UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

# Sistema de comandos e identificação de voz

Autor: Tiago Rafael Giorgetti Landim

Orientador: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2017

**TIAGO RAFAEL GIORGETTI LANDIM**

# SISTEMA DE COMANDOS E IDENTIFICAÇÃO DE VOZ

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

Curso de Engenharia Elétrica com ênfase em Eletrônica

ORIENTADOR: Prof. Dr. Evandro Luis Linhari Rodrigues

São Carlos

2017

Landim, Tiago Rafael Giorgetti
L257s Sistema de comandos e identificação de voz / Tiago Rafael Giorgetti Landim; orientador Evandro Luis Linhari Rodrigues. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. Reconhecimento de voz. 2. Raspberry Pi. 3. Identificação de voz. 4. Linux embarcado. I. Título.

# FOLHA DE APROVAÇÃO

Nome: Tiago Rafael Giorgetti Landim

Título: "Sistema de comandos e identificação de voz"

Trabalho de Conclusão de Curso defendido e aprovado em 22 / 06 / 2017,

com NOTA 7,3 ( sete , três ), pela Comissão Julgadora:

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

*Prof. Dr. Maximiliam Luppe - SEL/EESC/USP*

*Mestre André Luís Martins - Doutorando - SEL/EESC/USP*

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado José Carlos de Melo Vieira Júnior

# RESUMO

Este projeto visa a implementação de um sistema capaz de compreender comandos de voz e executar as tarefas correspondentes. O processo de reconhecimento de voz foi implementado com base em um módulo que utiliza o conversor de fala para texto chamado Wit.ai. Para a coversão de texto-para-fala, foi utilizado o sintetizador de voz Flite. O sistema foi desenvolvido em Raspberry Pi e permite a execução das seguintes tarefas: acionamento de lâmpadas, gravação de áudios, captura de imagens, escrita e leitura de textos, buscas na internet e identificação de voz. Todos os comandos foram desenvolvidos na linguagem de programação Python. Uma página na *web* foi desenvolvida para permitir o acesso aos dados armazenados pelo sistema. A identificação da voz do usuário permite que acessos indevidos ao sistema sejam reportados. Nota-se que o número de amostras treinadas resulta em melhorias na identificação da voz. Os mecanismos de reconhecimento de fala são limitados, o Wit.ai necessita de conexão à internet e seu processamento é inferior quando comparado a outros serviços pagos.

**Palavras-chave:** Reconhecimento de voz. Raspberry Pi. Identificação de voz. Linux embarcado.

# ABSTRACT

This project aims the implementation of a system capable of understanding voice commands and executing the corresponding tasks. The voice recognition process was implemented based in a module which uses a speech-to-text converter called Wit.ai. For the texto-to-speech conversion, was used the Flite synthesizer engine. The system was developed in Raspberry Pi and, permits the execution of the following tasks: lamps activation, audio recording, images capture, text writing and reading, searchs on the internet and voice identification. All the commands were developed in Python programming language. A webpage was developed to allow the access to all the recording data stored by the system. The user's voice recognition permits that improper acces to the system be reported. It is noted that number of samples directly results in an improvement in the voice identification. The voice recognition engines are limited, the Wit.ai needs internet conection and its processing is inferior when compared to others services that require subscription.



**Keywords:** Voice recognition. Raspberry Pi. Voice Recognition. Embebbed Linux.

# Sumário

## Lista de Figuras

## Lista de Gráficos

## Lista de Tabelas

# 1. Introdução

Novas tecnologias surgem constantemente de forma a facilitar a vida dos usuários e aumentar o conforto nos diversos ambientes. Em conjunto com estas tendências, a Internet das Coisas (IoT, do inglês, *Internet of Things*) vem ganhando cada vez mais espaço no mundo atual, permitindo a conexão entre os equipamentos e dispositivos do ambiente. A automação residencial está diretamente relacionada neste contexto, permitindo o controle sobre o ambiente a que está inserido, determinando o comportamento dos dispositivos ao seu redor.

Intrínseco com o desenvolvimento da humanidade, está a capacidade de comunicação entre os indivíduos por meio da fala. O uso de computadores está totalmente ligado à realidade moderna, desta forma a união de máquinas e comunicação por voz é a etapa a ser expandida pelo mundo. Pesquisas têm sido realizadas na área do reconhecimento de voz desde as décadas de 50 e 60 (SEABROOK, J, 2008), propondo tanto algoritmos para o reconhecimento da fala em si como nas suas aplicações. As grandes empresas desenvolvedoras de tecnologia da atualidade possuem seus próprios sistemas de assistente pessoal no mercado. Esses assistentes são comandados pela voz do usuário, reconhecendo o comando ordenado.

Diversas são as empresas que seguem o ramo de automação residencial, produzindo sistemas que permitem ao usuário o controle do ambiente, integrando todos os seus acionamentos, de forma a proporcionar o maior conforto.

