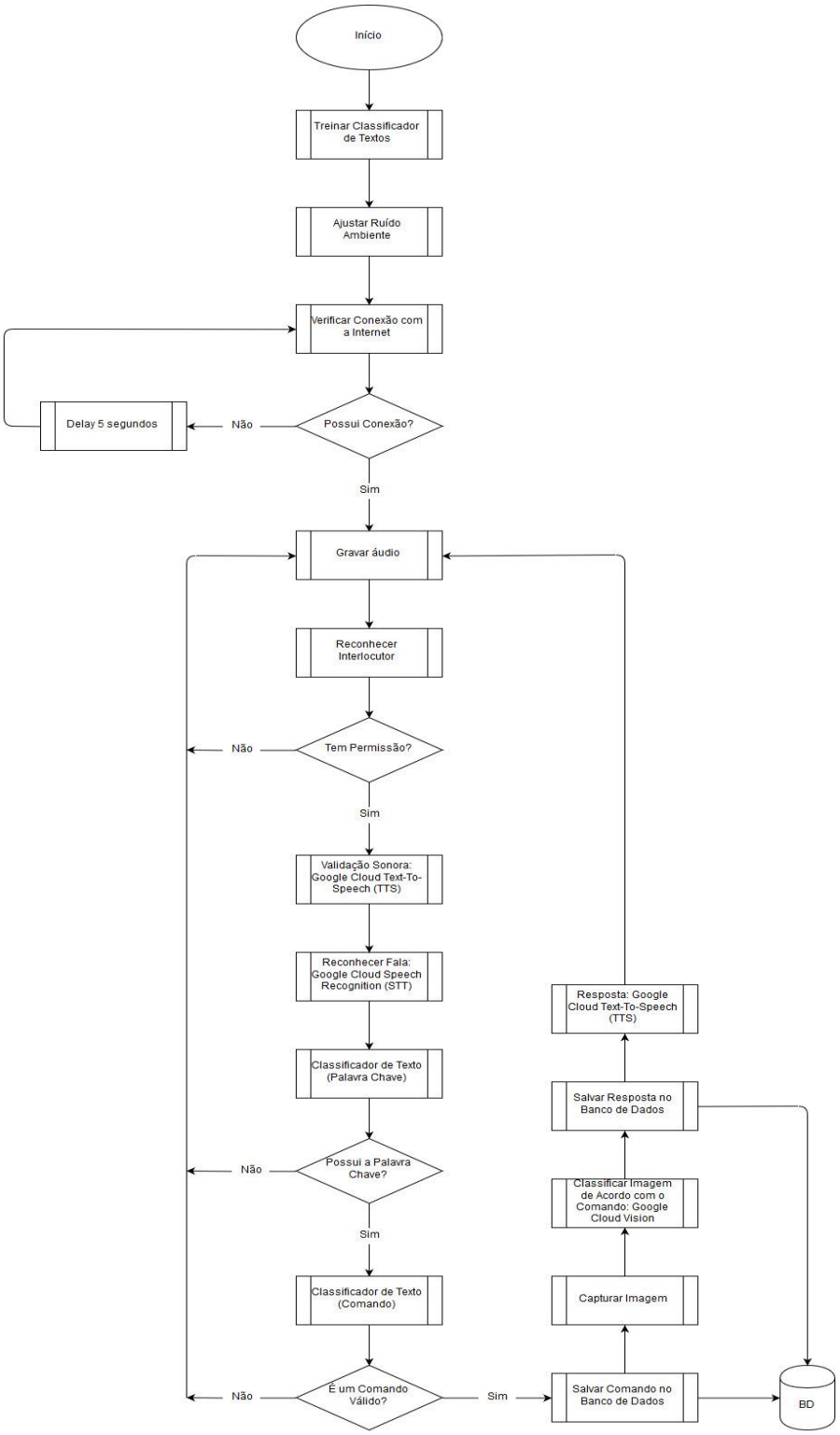

Figura 4.5: Diagrama da lógica principal do programa.

5. RESULTADOS E DISCUSSÕES

Com a implementação das funcionalidades planejadas para o presente projeto, o passo seguinte foi realizar os testes necessários. Esse capítulo será dividido em três seções, sendo que na primeira estão os resultados do reconhecimento de fala e de extração de informação de imagens, a segunda tratará da análise de consumo e a terceira da análise desempenho de requisitos do sistema.

5.1. Descrição de Imagens e Reconhecimento de Voz

Nesta etapa, foram necessárias duas considerações para que fosse possível garantir o funcionamento correto de desempenho do sistema: primeiro foi o tempo de resposta, já que não é conveniente que o usuário fique esperando por muito tempo a informação requisitada, e segundo foi a porcentagem de erros dos resultados obtidos, de modo que o usuário recebesse informação confiável do aplicativo. A Google Cloud API é uma ferramenta online, então a primeira dificuldade encontrada para que o processamento ocorresse de forma eficiente, foi a transferência de dados para o servidor da Google. Pois, quanto maior a quantidade de dados transmitidos pela rede e processados pela rotina do servidor, maior o tempo de latência para a resposta, significando que a largura de banda da Internet é um fator limitante.

5.1.1. Reconhecimento de Palavra Chave

Para teste do desempenho do sistema, foi realizada uma sequência de 105 testes de reconhecimento de fala (STT) em um ambiente silencioso e 50 em um ambiente ruidoso, com diversas pessoas conversando ao fundo. Vale ressaltar que os testes foram feitos com o microfone próximo ao locutor, cerca de 15 cm, devido a qualidade do microfone utilizado, que neste caso é um microfone integrado a uma *'Webcam'*, e não possui uma captação omnidirecional. Este teste foi referente às palavras-chave escolhidas para o sistema: Ok Google. A taxa de acerto foi anotada para ambos os casos.

Para o teste realizado em ambiente silencioso (105 amostras, para reconhecer a palavra-chave: "Ok Google"), o sistema reconheceu apenas a palavra-chave, nos casos em que falho não conseguiu reconhecer palavra alguma. As **Tabelas 5.1 e 5.2** abaixo mostram a relação de reconhecimento obtida:

Tabela 5.1: Taxa de reconhecimento em ambiente silencioso.

Palavras	Taxa de Reconhecimento
Ok Google	92,38%
Nenhuma Palavra	7,62%

Para o teste realizado em ambiente ruidoso (50 amostras), o sistema reconheceu mais de 3 palavras diferentes. A tabela abaixo mostra a relação de reconhecimento obtida:

Tabela 5.2: Taxa de reconhecimento em ambiente ruidoso.

Palavras	Taxa de Reconhecimento
Ok Google	54%
Outras palavras	14%
Nenhuma Palavra	32%

É possível notar que a taxa de reconhecimento depende de diversos fatores, sendo um deles a quantidade de ruídos no ambiente. Mesmo executando um ajuste do ruído de fundo, há uma queda da eficiência deste processo em um ambiente mais ruidoso. Outros fatores que podem influenciar nesta taxa são: o sotaque, a distância do interlocutor com o microfone, a dicção, a voz, a qualidade do áudio captado pelo microfone, etc. A utilização da palavra chave escolhida ocorreu naturalmente, já que é a mesma palavra chave utilizada por diversas aplicações do Google, e se mostrou eficiente.

5.1.2. Funcionalidade dos Comandos por Voz

Foi feita, ainda, uma análise quanto ao tempo de resposta para que o sistema consiga realizar o processo STT, bem como a precisão do processo. Como esse processo é feito através de computação em nuvem, necessitando de conexão com a internet, ele representa um ponto crítico do sistema.

Criou-se um método de avaliação para estudar o desempenho do sistema de reconhecimento de fala. Para os testes foram definidas duas larguras de banda: 10 Mbps, típica em conexões 4G, e 35Mbps. Para cada largura de banda, executou-se 3 testes para cada comando que pode ser executado no sistema, totalizando 10 chamadas. A Raspberry Pi transmite dados de voz para o servidor Google Cloud, através de sua RESTful API Google Cloud Speech Recognition. O servidor recebe, então, os pacotes de voz, traduz a voz em texto e envia o texto de volta ao cliente.

Cliente recebe o texto e calcula o atraso do texto recebido e a distância de Levenshtein, usada para mensurar a precisão da transcrição, ou a porcentagem de correspondência da string original, falada, e da string que é criada pelo ASR.

O cliente recebe a voz e transmite pacotes de voz para o servidor do Google. O Google começa a traduzir o pacote de voz assim que o recebe, e envia o texto reconhecido de volta ao cliente assim que o processo de reconhecimento acaba. O cliente registra a data e hora de cada pacote de resposta e também o horário de início da transmissão de voz para calcular a latência da transcrição. O cliente também compara o texto retornado ao original para calcular a precisão da transmissão. A distância de Levenshtein é usada para calcular a porcentagem de correspondência da string original e da string criada pelo Google Cloud Speech Recognition.

As métricas apresentadas abaixo, pelo teste efetuado são precisão e atraso da transcrição sob diferentes valores de largura de banda. Todos os resultados foram obtidos com a mesma configuração do sistema. Posteriormente será feita a analise do efeito da largura de banda em cada chamada. Esta seção de testes foi feita em ambiente sem presença de ruído. Os resultados se encontram nas **Tabelas de 5.3 à 5.12** a seguir. O percentual de acerto de cada imagem pode ser calculado, utilizando-se a distância de Levenshtein, pela seguinte equação:

$$\text{Erro} = \frac{\text{Distância de Levenshtein}}{\text{Total de caracteres na imagem}} \cdot 100\% \qquad (\mathbf{5.1})$$

Resultados para o comando de rotulação:

Tabela 5.3: Resultado para o comando de rotulação com largura de banda de 35 Mbps.

Métricas	Resultados
Comando	Ok google descreva o ambiente
Texto extraído	Ok google descreva o ambiente
Latência	1523 ms
Distância de Levenshtein	0
Percentual de acerto	100%

Tabela 5.4: Resultado para o comando de rotulação com largura de banda de 10 Mbps.

Métricas	Resultados
Comando	Ok google descreva o ambiente
Texto extraído	Ok google descreva o ambiente
Latência	2233 ms
Distância de Levenshtein	0
Percentual de acerto	100%

Resultados para o comando de detecção de textos:

Tabela 5.5: Resultado para para o comando de detecção de textos com largura de banda de 35 Mbps.

Métricas	Resultados
Comando	Ok google leia o texto
Texto extraído	Ok google ler o texto
Latência	1798 ms
Distância de Levenshtein	2
Percentual de acerto	90,91%

Tabela 5.6: Resultado para o comando de detecção de textos com largura de banda de 10 Mbps.