Dentro deste meio de desenvolvimento, as grandes companhias da área de tecnologia estão presentes no ramo de controle por voz.

Presente nos sistemas da Apple, a assistente pessoal inteligente Siri teve sua origem no instituto de pesquisa *SRI International* e foi co-fundada por Dag Kittlaus, Adam Cheyer e Tom Gruber (BOOSKER, B, 2013). Não demorou muito tempo para que outra gigante no ramo de tecnologia, a Microsoft, lançasse sua assistente digital, Cortana, em 2014, inicialmente para Windows Phone 8.1 e atualmente presente nos dispositivos com Windows 10 (ASH, M, 2015).

Mais recentemente, no final de 2016, Mark Zuckerberg, o CEO do Facebook, divulgou que tem desenvolvido um sistema de inteligência artificial, controlado por voz, que pudesse se conectar, comunicar e controlar sua casa.

O controle por voz demanda menos tempo do usuário, não há a necessidade de pegar um dispositivo e buscar a opção que realiza a função desejada, o usuário precisa apenas falar.

## 1.1. Motivação

A possibilidade de conversar com um sistema que atenda às necessidades dos usuários está na mente das pessoas, tendo sido intesificada pelos filmes de ficção científica. Na atualidade as maiores empresas do ramo de tecnologia têm buscado adentrar na área de assistentes pessoais e automação residencial por meio do reconhecimento de comandos por voz, trazendo ao mundo uma solução que apenas se imaginava como uma idéia futurística.

## 1.2. Objetivo

O presente projeto propõe o desenvolvimento de um sistema que realiza tarefas baseado no reconhecimento comandos de voz.. Sendo possível a identificação dos usuários préviamente cadastrados por meio da voz. Por fim, os arquivos gravados são exibidos em uma página na web.

## 1.3. Justificativa

Levando em consideração as tendências na área de tecnologia, viu-se a necessidade de seguir as pesquisas desenvolvidas e os atuais lançamentos das grandes empresas. O projeto foi desenvolvido utilizando a plataforma Raspberry Pi, Linux embarcado, uma *webcam* USB com microfone para recepção do áudio e, um conjunto de auto falantes para responder a cada comando.

## 1.4. Organização do trabalho

O projeto foi dividido em quatro capítulos. No primeiro capítulo é apresentado as características da Raspberry Pi, uma explicação sobre Linux embarcado, a linguagem de programação Python, o processo de reconhecimento de voz, o processo de conversão de texto para voz e o mecanismo de identificação de voz. O segundo capítulo apresenta as bibliotecas e módulos empregados no desenvolvimento do projeto. São também apresentados neste capítulo as configurações iniciais da placa e do sistema. O terceiro capítulo dedicado aos programas desenvolvidos para este projeto. Por fim, o capítulo quatro trata dos resultados do sistema, o funcionamento dos códigos desenvolvidos para a realização de comandos. Neste capítulo é também analisado a identificação de vozes.

# 2. Embasamento Teórico

## 2.1. Raspberry Pi

Raspberry Pi é um mini-computador desenvolvido pela Fundação Raspberry Pi. O modelo B foi escolhido para utilização. Este possui arquitetura ARMv6(32-bits) com 512 MB de memória RAM, duas portas USB, uma porta Ethernet de 100mb e entrada de vídeo HDMI.