Métricas	Resultados
Comando	Ok google leia o texto
Texto extraído	Ok google ler o texto
Latência	2612 ms
Distância de Levenshtein	2
Percentual de acerto	90,91%

Resultados para o comando de detecção de faces:

Tabela 5.7: Resultado para o comando de detecção de faces com largura de banda de 35 Mbps.

Métricas	Resultados
Comando	Ok google descreva a pessoa
Texto extraído	Ok google descrever pessoa
Latência	1131 ms
Distância de Levenshtein	3
Percentual de acerto	88,89%

Tabela 5.8: Resultado para o comando de detecção de faces com largura de banda de 10 Mbps.

Métricas	Resultados
Comando	Ok google descreva a pessoa
Texto extraído	Ok google descrever pessoa
Latência	1908 ms
Distância de Levenshtein	3
Percentual de acerto	88,89%

Resultados para o comando de detecção de logos:

Tabela 5.9: Resultado para o comando de detecção de logos com largura de banda de 35 Mbps.

Métricas	Resultados
Comando	Ok google detectar logo
Texto extraído	Ok google detectar logo
Latência	1258 ms
Distância de Levenshtein	0
Percentual de acerto	100%

Tabela 5.10: Resultado para o comando de detecção logos com largura de banda de 10 Mbps.

Métricas	Resultados
Comando	Ok google detectar logo
Texto extraído	Ok google detectar logo
Latência	2012 ms
Distância de Levenshtein	0
Percentual de acerto	100%

Resultados para o comando de detecção de pontos de referência:

Tabela 5.11: Resultado para o comando de pontos de referência com largura de banda de 35 Mbps.

Métricas	Resultados
Comando	Ok google identifique o ponto de referência
Texto extraído	Ok google identificar ponto de referência
Latência	2139 ms
Distância de Levenshtein	5
Percentual de acerto	88,37%

Tabela 5.12: Resultado para o comando de pontos de referência com largura de banda de 10 Mbps.

Métricas	Resultados
Comando	Ok google identifique o ponto de referência
Texto extraído	Ok google identificar ponto de referência
Latência	2761 ms
Distância de Levenshtein	5
Percentual de acerto	88,37%

A partir dos resultados é possível inferir que a largura de banda da conexão não influencia na precisão do reconhecimento da fala não havendo diferenças entre os resultados da tradução de voz para texto para ambas as conexões. A conexão influencia apenas na latência, resultando em uma latência média de 1569 ms para a largura de banda de 35 Mbps e de 2305 ms para 10 Mbps. Pode-se dizer que o aumento da latência deve-se ao aumento no tempo de transferência dos arquivos de voz, já que a taxa de transmissão não influi no processamento do audio pela API do Google Cloud. Outro ponto importante foi o percentual de acerto das chamadas, com uma média de 93,63%.

O GCSR utiliza o protocolo de conexão TCP. Neste protocolo de comunicação Web o remetente transmite os pacotes que podem chegar ao receptor fora de ordem ou podem se perder. No caso de perda o remetente retransmite o pacote.

Caso fosse utilizado o protoculo UDP, que não garante transferência de todos os pacotes sem perdas, o resultado poderia ter sido diferente para as larguras de banda utilizadas. Isso significa que se for diminuída a largura de banda disponível para esta comunicação, a taxa de perda de pacotes pode aumentar e consequentemente a latência do processo aumenta devido a atrasos. O GCSR inicia o processamento assim que recebe o primeiro pacote de voz, e posteriormente os reordena, diminuindo o impacto da latência de rede em seu processamento. Mas, é possível afirmar, a partir dos resultados, que o atraso aumenta com a diminuição da largura de banda, que naturalmente impacta na rapidez da transferencia de dados.

A vantagem de se utilizar o protocolo TCP é que não há perda de nenhum pacote, mas o servidor aguarda para receber pacotes perdidos e esse atraso pode afetar a saída do reconhecedor de fala. O GSR resolveu esse problema pela interação completa entre o reconhecedor de fala e o receptor.

Outro fator que influencia o tempo de cada chamada é o jitter, variação no atraso dos pacotes transmitidos. Essa variação faz com que os pacotes sejam recebidos fora de ordem. A conexão TCP faz o reordenamento de pacotes no receptor, fazendo com que este efeito não afete a precisão do GCSR. O jitter aumenta apenas o tempo de resposta da API.

5.1.3. Reconhecimento de Interlocutor

Outro processo analisado foi o reconhecimento de interlocutor. Mesmo aumentando a quantidade de amostras de áudio treinadas para cada interlocutor, num total de três, observa-se um certo limite para a qualidade do reconhecimento. Em muitas vezes, o processo é incapaz de reconhecer interlocutor algum, mais precisamente em 33 chamadas de um total de 92 que foram testadas. Outro problema é a detecção errônea de um interlocutor, mas este é mais raro, sendo notado apenas duas vezes.

5.1.4. Teste para Diferentes Dimensões de Imagens

A largura de banda da internet é um fator limitante para o usuário do aplicativo, porém a quantidade de dados transmitidas pode ser flexibilizada de modo que a imagem tenha sua resolução reduzida e exiba um número menor de bytes para ser representada. Essa redução na resolução da imagem vai resultar em um detalhamento menor da mesma, podendo dificultar seu processamento, ou até mesmo causar erros de interpretação de seu conteúdo, o que vai gerar resultados incorretos.

A Google Cloud Vision API apresenta as mesmas características quanto a resposta à largura de banda que foram observadas na **Seção 5.1.2**. O GCV inicia o processamento assim que recebe o primeiro pacote da imagem, e posteriormente os reordena, diminuindo o impacto da latência de rede em seu processamento. Porém o atraso aumenta com a diminuição da largura de banda, que naturalmente impacta na rapidez da transferencia de dados.

Com o intuito de confrontar a resolução da imagem, o tempo de processamento e os erros dos resultados, para cada solicitação de informação das imagens, a mesma imagem contida na **Figura 5.1** foi submetida com diversas resoluções, e os tempos de processamento foram medidos, todos utilizando-se uma largura de banda de 10Mbps. Na **Tabela 5.13** tem-se os resultados para três diferentes resoluções.

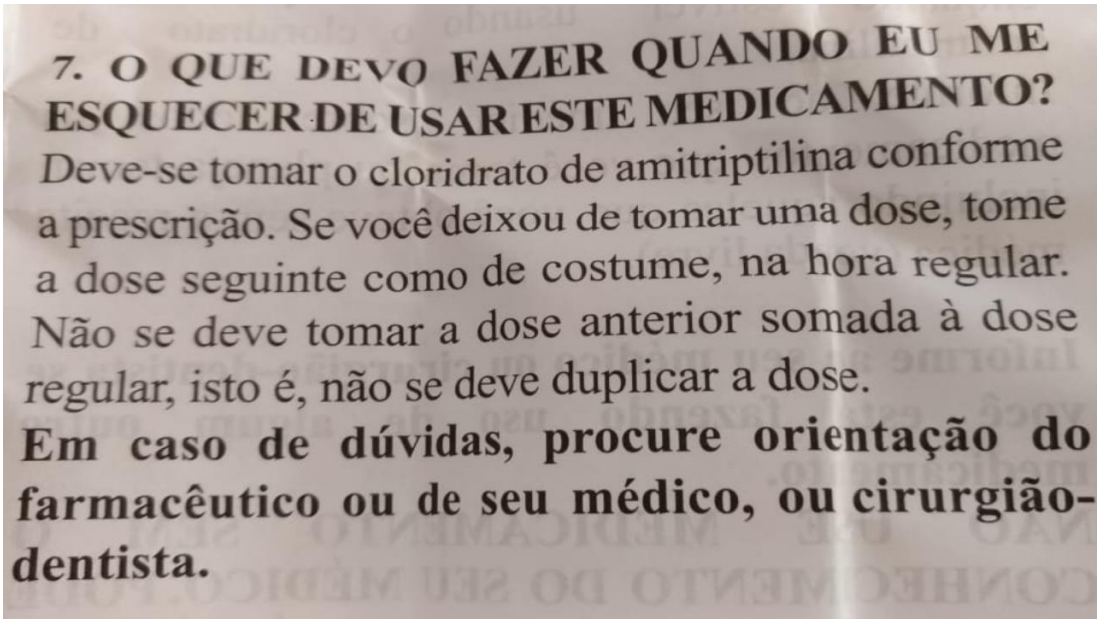
7. O QUE DEVO FAZER QUANDO EU ME ESQUECER DE USAR ESTE MEDICAMENTO?
Deve-se tomar o cloridrato de amitriptilina conforme a prescrição. Se você deixou de tomar uma dose, tome a dose seguinte como de costume, na hora regular. Não se deve tomar a dose anterior somada à dose regular, isto é, não se deve duplicar a dose.
Em caso de dúvidas, procure orientação do farmacêutico ou de seu médico, ou cirurgião-dentista.

Figura 5.1: Imagem de uma bula de remédio, com amassados e sombra de texto no verso.

Tabela 5.13: Resultado da extração de texto de uma bula de remédio para as resoluções de 1280x720, 720x405 e 480x270 pixels.

Resolução	Texto Extraído
1280x720	7. O QUE DEVO FAZER QUANDO EU ME ESQUECER DE USAR ESTE MEDICAMENTO? Deve-se tomar o cloridrato de amitriptilina conforme a prescrição. Se você deixou de tomar uma dose, tome a dose seguinte como de costume, na hora regular. Não se deve tomar a dose anterior somada à dose regular, isto é, não se deve duplicar a dose. Em caso de dúvidas, procure orientação do farmacêutico ou de seu médico, ou cirurgião-dentista.

720x405	7. O QUE DEVO FAZER QUANDO EU ME ESQUECER DE USAR ESTE MEDICAMENTO? Deve-se tomar o cloridrato de amitriptilina conforme a prescrição. Se você deixou de tomar uma dose, tome a dose seguinte como de costume, na hora regular. Não se deve tomar a dose anterior somada à dose regular, isto é, não se deve duplicar a dose. Em caso de dúvidas, procure orientação do farmacêutico ou de seu médico, ou cirurgião-dentista. TO O OTVAMDHVO
480x270	7. O QUE DEVO FAZER QUANDO EU ME ESQUECER DE USAR ESTE MEDICAMENTO? Deve-se tomar o cloridrato de amitriptilina conforme a prescrição. Se você deixou de tomar uma dose, tome a dose seguinte como de costume, na hora regular. Não se deve tomar a dose anterior somada à dose regular, isto é, não se deve duplicar a dose. Em caso de dúvidas, procure orientação do farmacêutico ou de seu médico, ou cirurgião-dentista. dentista. U OG OTVAD O

Na **Tabela 5.14**, abaixo, mostra que a resolução da imagem é proporcional ao tempo total de processamento da informação requisitada e inversamente proporcional a quantidade de acertos do resultado. Sendo fundamental lembrar que o tempo de processamento será sempre relacionado a largura de banda da internet, de modo que não são valores absolutos. O percentual de acerto de cada imagem pode ser calculado, utilizando-se a distância de Levenshtein, como na **Equação 5.1**.