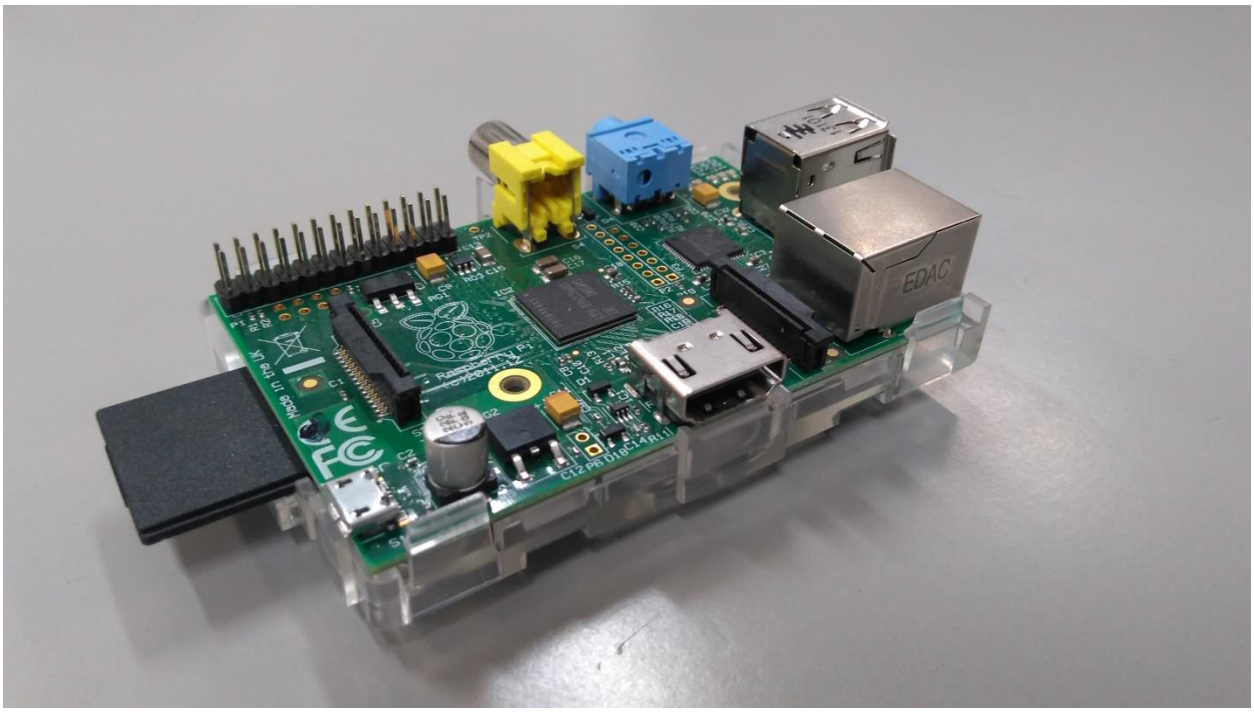


**Figura 1: Raspberry Pi Modelo B**

## 2.2. Linux Embarcado

Sistemas embarcados são aqueles em que um computador é completamente dedicado ao sistema ou dispositivo que ele controla. Por sua vez, Linux embarcado possui algumas vantagens em relação ao Linux tradicional: melhor interface com o usuário utilizando navegador e servidor *web*, *kernel* mais leve, portabilidade, compatibilidade e custos reduzidos.

## 2.3. Python

Python é uma linguagem de programação de alto-nível e orientada a objetos. Desenvolvida por Guildo van Rossum em 1991, ela tem como objetivo priorizar a legibilidade do código sobre a velocidade ou expressividade, além de exigir poucas linhas de código em relação a outras linguagens. A linguagem Python foi utilizada em todos os códigos desenvolvidos neste projeto.

## 2.4. Descrição do Projeto

O projeto é um sistema de reconhecimento e identificação de voz. As falas pronunciadas pelo usuário são convertidas em comandos para que o sistema execute tarefas. Frases pré-programadas correspondem às atividades capazes de serem realizadas pelo sistema. A partir da fala do usuário, um mecanismo converte a fala em texto através do mecanismo Wit.ai, esse texto é então analisado e comparado com os comandos pré-definidos. Uma vez identificada qual a tarefa que deve ser realizada, o sistema a executa. Dentre os comandos planejados estão: gravar áudios e captura de imagens por meio de uma câmera; realizar a escrita de arquivos de texto; consultar os arquivos salvos no sistema; realizar buscas na internet e retornar o conteúdo encontrado na biblioteca livre Wikipedia; buscar informações sobre um filme no banco de dados do IMDb. O sistema interage com o usuário respondendo aos comandos a partir da conversão de textos para áudios, realizados por meio do Flite. Para a identificação dos usuários é utilizada a biblioteca Piwho.

Desta forma, caso usuário já esteja ocupado realizando uma tarefa, ou suas mãos estejam ocupadas, isso não o impede de controlar os dispositivos. Em um laboratório, os dados e informações obtidas podem ser salvas e armazenadas, sem a necessidade do contato direto com o sistema.

A figura 2 apresenta o diagrama de blocos do sistema.