Tabela 5.14: Tempos da extração de texto para as resoluções de 1280x720, 720x405 e 480x270 pixels.

Métricas	**Resultados**		
Resolução	1280x720	720x405	480x270
Percentual de acerto	100%	96,45%	94,56%
Tempo de compressão	912 ms	591 ms	253 ms
Tempo de transferência e processamento	4142 ms	3617 ms	3190 ms
Tempo total	5053 ms	4208 ms	3443 ms

A partir da **Tabela 5.14** podem ser retiradas várias características importantes dos resultados obtidos. Primeiramente, os amassados no papel não afetaram a extração do texto. O que afetou o processo OCR foi o texto sombra formado no verso do papel que corresponde exatamente aos caracteres incorretos, em vermelho, do resultado apresentado pela **Tabela 5.13**, erro esse que não ocorreu para a resolução de 1280x720.

Assim, é possível inferir que a redução da resolução tornou o OCR menos preciso e esse problema foi potencializado pelo texto sombra apresentado no verso do papel gerando falsos reconhecimentos de caracteres.

É possível que se não houvesse texto no verso da bula o resultado das resoluções menores fosse praticamente idêntico ao de 1280x720, como pode ser comprovado pela **Tabela 5.13**. Porém, foi comprovado que dimensões inferiores podem reduzir a acurácia do resultado, como no resultado obtido, com reconhecimento errôneo de alguns caracteres, mas, por outro lado, dimensões superiores aumentam o tempo de processamento e uso da largura de banda. A partir desta análise, a resolução escolhida para a captura de imagens foi 1280x720 pixels, resolução máxima suportada pela *'Webcam'* utilizada. Esta resolução é muito próxima do valor recomendado pela Google: 1024x768. A utilização de imagens com resoluções inferiores não é justificada quando pode-se perder o valor da informação para se obter um ganho de apenas 1610 ms no tempo deste processo.

5.1.5. Teste para Reconhecimento de Texto Girado

Fazendo a consideração de que o usuário não terá como identificar qual a posição do texto no qual deseja obter informações, é possível que ocorra a captura de um texto disposto em diferentes ângulos. Dessa forma, ao se realizar os testes sobre os textos com diferentes ângulos, foi identificado que entre -90º, 90º e 180º não há nenhum problema na extração de informação. Para todos os graus de rotação o percentual de acerto foi de 100%. A imagem utilizada no teste pode ser visualizada na **Figuras 5.2**.

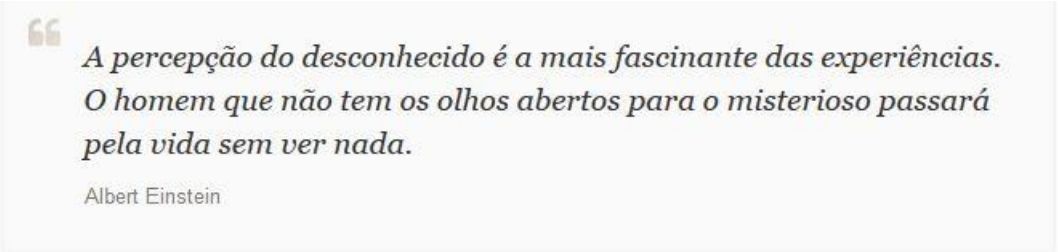

Figura 5.2: Extração de texto de imagem variando a rotação.

Tabela 5.15: Extração de texto para imagem rotacionada em 0, 90, -90 e 180º.

Rotação	Texto extraído
0º	A percepção do desconhecido é a mais fascinante das experiências. O homem que não tem os olhos abertos para o misterioso passará pela vida sem ver nada. Albert Einstein

90º	A percepção do desconhecido é a mais fascinante das experiências. O homem que não tem os olhos abertos para o misterioso passará pela vida sem ver nada. Albert Einstein
-90º	A percepção do desconhecido é a mais fascinante das experiências. O homem que não tem os olhos abertos para o misterioso passará pela vida sem ver nada. Albert Einstein
180º	A percepção do desconhecido é a mais fascinante das experiências. O homem que não tem os olhos abertos para o misterioso passará pela vida sem ver nada. Albert Einstein

5.1.6. Teste para Reconhecimento de Texto Manuscrito

Uma outra possibilidade na utilização do aplicativo seria a leitura de bilhetes escritos à mão. Foram realizados alguns testes para entender os limites das capacidades da API, quanto a escrita à mão, seja de forma ou cursiva.

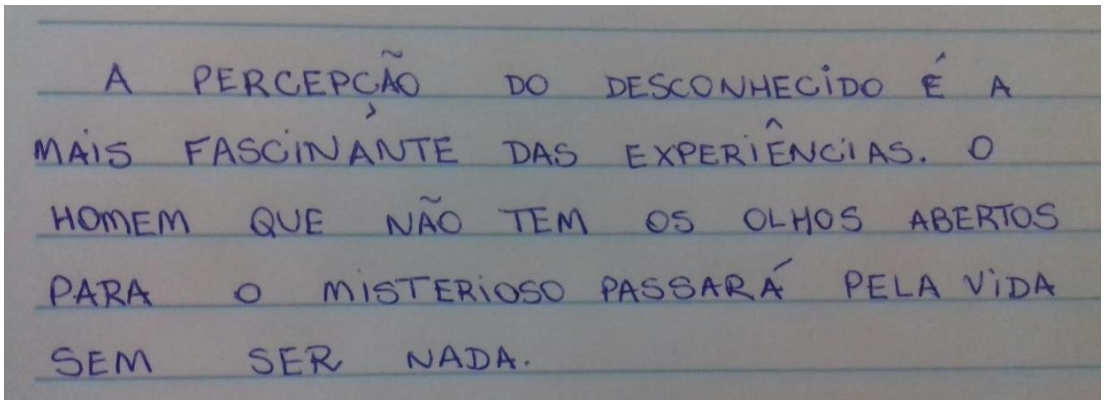

Figura 5.3: Texto escrito a mão com letra de forma.

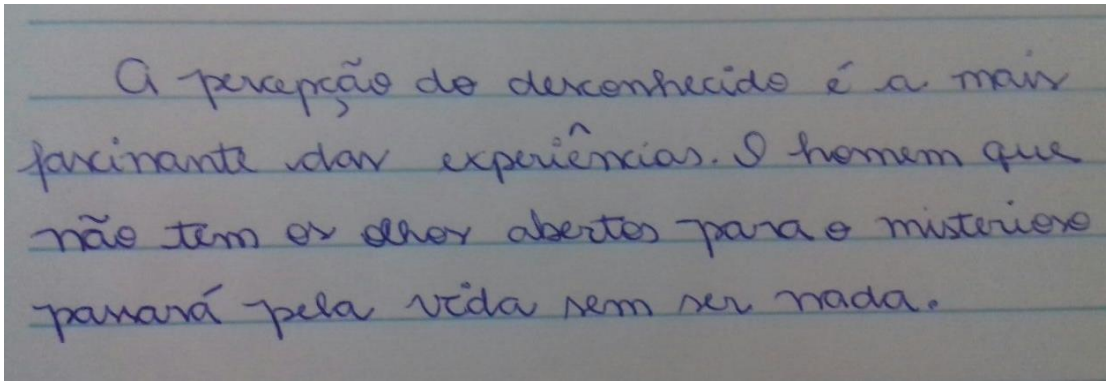

Figura 5.4: Texto escrito a mão com letra cursiva.

Tabela 5.16: Resultado da extração de texto em escrita de forma.

Tipo de texto	Texto extraído	Acerto
Manuscrito com letra de forma	A PERCEPCAO DO DESCONHECIDO E A MAIS FASCINANTE DAS EXPERIENCIAS. O HOMEM QUE NÃO TEM OS OLHOS ABERTOS PARA O MISTERIOSO PASSARA PELA VIDA SEM SER NADA.	96,84%
Manuscrito com letra cursiva	- a percepção de desconhecido a mi ancinante day experiencias, o homem que mãe tim ey sdhey abertos para e musteriero pawana pela Ntda sem er nada	81,01%

Os resultados obtidos mostraram que existe uma limitação maior da API quanto a identificação de caracteres escritos à mão, quando comparadas com as digitais. A **Tabela 5.16** mostra um exemplo do reconhecimento de um texto manuscrito com letra de forma e um manuscrito com letra cursiva, ambos contendo o mesmo conteúdo, capturados com a mais alta resolução que o aplicativo oferece, de 1280x720 pixels. Ainda assim houve erros de identificação do texto, como está destacado na **Tabela 5.16** acima. No caso do texto manuscrito com letra de forma os erros foram notados apenas na falta de acentuação. Entretanto, observa-se que os resultados se tornam críticos quando os textos são escritos em letra cursiva. Nesse caso, o reconhecimento acontece, porém com uma quantidade bem pequena do total de caracteres, apresentando diversos erros.

5.1.7. Teste para Rotulação de Elementos da Cena

Um dos objetivos principais do projeto, é que o usuário saiba o que está se passando ao seu redor. Portanto, foram realizados testes para saber quais as limitações da API na detecção de objetos a partir de imagens, apontando a câmera para diversos objetos, com o intuito de avaliar sua performance nesse quesito. Nas **Figuras 5.5, 5.6, 5.**7 **e 5.8** tem-se os resultados completos retornados para extração de rótulos da imagem de uma calopsita, dos arcos da EESC (Escola de Engenharia de São Carlos), de um notebook, e de um carro.

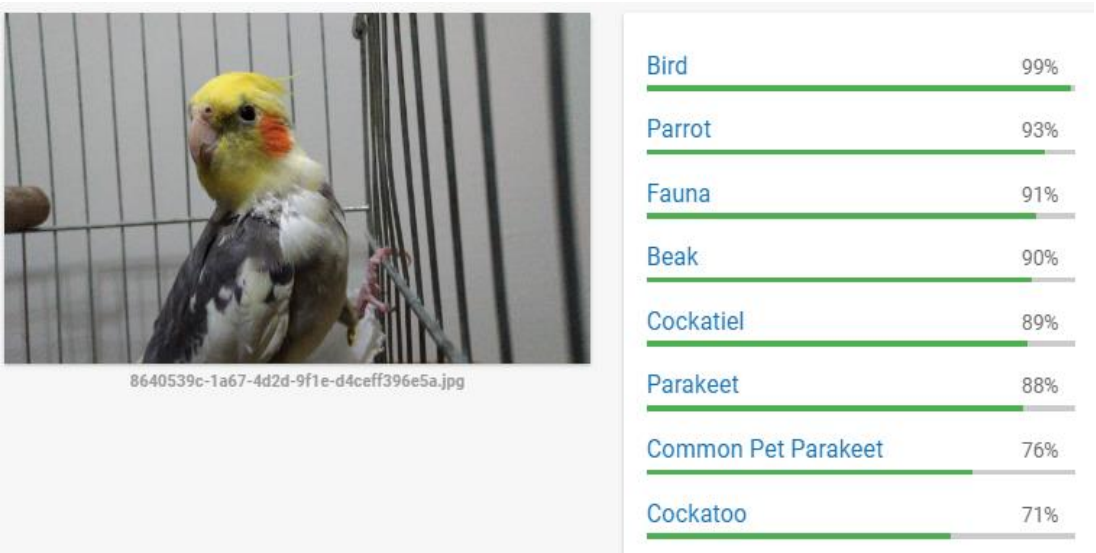

Figura 5.5: Rótulos extraídos da imagem de uma calopsita.

É possível identificar que há características que não são aplicáveis para o animal da foto, como por exemplo, “papagaio” (*parrot*), “periquito” (*parakeet*) e “cacatua” (*cockatoo*). Porém, como o aplicativo não considera resultados com pontuação abaixo de 80%, alguns rótulos foram desconsiderados no resultado visualizado pelo usuário.

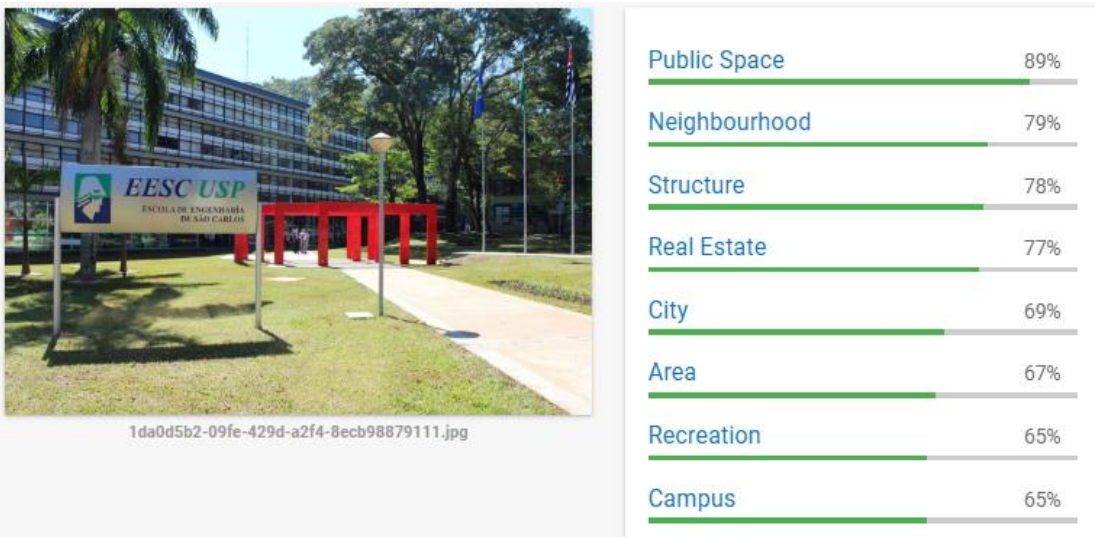

Figura 5.6: Rótulos extraídos da imagem dos arcos da EESC.

Pode-se identificar que há rótulos que não são aplicáveis para a paisagem da foto, como por exemplo, “imobiliária” (*real state*) e “recreação” (*recreation*). Alguns rótulos foram desconsiderados no resultado visualizado pelo usuário, e resultados relevantes como “campus” foram eliminados da resposta ao usuário.

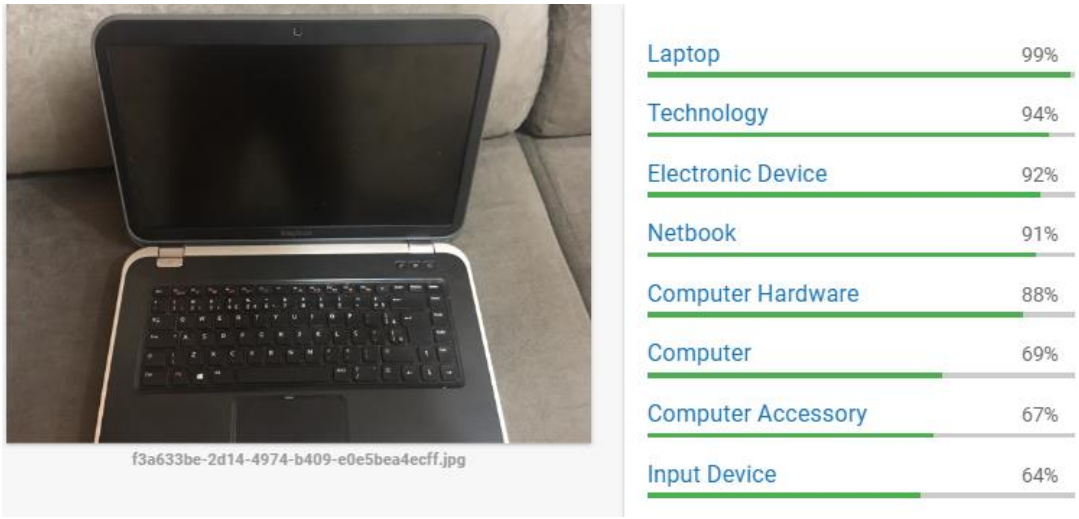

Figura 5.7: Rótulos extraídos da imagem de um notebook.

Neste caso, é possível identificar que a maioria rótulos se aplicam ao notebook da foto. Porém, como o aplicativo não considera resultados com pontuação abaixo de 80%, um rótulo relevante foi desconsiderado no resultado visualizado pelo usuário: "computador" (*computer*).

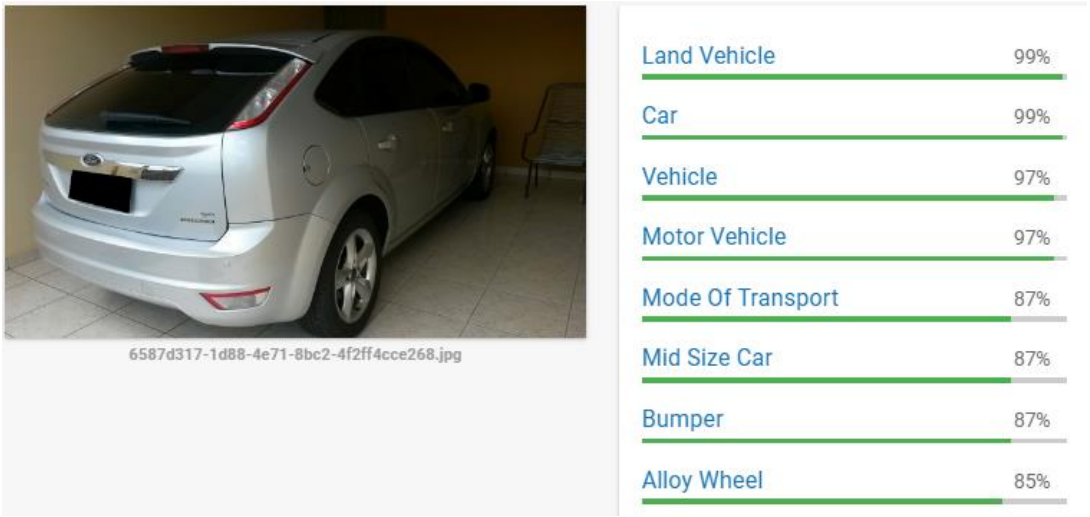

Figura 5.8: Rótulos extraídos da imagem de um carro.

Neste caso, identificou-se que todos rótulos retornados se aplicam ao carro da foto. Foram excessivos resultados para descrever a imagem, tantos que o limite de 8 resultados máximos que foi estabelecido foi aplicado.

A escolha do valor mínimo da pontuação para que cada rótulo seja aceito pelo aplicativo foi puramente empírico. Para a **Figura 5.7** o resultado foi satisfatório para a descrição, mas para a Figura 5.6 o valor escolhido de pontuação eliminou quase todos os resultados mais corretos, como por exemplo "campus" que era a palavra mais adequada, deixando apenas "espaço público".

É difícil definir um valor que simultaneamente seja capaz de descrever um elemento da cena e que não sobrecarregue o usuário com informações, muitas vezes desnecessárias, e uma descrição curta e precisa dificilmente será conseguida. A **Figura 5.8** apresenta 8 resultados, todos corretos, mas excessivos e alguns desnecessários para a descrição do automóvel.

Nos casos citados, independentemente de qual foi o critério para ignorar alguns resultados, todos os rótulos quase sempre tiveram relação com o objeto na imagem. Em nenhum caso a API falhou em identificar ao menos um rótulo corretamente para a imagem capturada.

5.1.8. Teste para Detecção de Ponto de Referência

A **Figura 5.9** a seguir apresenta a Catedral da cidade de São Carlos e foi utilizada para testar a capacidade do sistema de reconhecer um ponto de referência corretamente. A chamada da API foi capaz de reconhecer corretamente o ponto de referência utilizado, o retorno obtido foi: Catedral de São Carlos.

Figura 5.9: Catedral de São Carlos.

5.1.9. Teste para Detecção de Logo

A **Figura 5.10** a seguir apresenta o logo da USP e foi utilizada para testar a capacidade do sistema de reconhecer um logo corretamente. A chamada da API foi capaz de reconhecer corretamente o logo utilizado, o retorno obtido foi: Universidade de São Paulo.

Figura 5.10: Logo da Universidade de São Paulo.
Fonte: http://www5.usp.br/

5.1.10. Teste para Classificação de Expressão Facial

A classificação de expressões faciais é mais uma das características que se pode obter de uma imagem. As **Figura 5.11, 5.12, 5.13 e 5.14** mostram os resultados obtidos após solicitar o serviço de detecção de faces e extração de suas expressões faciais.