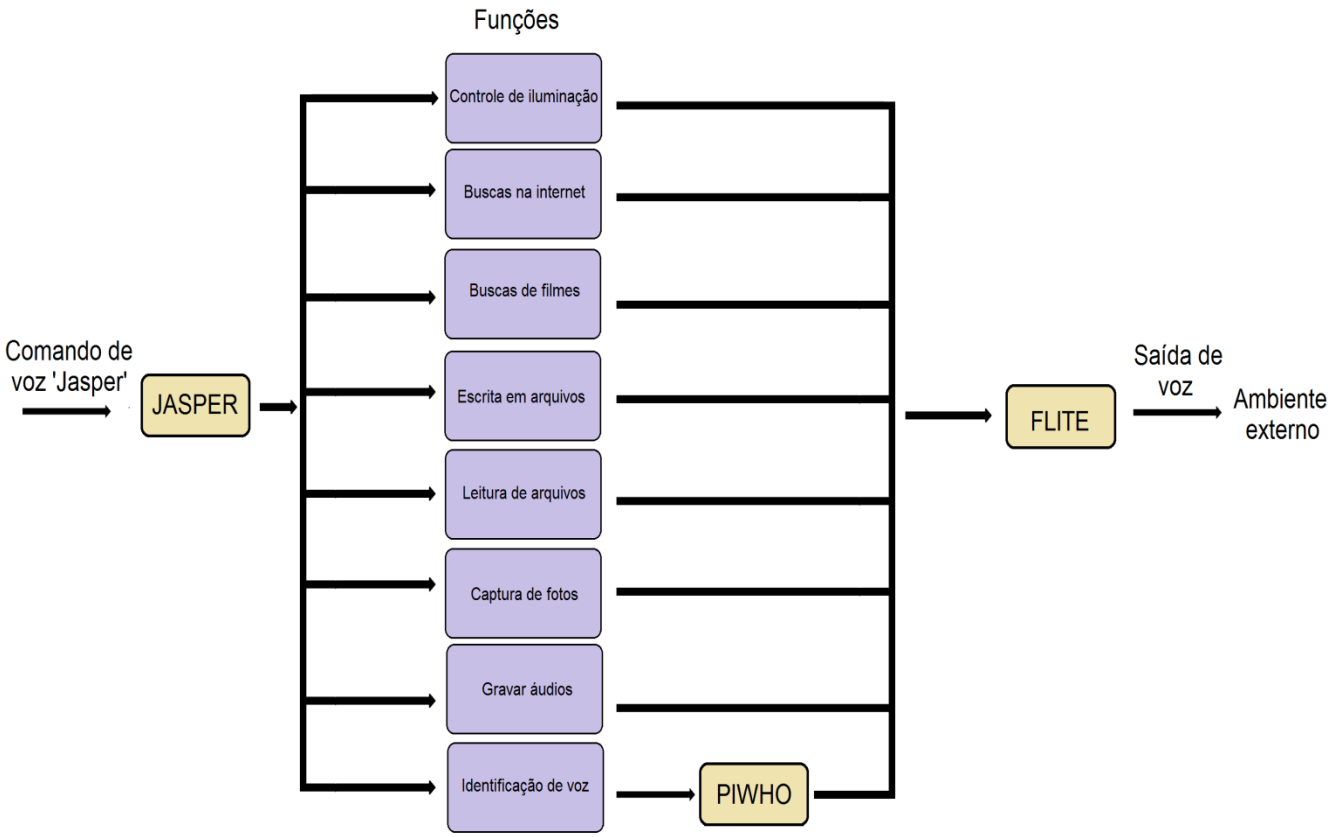


**Figura 2: Diagrama do sistema**

A figura 3 apresenta o diagrama de blocos de um processo do sistema.

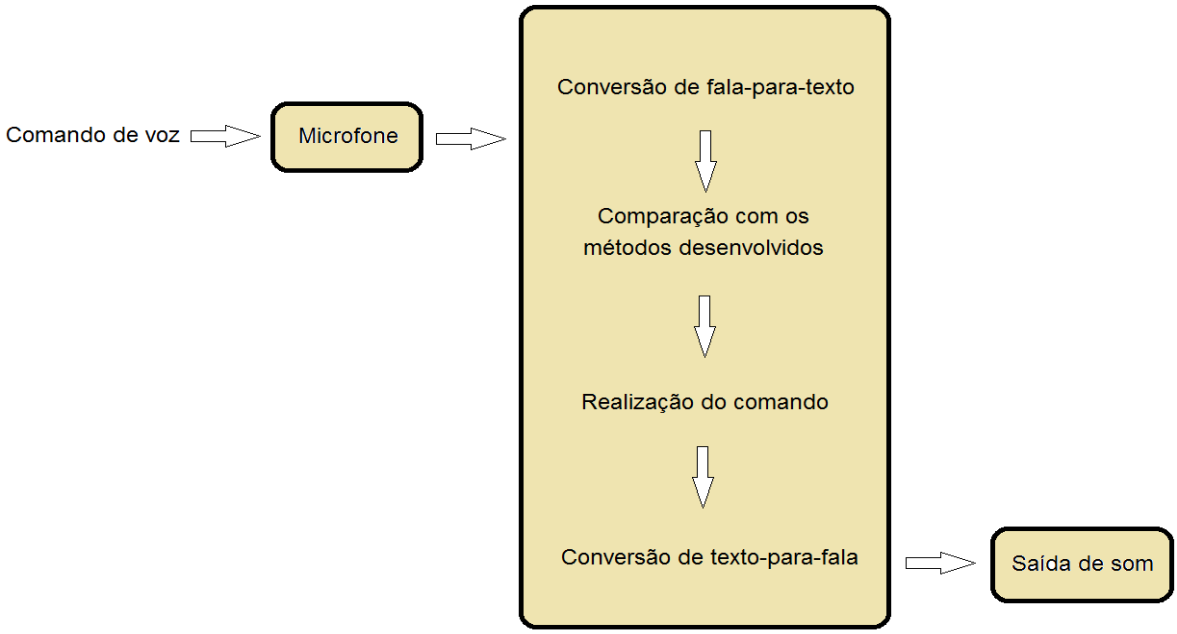


**Figura 3: Esquemático do sistema**

Por fim, todo o conteúdo armazenado no sistema, como resultado dos comandos, pode ser visualizado numa página na web. São informações contidas no *website*: dados acerca do projeto, fotos, áudios gravados, e a lista de comandos a serem realizados. Também pode ser visualizado na página as tentativas de acesso sem permissão, quando uma voz não registrada tenta ter acesso a alguns métodos, a imagem uma imagem é capturada e o horário é salvo.