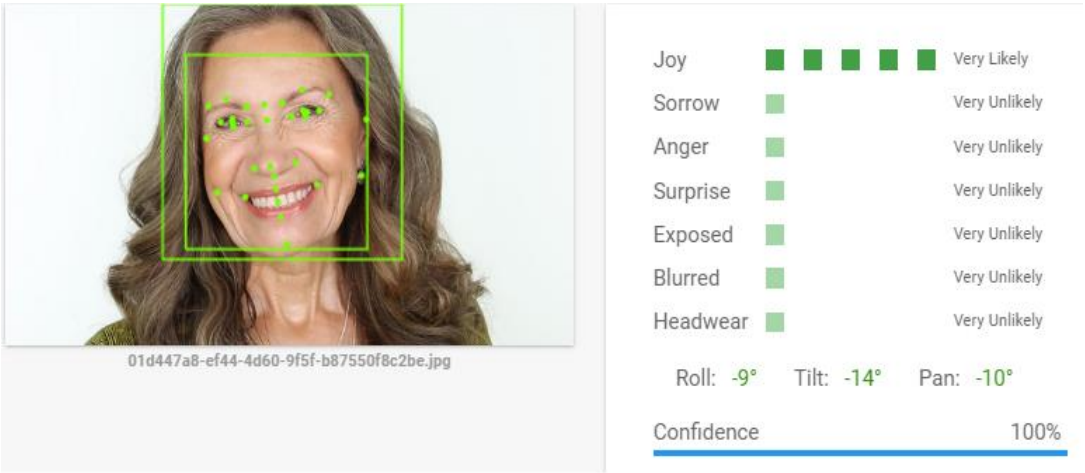

Figura 5.11: Expressão facial de uma mulher alegre.
Fonte: https://imageshack.us/

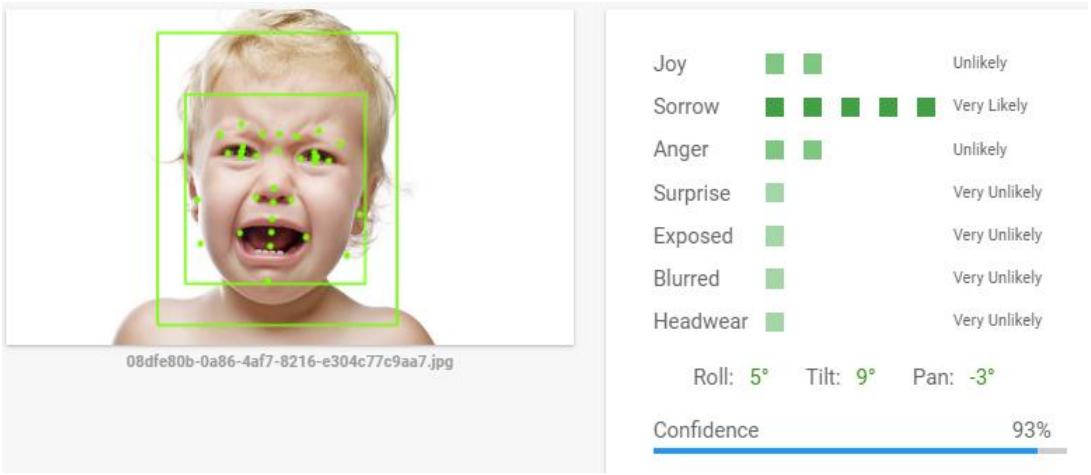

Figura 5.12: Expressão facial de uma criança triste.
Fonte: Fonte: https://imageshack.us/

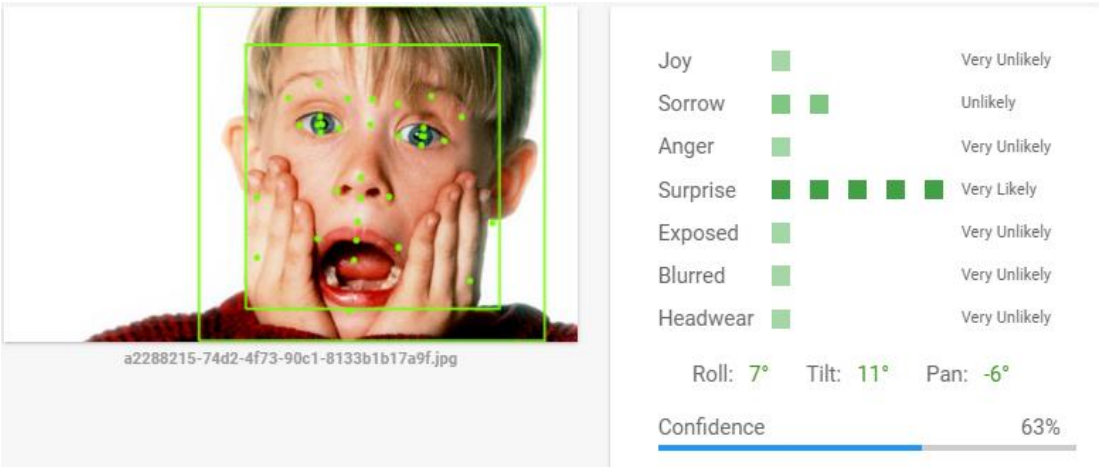

Figura 5.13: Expressão facial de um garoto surpreso.
Fonte: https://imageshack.us/

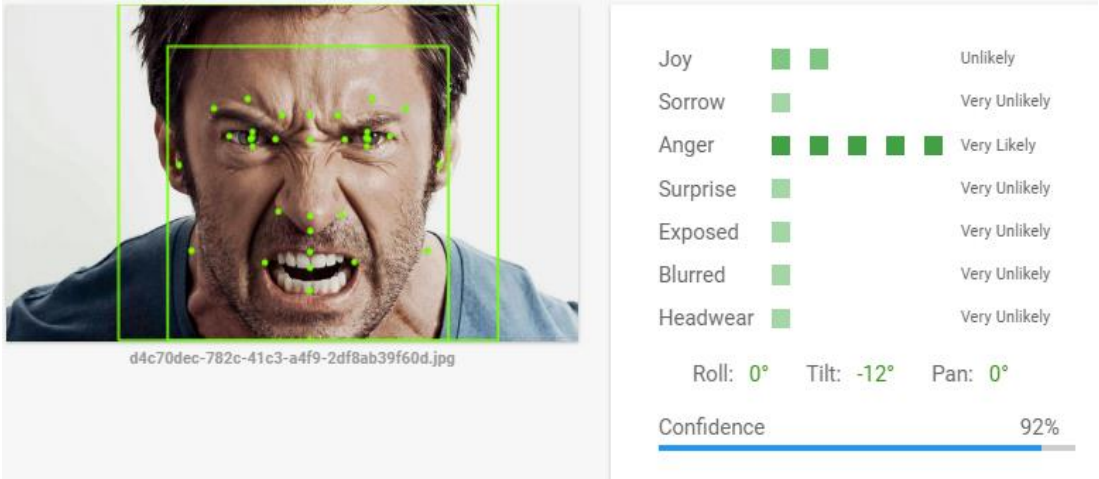

Figura 5.14: Expressão facial de um homem irritado.
Fonte: https://imageshack.us/

Após os testes, todas as expressões faciais foram corretamente classificadas, como pode ser observado nos resultados anteriores. A detecção das expressões de alegria, tristeza e raiva apresentaram um resultado com um valor de confiança elevado. Porém na detecção da expressão de surpresa a confiança obtida foi menor, mesmo tendo reconhecido corretamente a face na imagem, isso pode ser devido à uma certa semelhança na expressão de surpresa com outras expressões faciais como tristeza ou alegria.

5.2. Análise de Consumo

Um dos requisitos do sistema é seu funcionamento ininterrupto de no mínimo 1 hora. Mediu-se o consumo médio de corrente do sistema utilizando-se um amperímetro e foi observado que durante a sua execução e uso, a corrente máxima utilizada, com o WiFi ligado e com a Webcam USB conectada não ultrapassou 1,5 A.

Porém, de acordo com as especificações técnicas encontradas no site oficial da Raspberry Pi, o consumo de energia máximo desta Single Board Computer é de 2,5 A. Assim, para garantir que o sistema funcione sem interrupções e corretamente, considerou-se a saída máxima de corrente disponível para o Powerbank utilizado que é de 2,4 A; um pouco menor que o valor recomendado pelo fabricante, mas que não representou nenhum impedimento durante os testes (visto que a corrente máxima que foi medida não ultrapassou 1,5 A).

De acordo com a **Tabela 4.2**, a capacidade real do powerbank utilizado é 3,7 V / 10000 mAh, assim fornecendo 5 V, e 2400 mA:

$$\text{Horas de funcionamento} = \frac{10000\ \text{mAh} \cdot \frac{3{,}7\ \text{V}}{5\ \text{V}}}{2400\ \text{mA}} = 3{,}08\ \text{horas} \qquad (\mathbf{5.2})$$

Então, o sistema foi dimensionado para funcionar por no mínimo 3 horas, em sua capacidade máxima de processamento. Como todo o processamento bruto é realizado em nuvem (STT, TTS, CV), o sistema consome uma corrente muito menor em sua execução. Durante os testes observou-se que o sistema é capaz de funcionar por 5 horas seguidas. Com este resultado, o requisito de autonomia de funcionamento do sistema foi atendido e superado em 5 vezes.

5.3. Análise de Desempenho de Requisitos do Sistema

A partir dos resultados apresentados nas **Seções 5.1 e 5.2**, referentes aos testes descritos na **Seção 3.1.3** é possível analisar o cumprimento ou não dos requisitos funcionais e não funcionais listados nas **Seções 3.1.1 e 3.1.2**. As **Tabelas 5.17 e 5.18** abaixo apresenta um resumo do cumprimento de tais requisitos:

Tabela 5.17: Cumprimento de requisitos funcionais.

Requisito Funcional	Estado
O sistema deve permitir que o usuário execute comandos de voz para operar suas funcionalidades	Atendido
O sistema deve oferecer ao usuário as funcionalidades de extração de informações visuais de textos impressos ou escritos à mão, características visuais de objetos e do ambiente ao seu redor	Atendido
O sistema deve registrar todas as informações extraídas	Atendido
O sistema deve retornar ao usuário dados em forma de som e sintetizados em voz	Atendido
O sistema deve ser capaz de estabelecer conexão contínua com a internet através de um *'hotspot'*	Atendido

Tabela 5.18: Cumprimento de requisitos não-funcionais.

Requisito Não-Funcional	Estado
O sistema deve ser barato, não ultrapassando o valor R$ 500,00	Atendido
O sistema deve garantir sua usabilidade por pessoas com deficiência visual	Não-Verificado
O sistema deve possuir uma fonte de energia que o atenda ininterruptamente durante no mínimo 1 hora	Atendido
O sistema deve garantir a possibilidade de utilização de suas funções ininterruptamente, respeitando os possíveis limites da fonte de energia	Atendido
O sistema deve exibir resultados de extração de informações com confiabilidade acima de 80%	Atendido
O sistema deve possuir uma latência máxima de 10 segundos, para garantir a sua usabilidade	Atendido
O sistema deve constituir um dispositivo portátil	Parcialmente Atendido

Observa-se que todos os requisitos funcionais foram atendidos, do ponto de vista técnico, o projeto implementado atende ao conceito inicialmente concebido.

O requisito que faz com que o projeto seja acessível foi atendido, a soma dos valores de todos os componentes utilizados no projeto foi de aproximadamente R$ 430,00, se comparado aos valores sugeridos por outros projetos de ponta, como o eSight, referência no desenvolvimento de dispositivos para auxílio a deficientes visuais, de $9,995.00 USD, ou aproximadamente R$ 37.600,00, pode-se dizer que o sistema implementado possuí a acessibilidade desejada, considerando que neste projeto foi computado apenas o valor do protótipo. O eSight não atende pessoas completamente cegas, mas é um sistema que possui alta tecnologia e foi projetado para se focar automaticamente, permitindo aos usuários que possuam certo grau de cegueira enxergar um objeto à curta ou longa distância à vontade, além de oferecer a capacidade de ampliar manualmente a imagem.

Os valores médios de tempo de execução a cada chamada realizada no projeto dependem da largura de banda disponível e não ultrapassaram 7 segundos, respeitando a latência máxima de 10 segundos. A latência apresentada no processo de texto para fala utilizado neste projeto não ultrapassou 1000 ms e o processo sempre foi realizado com sucesso.

Apesar de o sistema apresentar uma autonomia de funcionamento considerável, o peso e a dimensão acrescentada pelo uso do Powerbank de 10000 mAh impossibilitou a implementação de um dispositivo Wearable, que constituía uma das ideias inicias do projeto. O sistema é portátil, mas não possui o conceito de praticidade que foi inicialmente proposto.

Um dos requisitos não-funcionas, mas, um dos mais importantes do presente trabalho, que não foi verificado foi a usabilidade do sistema desenvolvido por um deficiente visual. Algo a ser pensado em futuros projetos que façam uso desta monografia.

Outra ideia que surgiu ao final da execução, mas que não houve tempo hábil de ser implementada, foi a de o sistema constituir um módulo que fosse capaz de detectar obstáculos próximos do indivíduo que o utilize.

Uma consideração importante a se fazer, é a implementação de uma REST API em Flask que não foi alvo de testes durante os resultados, porém, a partir dos dados que foram gravados por este módulo do projeto é que se obteve os dados referentes a tais resultados.

6. CONCLUSÕES

Este projeto teve como objetivo a aplicação de conceitos fundamentados durante o curso de Engenharia Elétrica – Ênfase em Eletrônica a um problema real que talvez ainda não tenho recebido a merecida atenção de profissionais da área: a adoção e o uso de tecnologias atuais e acessíveis para inclusão de pessoas com deficiência, mais especificamente, para a inclusão de pessoas com deficiência visual. Os resultados do trabalho desenvolvido mostram que as APIs da Google Cloud utilizadas: Speech Recognition, Text-to-Speech e Vision, são ferramentas poderosas no campo de reconhecimento de fala, síntese de voz e visão computacional, respectivamente. Neste capitulo será feita a conclusão do desenvolvimento do presente trabalho.

Apesar de as APIs utilizadas terem se mostrado eficientes para o propósito deste projeto, as ferramentas que executam o processo de fala para texto e de classificação de imagens apresentaram algumas limitações. A primeira limitação é o fato intrínseco de que todas as três APIs utilizadas foram implementadas para operar em nuvem, sendo obrigatória a conexão com a Internet. Isso fez com que, devido às limitações de custo, o sistema desenvolvido sempre necessite um celular que possua Internet móvel com um *'hotspot'* configurado para que a Raspberry Pi se conecte à Internet.

A conexão TCP utilizada pelo processamento em nuvem da Google Cloud visa fornecer a máxima precisão, porém estes aplicativos oferecem um resultado preciso no custo do atraso. A qualidade da experiência dos usuários pode ser afetada caso haja uma diminuição da largura de banda na conexão, que representa outro ponto crítico da implementação.

Uma dúvida que surgiu durante a execução do trabalho foi: já que há a necessidade do uso de um smartphone para seu funcionamento, porque não utilizar um smartphone na implementação de todo o sistema, já que ele possui todos os hardwares necessários para o mesmo? Talvez haja essa possibilidade em algum trabalho futuro. Mas, a princípio, na idealização do sistema, o objetivo era o desenvolvimento de um dispositivo *'wearable'*, que fosse complementar os poucos aplicativos que existem para auxiliar deficientes visuais disponíveis nos smartphones atuais. Como foi explicitado nos resultados, não foi possível constituir um dispositivo *'wearable'* devido ao peso e volume de todo o hardware utilizado.

Outro ponto negativo da utilização das APIs do Google Cloud é a limitação da gratuidade de sua utilização, como um dos principais objetivos do projeto é o de inclusão digital e acessibilidade, atribuir um preço para a utilização deste tipo de sistema iria contra este propósito.

Apesar de todas as limitações constatadas, houve vários pontos positivos que podem ser destacados no projeto. Os comandos por voz, utilizando a classificação de textos se mostrou versátil e genérico o suficiente para entender o comando executado pelo usuário, além disso, o processo de transcrição da fala para texto surpreendeu na quantidade de acertos em ambiente controlado, sem ruídos.

A leitura de textos, que é uma funcionalidade importante no sistema obteve ótimos resultados para textos digitalizados e manuscrito em letra de forma, e um resultado aceitável para texto manuscrito em letra cursiva. Com esta funcionalidade é possível a leitura de rótulos de produtos, folhetos, livros, cupons fiscais, dinheiro e até bulas de remédio. A descrição de imagens, apesar de ser difícil definir um nível de confiança para a extração das informações mais relevantes, apresentou-se útil, sendo possível descrever genericamente ambientes e objetos, expressões faciais de pessoas, logos e pontos de referência.

Mesmo com resultados promissores, apesar das limitações levantadas, um ponto importante do trabalho seria a validação do projeto com o seu principal público alvo: os deficientes visuais. Novamente, fica a deixa para um futuro trabalho. Sabe-se que o avanço de tecnologias assistivas dependem de muitos atores, incluindo provedores de serviços governamentais, educadores, empregadores, profissionais de desenvolvimento e da indústria de inovações tecnológicas. Os esforços devem se concentrar em aumentar sua conscientização e na implementação de ambientes digitais livres de barreiras e inclusivos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] The World Bank, "World Development Report 2016: Digital Dividends," Relatório Técnico, Washington DC - EUA, 2016.

[2] ITU (International Telecommunication Union), "Home Computer Use and the Development of Human Capital," World Telecommunication/ICT Indicators Database, ITU, Geneva, 2016.

[3] ITU (International Telecommunication Union), "Measuring the Information Society Report 2016," Quarterly Journal of Economics 126, ITU, Geneva, 2016.

[4] World Health Organization, "World Report on Disability," The World Bank, Geneva, Switzerland, 2011.

[5] Secretaria Especial dos Direitos Humanos, "Técnologia Assistiva," 2009. [Online]. Available: http://www.pessoacomdeficiencia.gov.br/app/sites/default/files/publicacoes/livro-tecnologia-assistiva.pdf. [Acesso em 27 abril 2018].

[6] D. Mulfar, A. L. Minnolo e A. Puliafito, "Wearable Devices and IoT," em *IEEE*, University of Messina, Italy, 2017.

[7] P. Mell e T. Grance, "The NIST Definition of Cloud Computing," NIST - National Institute of Standards and Technology, U.S. Department of Commerce, 2011.

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin e M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Electrical Engineering and Computer Sciences, University of California, Berkeley, Califórnia - EUA, 2009.

[9] X. D. Huang, Y. Ariki e M. A. Jack, Hidden Markov Models for Speech Recognition, Edinburgh University: R. State of the Art in Speech. Workshop in Integrating Speeching and Natural Language. University College Dublin, 1992, 1990.

[10] N. B. Yoma, “Large Vocabulary SpeakerIndependent Continuous Speech Recognition: The SPHINX System,” Department of Computer Science, Carnegie Mellon University, 1988.

[11] L. C. Sousa, “Adaptação de Locutor de Sistemas de Reconhecimento de Fala,” Dissertação de Mestrado, UNICAMP, SP, 2004.

[12] W. L. Hosch, “Machine learning - Artificial Intelligence,” Encyclopedia Britannica, [Online]. Available: https://global.britannica.com/technology/machine-learning. [Acesso em 10 Abril 2018].

[13] P. Simon, Too Big to Ignore: The Business Case for Big Data, 2013.

[14] R. Kohavi e F. Provost, “Machine Learning,” em *Glossary of terms*, Netherlands, Kluwer Academic Publishers, 1998, pp. 271-274.

[15] A. Graves, A. R. Mohamed e G. Hinton, “Speech recognition with deep recurrent neural networks,” em *IEEE*, 2013.

[16] A. Krizhevsky, I. Sutskever e G. E. Hinton, “Imagenet classication with deep convolutional neural networks,” *Proc. Adv. Neural Inf. Process. Syst.,* pp. 1097-1105, 2012.

[17] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer e R. Harshman, “Indexing by latent semantic analysis,” *J. Amer. Soc. Inf.,* vol. 41, nº 6, p. 391, 1990.

[18] B. Pang e L. Lee, “Opinion mining and sentiment analysis,” *Found. Trends Inf. Retr.,* vol. 2, nº 1, pp. 1-135, 2008.

[19] J. R. Parker, Algorithms for Image Processing and Computer Vision, Indianápolis, Indiana - EUA: Wiley Publishing , 2011.

[20] “TapTapSee,” [Online]. Available: https://taptapseeapp.com/. [Acesso em 02 Abril 2018].

[21] “BeMyEyes,” [Online]. Available: https://www.bemyeyes.com/. [Acesso em 02 Abril 2018].