## 2.5. Reconhecimento de Voz

Fala é uma forma de comunicação baseada em uma sintaxe de léxicos e nomes de um vocabulário. Léxico é definido como um acervo de palavras para um determinado idioma, o conjunto de palavras disponíveis para as pessoas dialogarem em uma mesma língua. Cada palavra é formada por um conjunto de vogais e consoantes que geram uma combinação fonética.

Um fonema é a menor unidade sonora que constitui uma palavra, sendo responsável por distinguir as palavras de um mesmo idioma. Correspondendo à acústica que o ouvido registra. Por exemplo, em português a diferença de significado entre as palavras “este” e “esta” é resultante apenas da troca do fonema /e/ pelo fonema /a/. Outro exemplo, em inglês, as palavras “kill” e “kiss” terminam com dois fonemas distintos (/l/ e /s/, respectivamente) e, alterar um pelo outro altera os significado da palavra. Fonemas são representados por barras(/ /).

Já fones são sons pronunciados pelos falantes de uma língua, apresentando variações regionais. Sua transcrição fonética é dada por colchetes([ ]). A combinação dos movimentos para se gerar a fala correspondem aos fones. Por exemplo, o fone [t] corresponde à movimentação dos lábios, língua e dentes para gerar este som. Os fones são produzidos por vogais e consoantes.

Levando em conta esses fatos, a dificuldade em se trabalhar com voz se deve a alguns aspectos:

- Dificuldade em remover barulhos e ruídos.
- Definir com precisão o início e fim de uma palavra.
- Palavras que possuem o mesmo som porém com significados diferentes (homônimos). Este fenômeno está presente em todos os idiomas, em português alguns exemplos são:

noz / nós, acento / assento, manga (fruta / parte da camisa que cobre os braços), e em inglês: red / read, them all / the mall, dentre outros.

- A estrutura gramatical e a semântica das palavras.

O processo de reconhecimento de voz é dividido em algumas etapas. Inicialmente um pedaço contínuo de fala (com início e fim determinados com uma pausa clara) é digitalizado a partir de um conversor analógico digital, as vibrações produzidas são desta forma convertidas em dados digitais. A forma de onda captada é filtrada para eliminar ruídos e interferências. Em seguida é efetuado o processo de obtenção do espectrograma aplicando a Transformada Rápida de Fourier (em inglês *Fast Fourier Transform*, FFT), o espectrograma mostra como a frequência muda de intensidade com o tempo. Divide-se em frações menores da fala, obtendo sequências de frames acústicos que não sejam maiores do que uma sílaba. Estes sons são comparados com os fonemas mapeados em um dicionário fonético correspondente ao idioma do interlocutor, todas as combinações de possíveis são testadas para ver se conferem com o áudio obtido. Além do dicionário fonético também podem ser utilizados modelos que descrevem um objeto matemático que reúne atributos comuns de uma palavra e funções com algoritmos baseados em aprendizado de máquina.

Modelos de linguagem tem uso restrito a busca de palavras, definindo qual palavra acompanha uma palavra anteriormente reconhecida, removendo palavras improváveis. Restringir este vocabulário permite uma identificação mais eficaz da palavra seguinte, resultanto em alta precisão.

Os principais modelos de linguagem utilizam modelos n-gram, que determinam a sequência de palavras a serem utilizadas por meio de análise estatística.

O modelo de voz de decodificação de voz muito utilizado é chamado de Modelo Oculto de Markov (em inglês *Hidden Markov Model*, HMM). Sendo descrito como uma sequência de estados que mudam cada um de acordo com uma probabilidade.

A figura 4 apresenta o procedimento descrito.

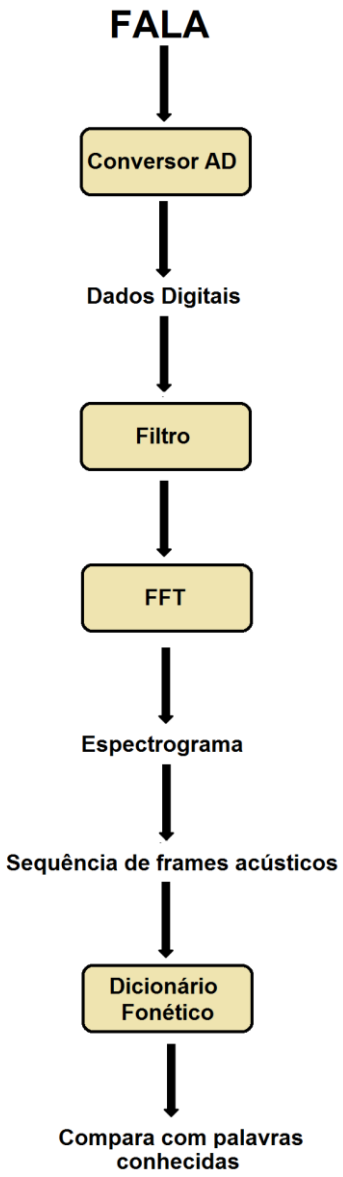


**Figura 4: Conversão de fala para texto**

Os mecanismos de reconhecimento de voz que transformam falas gravadas em texto escrito utilizando a metodologia descrita nesta seção. Dentre os mecanismos existentes, três em particular merecem destaque: Google Speech API, PocketSphinx e Wit.ai.

O Google possui uma API que realiza essa conversão de áudio para texto ao aplicar modelos de redes neurais, podendo reconhecer mais de 80 idiomas em diversas aplicações. O processamento não é realizado na própria Raspberry Pi, o código requer acesso à internet para acessar o servidor online e então realizar a conversão.

## 2.6. Conversão de texto para fala

Os sistemas conversores de texto para fala consistem de um mecanismo de saída capaz de gerar som de forma análoga ao trato vocal e de uma entrada de texto.

Os módulos de alta performace para pronunciação de palavras devem conter um dicionário com regras de pronúncia sofisticadas. O dicionário trata das palavras com sintaxe ou contexto dependente, e palavras irregulares que não possuem regras específicas. As regras de pronúncia são utilizadas para o restante das palavras, incluindo novas palavras e nomes.

O processo de conversão inicia-se com a normalização do texto, transformando em letras outros formatos de textos. Por exemplo: “10” torna-se “ten” e “Mr.” torna-se “mister”.

Na próxima etapa é utilizada o dicionário de exceções para as palavras que não são pronunciadas de acordo com as regras do idioma em que está inserido. A transcrição fonética da pronúncia dessas palavras estão contidas neste dicionário, desta forma que a pronuncia seja feita corretamente. Também estão armazenadas informações sobre a gramática de palavras particulares.

Algumas regras convertem as palavras em transcrições de fonemas, estes são representações mais exatas da pronúncia. Isso significa que, a palavra “cat” (gato, em inglês) é representada como “k z t”. Também são identificadas as terminações gramaticais como “-ing”, “-ly” e “-ness”. O próximo passo é a criação dos padrões de entonação contidos nas regras prosódicas.

As regras fonéticas são responsáveis pela afinação das pronúncias, por exemplo, o fonema “t” é pronunciado de forma diferente em “tank”, “notebook’’ e “cat”.

O módulo de geração de voz transforma os fonemas em pequenas unidades de fala que são convertidos em parâmetros de voz. Esse método permite a produção de um vozes variadas, uma diferente da outra.

## 2.7. MARF

O *Modular Audio Recognition Framework* (MARF) é um *framework* responsável pelas etapas de Treinamento e Reconhecimento, desenvolvido pelo *MARF Research and Development Group*.

O reconhecimento realiza a identificação da amostra baseado nos padrões armazenados pelo treinamento. É necessário que o áudio contendo a frase esteja no formato .WAV e seja de 16 bits. A frequência de 8000 Hz é o necessário para conter todas as

frequências do espectro de voz, obedecendo a lei de Nyquist. Outros formatos de áudio como .MP3 não estavam implementados na versão do MARF utilizado neste projeto.

O treinamento possui uma primeira etapa de pré-processamento. Essa etapa do MARF normaliza as amplitudes dos arquivos de áudio a serem treinados, elas devem permanecer no intervalo [-1.0, 1.0]. A normalização é necessária para garantir que todas as vozes permaneçam no mesmo nível.

A próxima etapa é a extração das características, realizada através de pequenas porções da amostra da voz. A partir dessas pequenas porções é que são extraídas as características da voz. Essa divisão é feita garantindo que a amplitude diminua gradativamente conforme se aproxima dos extremos. Essa técnica de cortar a amostra em pedaços menores para ser analisada é chamada de janela. Uma técnica de extração das características é o *Linear Predictive Coding* (LPC). Essa técnica avalia as seções das formas de onda da fala determinando um conjunto de coeficiente. Esse método possui um armazenamento limitado de dados.

Os desenvolvedores do *framework* informam que a configuração que apresenta a melhor taxa de treinamento e reconhecimento corresponde à utilização do pré processamento com o algoritmo *Endpoint*, e a extração das características com a técnica LPC.