[22] M. Avila, K. Wolf, A. Brock e N. Henze, “Remote Assistance for Blind Users in Daily Life: A Survey about Be My Eyes,” em *The 9th ACM International Conference on Pervasive Technologies Related to Assistive Environments - PETRA 16*, Corfu Island, Grécia, 2016.

[23] H. Kwak e J. An, “Revealing the Hidden Patterns of News Photos: Analysis of Millions of News Photos through GDELT and Deep Learning-Based Vision APIs,” em *The Workshops of the Tenth International AAAI Conference on Web and Social Media*, Qatar, 2016.

[24] “eSight: helping the legally blind see,” [Online]. Available: https://www.investinontario.com/success-stories/esight-helping-legally-blind-see. [Acesso em 10 Abril 2018].

[25] “Glasses from eSight help legally blind Colts fan see game for first time,” [Online]. Available: https://www.si.com/tech-media/2017/01/05/esight-glasses-legally-blind-indianapolis-colts-fan. [Acesso em 05 Abril 2018].

BIBLIOGRAFIA CONSULTADA

[26] "Projeto de Auxílio às Pessoas com Deficiência Visual Vence Concurso Intel de Tecnologia," [Online]. Available: http://napsol.icmc.usp.br/pt-br/node/272. [Acesso em 05 Abril 2018].

[27] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang e P. Bahl, "Anatomizing application performance differences on smartphones," ACM Mobile systems, applications, and services, 2010.

[28] "Levenshtein distance," [Online]. Available: http://rosettacode.org/wiki/Levenshtein_distance. [Acesso em 05 Abril 2018].

[29] A. Porter, M. Muztoba e U. Y. Ogras, "Human - Machine Communication for Assistive loT Technologies," em *IEEE*, 2016.

[30] I. Brandié, "Democratization in Science and Technology through Cloud Computing," em *IEEE*, 2014.

[31] D. Mulfari, A. Celesti, M. Fazio, M. Villari e A. Puliafito, "Using Google Cloud Vision in Assistive Technology Scenarios," em *IEEE*, 2016.

[32] D. Pavel, V. Callaghan e A. K. Dey, "Democratization of health care through selfmonitoring," em *IEEE*.

[33] D. Mulfari, A. L. Minnolo e A. Puliafito, "Wearable Devices and IoT as Enablers of Assistive Technologies," em *IEEE*, 2017.

[34] H. Nigel, "The Designer and Development of Assistive Technology," em *IEEE*, 2017.

[35] A. Hassan e A. Mahmood, "Convolutional Recurrent Deep Learning Model for Sentence Classification," em *IEEE*, 2018.

[36] H. Hosseini, B. Xiao e R. Poovendran, "Google's Cloud Vision API Is Not Robust To Noise," em *IEEE*, 2017.

[37] M. Dascalu, A. Moldoveanu, O. Balan, R. G. Lupu, F. Ungureanu e S. Caraiman, "Usability Assessment of Assistive Technology for Blind and Visually Impaired," em *IEEE*, 2017.

[38] M. Assefi, M. Wittie e A. Knight, "Impact of Network Performance on Cloud Speech Recognition," em *IEEE*, 2015.

[39] Z. Liu, Y. Luo, J. Cordero, N. Zhao e Y. Shen, "Finger - Eye: A Wearable Text Reading Assistive System for the Blind and Visually Impaired," em *IEEE*, Angkor Wat, Cambodia, 2016.

[40] T. V. Mataró, F. Masulli, S. Rovetta, A. Cabri, C. Traverso, E. Capris e S. Torretta, "An Assistive Mobile System Supporting Blind and Visual Impaired People when are Outdoor," em *IEEE*, 2017.

[41] G. B. Quispe, J. M. Figueroa, F. P. Mansilla e J. R. Humaní, "A Friendly Speech User Interface based on Google Cloud Platform to Access a Tourism Semantic Website," em *IEEE*, 2017.

APÊNDICES

Laço principal **main.py**:

```python
#!/usr/bin/env python3

import os
import urllib.request
import speech_recognition as sr
from gcp.recognition import Recognition, Noise
from gen.play import Play
from gcp.speak import Speak
from gen.snapshot import Snapshot
from gcp.vision import Vision
from nlp.classifier import Classification
from nlp.train import main as Train
from who.speaker import Speaker
from time import sleep

def main():
    Train()
    Noise(duration=10)

    r = sr.Recognizer()

    first = True
    while True:
        try:
            with urllib.request.urlopen('http://www.google.com') as url:
                url.read()
            break
        except:
            if first == True:
                Play('internet')
            continue
        first = False
        sleep(5)

    Play('start')
    print('Fale')

    while True:
        with sr.Microphone(device_index=3) as source:
            audio = r.listen(source)
        name = Speaker()

        if name != 'Arian':
            continue

        speech = Recognition(audio)
        if type(speech) is str:
            classs = Classification(speech, 'keyword')

            if classs == 'keyword':
                classs = Classification(speech, 'tipo')
                if classs != None:
                    Play('success')
                    image = Snapshot()
                    Play('camera')
                    response = Vision(classs, image)
                    Speak(response)
                else:
                    print('Erro')
                    Play('error')
```

```
            elif classs == 'train':
                Play('name')
                while True:
                    speech = Recognition(audio)
                    if type(speech) is str:
                        break
                os.system("who/train.py 1")
            elif classs == 'shutdown':
                call(['shutdown', 'now'], stdout=open(os.devnull, 'wb'))

        else:
            print('Erro')
            Play('error')


if __name__ == '__main__':
    main()
```

/gcp/recognition.py:

```
#!/usr/bin/env python3

import speech_recognition as sr
import os


def Recognition(audio):
    with open(os.environ["GOOGLE_APPLICATION_CREDENTIALS"]) as f:
        credentials = f.read()

    r = sr.Recognizer()

    try:
        return(r.recognize_google_cloud(audio, language="pt-BR",
credentials_json=credentials).rstrip())
    except sr.UnknownValueError:
        return(404)
    except sr.RequestError as e:
        print(e)
        return(400)


def Noise(duration=1):
    r = sr.Recognizer()
    with sr.Microphone(device_index=3) as source:
        r.adjust_for_ambient_noise(source, duration)
```

/gcp/speak.py:

```
#!/usr/bin/env python3

import os
from gtts import gTTS
from subprocess import call


def Speak(text):
    audio = gTTS(text=text, lang='pt')

    audio.save('/tmp/audio.mp3')
```

```
12      call(['omxplayer', '-o', 'local', '/tmp/audio.mp3'],
    stdout=open(os.devnull, 'wb'))
```

/gcp/vision.py:

```
1   #!/usr/bin/env python3
2
3   import json
4   from google.cloud import vision
5   from google.cloud.vision import types
6   from google.protobuf.json_format import MessageToJson as toJSON
7   from maz.translate import Translation
8
9
10  def Vision(typee, path):
11      client = vision.ImageAnnotatorClient()
12
13      file_name = path
14      with open(file_name, 'rb') as image_file:
15          content = image_file.read()
16
17      image = types.Image(content=content)
18
19      result = ''
20      result_en = ''
21
22      if typee == 'label':
23          response = client.label_detection(image=image)
24          labels = response.label_annotations
25          if len(labels) != 0:
26              for label in labels:
27                  if label.score >= 0.75:
28                      result_en += label.description + ', '
29              result = Translation(result_en)
30          else:
31              result = 'Não foi possivel classificar a imagem'
32
33      elif typee == 'text':
34          response = client.text_detection(image=image)
35          if len(response.text_annotations != 0):
36              result = response.text_annotations[0].description.replace('\n',
    '')
37          else:
38              result = 'Não foram detectados textos'
39
40      elif typee == 'face':
41          response = json.loads(toJSON(client.face_detection(image=image)))
42          if len(response['faceAnnotations']) != 0:
43              result = 'A imagem mostra a face de uma pessoa. Ela está '
44              result += 'Alegre: %s;' %
    response['faceAnnotations'][0]['joyLikelihood']
45              result += 'Chateada: %s;' %
    response['faceAnnotations'][0]['sorrowLikelihood']
46              result += 'Irritada: %s;' %
    response['faceAnnotations'][0]['angerLikelihood']
47              result += 'Surpresa: %s;' %
    response['faceAnnotations'][0]['surpriseLikelihood']
48              result.replace('LIKELIHOOD_UNSPECIFIED', 'Probabilidade não
    especificada')
49              result.replace('VERY_UNLIKELY', 'Muito improvável')
50              result.replace('UNLIKELY', 'Improvável')
51              result.replace('POSSIBLE', 'Possível')
52              result.replace('VERY_LIKELY', 'Muito provável')
```

```
            result.replace('LIKELY', 'Provável')

        else:
            result = 'Não foram detectados rostos'

    elif typee == 'logo':
        response = client.logo_detection(image=image)
        logos = response.logo_annotations
        if len(logos) != None:
            for logo in logos:
                result_en += logo.description + ', '
            result = Translation(result_en)
        else:
            result = 'Não foram detectados logos'

    elif typee == 'landmark':
        response = client.landmark_detection(image=image)
        landmarks = response.landmark_annotations
        if len(landmarks) != None:
            for landmark in landmarks:
                result_en += landmark.description + ', '
            result = Translation(result_en)
        else:
            result = 'Não foram detecatos marcos'

    else:
        result = 'Nenhum comando foi reconhecido'

    return(result)
```

/gen/play.py:

```
#!/usr/bin/env python3

from subprocess import call
import os


def Play(sound):
    call(['omxplayer', '-o', 'local', 'gen/sounds/%s.mp3' % sound],
stdout=open(os.devnull, 'wb'))
#!/usr/bin/env python3
```

/gen/sounds.py:

```
#!/usr/bin/env python3

from gtts import gTTS
import os


def Save(text, name):
    audio = gTTS(text=text, lang='pt')

    audio.save('/root/virtualenv/embraer/root/gen/sounds/%s.mp3' % name)

def main():
    Save('Programa iniciado', 'start')
    Save('Encerrando', 'closing')
    Save('Dados obtidos com sucesso', 'get')
    Save('Diga o seu nome', 'name')
```

```
    Save('Diga ok google', 'key')
    Save('Entendi uma frase diferente, repita', 'repeat')
    Save('Cancelando', 'cancel')
    Save('O sistema não está conectado à internet', 'internet')


if __name__ == '__main__':
    main()
```

/gen/snapshot.py:

```
#!/usr/bin/env python3

from time import time
from subprocess import call


def Snapshot():
    dt = time()
    call(['fswebcam', '-d','/dev/video0', '--jpg', '9', '-r', '1280x720', '--no-banner', '/root/virtualenv/embraer/root/gen/images/%s.jpg' % dt])
    return ('/root/virtualenv/embraer/root/gen/images/%s.jpg' % dt)
```