Essas duas etapas são realizadas tanto no processo de treinamento como de identificação. No treinamento as características extraídas das amostras de áudio são armazenadas para posterior comparação.

No reconhecimento a amostra a ser identificada tem suas características extraídas, em seguida utiliza-se um método de classificação para determinar a proximidade entre as características da amostra analisada e as do modelo treinado a ser comparado. Este valor é chamado de distância e está contido no intervalo [0,1]. Quanto mais próximo de 0, mais próximas são as características umas das outras e, quanto mais próximo de 1 menor a chance de pertencerem ao mesmo indiívuo.

A Distância de Chebyshev é o método mais indicado para classificação. Sua representação matemática é dada por:

$$d(x, y) = \sum_{k=1}^{n} (|x_k - y_k|)$$

Onde *x* e *y* são os vetores de características, ambos de tamanho *n*.

# 3. Materiais e Métodos

Nesta seção é descrito as bibliotecas utilizadas no projeto.

## 3.1. Instalação Raspbian e configurações iniciais

Inicialmente foi *instalado o sistema operacional na plataforma Raspberry Pi utilizando o NOOBS (New Out Of Box*), armazenado em um cartão SD classe 8, contendo o Raspbian Jessie. Raspbian é uma versão GNU/Linux projetado especificamente para Raspberry Pi, e Jessie é sua versão disponível mais popular e atualizada. Os passos para a instalação do sistema operacional utilizando o NOOBS estão descritos no tutorial The Raspberry Pi Foundation, **Getting Started With you Raspberry Pi**.

Para configurar a conexão a internet e definir IP estático, foi alterado o arquivo *etc/network.interfaces* para a seguinte configuração:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
address 10.235.0.135
netmask 255.255.252.0
network 10.235.3.0
gateway 10.235.0.1
pre-up ifconfig eth0 hw ether 00:01:02:03:05:11
dns-nameservers  143.107.225.6 143.107.182.2 8.8.8.8
```

Em seguida para a habilitação do acesso SSH foi necessário modificar o arquivo *etc/ssh/sshd_config* para acessar a porta 22000.

Aa plataforma de código aberto Jasper foi utilizada como base para o projeto. Contendo a configuração inicial de integração dos mecanismo de conversão de fala para texto com o Flite para conversão de texto para áudio e, a busca dos códigos em Python a serem desenvolvidos.

## 3.2. Wit.ai

O Wit.ai é uma API para reconhecer fala e transformá-la em texto através da incorporação de mecanismos de processamento de linguagens. Por basear-se no uso de serviços em nuvem, a API requer conexão com a internet, sendo todo o processamento desenvolvido fora da Raspberry Pi. O Wit utiliza um elemento de *crowdsourcing* para treinar os algoritmos de reconhecimento, agregando contribuições dos desenvolvedores que o utilizam em suas aplicações.

## 3.3. Flite

Flite é um sistema sintetizador de texto para voz de código aberto desenvolvido na Carnegie Mellon University (BLACK, A. W.; LENZO, K. A, 2014). Projetado inicialmente para pequenos sistemas embarcados como um sistema de conversão de texto para voz alternativa ao sistema Festival, devido este ser pesado e devagar. Todo o processamento ocorre na Raspberry Pi.

O Flite consiste de: uma biblioteca que contém o código de síntese e os modelos de processamento linguísticos, os léxicos, as regras de som e um banco de dados.

## 3.4. Jasper

Jasper é uma plataforma *open source* para o desenvolvimento de aplicações de controle por voz. Desenvolvida por Charles Marsh e Shubhro Saha, a plataforma foi lançada sob a licensa do MIT (MARSH, C.; SHUBHRO, S, 2014). Jasper foi escolhido por ter sido desenvolvido especificamente para a Raspberry Pi Modelo B. A sua instalação e configuração é explicada na seção 3.5.

## 3.5. Configuração das preferências do usuário

A configuração do usuário para definir os mecanismos de reconhecimento e sintetização de voz estão armazenados no arquivo *~/.jasper/profile.yml*, este arquivo organiza texto em um formato que é de fácil compreensão humana, sendo gerado a partir *populate.py*.

Para a utilização do mecanismo de conversão de fala para texto escolhido, Wit.ai, foi necessário obter um token de acesso diretamente do website do grupo. Para converter texto em fala foi utilizado o mecanismo Flite.

O arquivo contendo as definições do usuário ficou no seguinte formato:

```
stt_engine: witai
witai-stt:
  access_token:    XXXXXXXXXXXXXXXXXXXX

tts_engine: flite-tts
flite-tts:
  voice: 'slt'
```

## 3.6. Módulos

Os módulos se encontram dentro da pasta *jasper/client/module*s, lá são adicionados todos os módulos desenvolvidos. Eles seguem a estrutura definida pelo Jasper.

As palavras-chave são as palavras que o sistema precisa identificar quando o usuário fala no microfone. O dicionário dessas palavras deve ser pequeno para aprimorar a precisão.

É requisitado uma lista de *strings* chamada WORDS, dentro desta lista que estão contidas as palavras-chave.

O arquivo Sentence.py será utilizado para compreender o funcionamento e a codificação dos módulos.