/nlp/classifier.py:

```
#!/usr/bin/env python3

import nltk
import re
import yaml
from nltk.stem import RSLPStemmer


def LoadMemory(filename):
    fileW = open("nlp/files/"+filename+"_words.nlp", 'r')
    words = fileW.read()
    words = yaml.load(words)
    return words


def Calculate_Class_Score(sentence, class_name, corpus_words):
    expression = '[!-@[-`{-¿ÆÐÑ×ØÝ-ßä-æëðñö-øý-ÿ]'
    stemmer = RSLPStemmer()
    score = 0
    sentence = re.sub(expression, '', sentence)
    sentence = nltk.word_tokenize(sentence)
    for word in sentence:
        if stemmer.stem(word.lower()) in corpus_words[class_name]:
            score += corpus_words[class_name][stemmer.stem(word.lower())]
    return score


def Classification(sentence, filename):
    corpus_words = LoadMemory(filename)

    high_class = None
    high_score = 0
    for class_name in list(corpus_words.keys()):
        score = Calculate_Class_Score(sentence, class_name, corpus_words)
        if score > high_score:
            high_class = class_name
```

```
            high_score = score

    return high_class
```

/nlp/train.py:

```
#!/usr/bin/env python3

from nlp.trainer import Learning
import requests


def Train_Keyword():
    filename = "keyword"

    Delete("nlp/files/"+filename+"_words.nlp")
    Delete("nlp/files/"+filename+"_sentences.nlp")

    training_data = []

    training_data.append({"class":"keyword", "sentence":"ok google"})
    training_data.append({"class":"keyword", "sentence":"focus"})
    training_data.append({"class":"keyword", "sentence":"foco"})

    Learning(training_data, filename)

def Train_Train():
    filename = "train"

    Delete("nlp/files/"+filename+"_words.nlp")
    Delete("nlp/files/"+filename+"_sentences.nlp")

    training_data = []

    training_data.append({"class":"train", "sentence":"treinar"})
    training_data.append({"class":"train", "sentence":"treino"})
    training_data.append({"class":"train", "sentence":"treinamento"})

    Learning(training_data, filename)

def Train_Shutdown():
    filename = "shutdown"

    Delete("nlp/files/"+filename+"_words.nlp")
    Delete("nlp/files/"+filename+"_sentences.nlp")

    training_data = []

    training_data.append({"class":"train", "sentence":"shutdown"})
    training_data.append({"class":"train", "sentence":"desligar"})
    training_data.append({"class":"train", "sentence":"encerrar"})

    Learning(training_data, filename)

def Train_Tipo():
    filename = "tipo"

    Delete("nlp/files/"+filename+"_words.nlp")
    Delete("nlp/files/"+filename+"_sentences.nlp")

    training_data = []

    training_data.append({"class":"label", "sentence":"ambiente"})
```

```
    training_data.append({"class":"label", "sentence":"objeto"})
    training_data.append({"class":"label", "sentence":"descrever"})
    training_data.append({"class":"label", "sentence":"descrição"})
    training_data.append({"class":"label", "sentence":"paisagem"})

    training_data.append({"class":"face", "sentence":"face"})
    training_data.append({"class":"face", "sentence":"rosto"})
    training_data.append({"class":"face", "sentence":"humor"})
    training_data.append({"class":"face", "sentence":"pessoa"})
    training_data.append({"class":"face", "sentence":"cara"})

    training_data.append({"class":"text", "sentence":"texto"})
    training_data.append({"class":"text", "sentence":"manuscrito"})
    training_data.append({"class":"text", "sentence":"mensagem"})
    training_data.append({"class":"text", "sentence":"leitura"})
    training_data.append({"class":"text", "sentence":"leia"})

    training_data.append({"class":"text", "sentence":"marco"})
    training_data.append({"class":"text", "sentence":"ponto de referência"})

    training_data.append({"class":"text", "sentence":"logo"})
    training_data.append({"class":"text", "sentence":"rótulo"})

    Learning(training_data, filename)


def Delete(filename):
    with open(filename, "w"):
        pass


def main():
    Train_Keyword()
    Train_Train()
    Train_Shutdown()
    Train_Tipo()


if __name__ == "__main__":
    main()
```

/nlp/trainer.py:

```
#!/usr/bin/env python3

import nltk
import re
import yaml
from nltk.stem import RSLPStemmer


def LoadMemory(filename):
    fileW = open("nlp/files/"+filename+"_words.nlp", 'r')
    words = fileW.read()
    fileW.close()
    words = yaml.load(words)
    return words


def LoadSentences(filename):
    fileE = open("nlp/files/"+filename+"_sentences.nlp", 'r')
    sentences = fileE.read()
    fileE.close()
```

```
    return sentences


def SaveMemory(words, filename):
    fileW = open("nlp/files/"+filename+"_words.nlp", 'w')
    fileW.write(str(words))
    fileW.close()


def SaveSentences(sentences, filename):
    fileE = open("nlp/files/"+filename+"_sentences.nlp", 'a')
    fileE.write(sentences + "\n")
    fileE.close()


def Learning(training_data, filename):
    stemmer = RSLPStemmer()
    expression = '[!-@[-`{-¿ÆÐÑ×ØÝ-ßä-æëðñö-øý-ÿ]'
    corpus_words = LoadMemory(filename)

    for data in training_data:
        loaded = LoadSentences(filename)
        sentences = data['sentence']
        sentences = re.sub(expression, '', sentences)
        sentences = stemmer.stem(sentences.lower())

        if sentences in loaded:
            continue

        SaveSentences(sentences, filename)
        sentence = nltk.word_tokenize(sentences)
        class_name = data['class']
        if corpus_words is None:
            corpus_words = {}
            corpus_words[class_name] = {}
        else:
            if class_name not in list(corpus_words.keys()):
                corpus_words[class_name] = {}
        for word in sentence:
            if word not in list(corpus_words[class_name].keys()):
                corpus_words[class_name][word] = 1
            else:
                corpus_words[class_name][word] += 1

    SaveMemory(corpus_words, filename)
```

/api/app.py

```
#!/usr/bin/env python3

from flask import Blueprint
from flask_restful import Api
from resources.personagem import Personagem_Resource, Personagem_Spoiler


api_bp = Blueprint('api', __name__)
api = Api(api_bp)


api.add_resource(Personagem_Resource, '/Personagem')
api.add_resource(Personagem_Spoiler, '/Personagem/Spoiler')
```

/who/speaker.py

```
#!/usr/bin/env python3

from piwho import recognition


def Speaker(audio):
    recog = recognition.SpeakerRecognizer()
    nome = recog.identify_speaker("rec.wav")
    return nome
```

/who/train.py

```
#!/usr/bin/env python3

import os
import sys
import speech_recognition as sr
from piwho import recognition
from gen.play import Play
from gcp.speak import Speak
from nlp.classifier import Classification
from subprocess import call


def main(name):
    recog = recognition.SpeakerRecognizer()
    recog.speaker_name = name
    r = sr.Recognizer()

    Speak('Treinamento para %s' % name)
    Play('key')

    with open(os.environ['GOOGLE_APPLICATION_CREDENTIALS']) as f:
        credentials = f.read()

    while True:
        with sr.Microphone(device_index=3) as source:
            audio = r.listen(source)

        try:
            speech = (r.recognize_google_cloud(audio, language='pt-BR',
credentials_json=credentials).rstrip())
            classe = Classification(speech, 'keyword')
            if classe == 'keyword':
                with open('%s.wav' % dt, "wb") as f:
                    f.write(audio.get_wav_data())
                recog.train_new_data()
                break
            elif classe == 'cancelar':
                break
            else:
                Play('repeat')

        except:
            Play('error')


if __name__ == "__main__":
    main(sys.argv[0])
```

/api/config.py

```
#!/usr/bin/env python3

import os
import credentials


basedir = os.path.abspath(os.path.dirname(__file__))
SQLALCHEMY_ECHO = False
SQLALCHEMY_TRACK_MODIFICATIONS = True
SQLALCHEMY_DATABASE_URI =
"postgresql://%s:%s@localhost/db" %(credentials.dbUser,
credentials.dbPassword)
```

/api/app.py

```
#!/usr/bin/env python3

from flask import Blueprint
from flask_restful import Api
from resources.speech import Speech_Resource
from resources.vision import Vision_Resource


api_bp = Blueprint('api', __name__)
api = Api(api_bp)


api.add_resource(Speech_Resource, '/Speech')
api.add_resource(Vision_Resource, '/Vision')
```

/api/migrate.py

```
#!/usr/bin/env python3

from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand
from model import db
from run import create_app

app = create_app('config')

migrate = Migrate(app, db)
manager = Manager(app)
manager.add_command('db', MigrateCommand)


if __name__ == '__main__':
    manager.run()
```

/api/run.py

```
#!/usr/bin/env python3

from flask import Flask


def create_app(config_filename):
```

```
    app = Flask(__name__)
    app.config.from_object(config_filename)

    from app import api_bp
    app.register_blueprint(api_bp, url_prefix='/api')

    from model import db
    db.init_app(app)

    return app


if __name__ == "__main__":
    app = create_app("config")
    app.run(debug=True, host='0.0.0.0')
```

/api/model.py

```
#!/usr/bin/env python3

from flask import Flask
from marshmallow import Schema, fields, pre_load, validate
from flask_marshmallow import Marshmallow
from flask_sqlalchemy import SQLAlchemy
from time import time


ma = Marshmallow()
db = SQLAlchemy()


class Speech(db.Model):
    __tablename__ = 'speech'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    audio = db.Column(db.String(100), nullable=False)
    speech = db.Column(db.String(100), nullable=False)
    date = db.Column(db.Integer(15), nullable=False)

    def __init__(self, audio, speech):
        self.audio = audio
        self.speech = speech
        self.date = time()


class Speech_Schema(ma.Schema):
    id = fields.Integer()
    audio = fields.String()
    speech = fields.String()
    date = fields.Integer()


class Vision(db.Model):
    __tablename__ = 'vision'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    image = db.Column(db.String(100), nullable=False)
    description = db.Column(db.String(100), nullable=False)
    date = db.Column(db.Integer(15), nullable=False)

    def __init__(self, image, description):
        self.image = image
        self.description = description
        self.date = time()

```

```
46
47  class Vision_Schema(ma.Schema):
48      id = fields.Integer()
49      image = fields.String()
50      description = fields.String()
51      date = fields.Integer()
```