```
WORDS = [“RANDOM”]
```

No método isValid(input) são definidos quais inputs são válidos, isso significa que retorna True caso o input esteja relacionado com something e False caso contrário.

```
def isValid(text):
    return bool(re.search(r'random', text, re.IGNORECASE))
```

Qualquer palavra utilizada no isValid deve também ser adicionada na lista WORDS.

A seguinte seção de código requisita o método handle(input, mic, profile), sendo input relacionado com a fala, mic é o objeto do microfone utilizado para capturar a voz e, profile refere-se às informações do usuário contidas em *profile.yml*.

```
def handle(text, mic, profile):

    messages = ["This is an example.",
                "I will be back to help you.",
                "Your wish is an order",
                "What do you command?",
                "This is a speech recognition system."]

    message = random.choice(messages)

    mic.say(message)
```

Cinco frases foram definidas para serem aleatoriamente ditas quando requisitada a ação. Elas são armazenadas em *messages* para posteriormente serem escolhidas aleatoriamente através de *random.choice(messages*). Em seguida o programa envia a frase escolhida para ser transformada de texto para fala em *mic.say*.

### 3.7. Arquivos de sistema

Para iniciar o sistema o arquivo *jasper.py* deve ser executado, nele as informações contidas no profile do usuário são carregadas para checar as configurações. O arquivo de execução determina quais mecanismos de conversão de fala para texto e texto para fala se deve utilizar baseado no que está definido no profile.

O arquivo *conversation.py* fica em execução infinitamente até que o Jasper seja encerrado. Sua função é identificar as palavras-chave que o usuário disse ou se simplesmente não houve nenhum comando.

O arquivo brain.py é responsável por cruzar as referências do input do usuário com a lista de módulos. A ordem dos módulos importa, a execução é finalizada assim que o primeiro módulo que contém o entrada informado é encontrado. Por isso os módulos são organizados de acordo com sua prioridade, e esta deve ser definida dentro do código de cada módulo. A

entrada obtida pelo microfone é comparada com as possíveis palavras-chave (WORDS) de cada módulo.

### 3.8. Piwho

Piwho é uma biblioteca gratuita e *open source* desenvolvida por Aditya Khandkar. A biblioteca é baseada no framework MARF para uso em Raspberry Pi. Foi utilizada para o treinamento dos usuários e o reconhecimento de voz.

### 3.9. Treinamento do de modelos

O treinamento do modelo é realizado em Python, como a biblioteca Piwho é utilizada para essa função, ela deve ser importada. Seu código *recognition.py* contém a classe *SpeakerRecognizer*, nela contém os métodos de treinamento dos dados e identificação de características. Primeiramente, o caminho até o arquivo de áudio do indivíduo a ser treinado deve ser passado como parâmetro para a classe. Em seguida,o nome correspondente ao treinamento do modelo é informado para o método *speaker_name*. Por fim, para realizar o treinamento basta utilizar o método *train_new_data()*.

Os arquivos de áudio utilizados para o treinamento da voz devem estar no formato .wav. O método de treinamento do modelo confere se o arquivo de áudio possui as demais características obrigatórias (8000 Hz, 16 bits, 1 canal) e se necessário converte o arquivo de modo a atendê-las.

```
from piwho import recognition
recog =
recognition.SpeakerRecognizer("/home/recordings/speaker_TiagoLandim/speaker_TiagoL5.wav")
recog.speaker_name = 'TIAGO'
recog.train_new_data()
```

O treinamento gera dois tipos de arquivos: speakers.txt e outro no formato .gzbin. O arquivo .gzbin contém as características da voz extraídas do arquivo de áudio. Já o arquivo speakers.txt contém cada usuário treinado, indicando o ID, o nome atribuído, e os arquivos de áudio utilizados para o treinamento.

0,NOME_DO_USUÁRIO_1,ARQUIVO_DE_TREINAMENTO_1.wav|ARQUIVO_DE_TREINAMENTO_2.wav|ARQUIVO_DE_TREINAMENTO_3.wav|

1,NOME_DO_USUÁRIO_2,ARQUIVO_DE_TREINAMENTO_4.wav|

2,NOME_DO_USUÁRIO_3,ARQUIVO_DE_TREINAMENTO_5.wav|

O reconhecimento indica o quão próximo o áudio está para cada modelo treinado. É atribuído ao áudio analisado o modelo que mais aproxima suas características de voz.

# 4. Discussões

Esta seção trata dos módulos desenvolvidos em Python para este projeto.

## 4.1. get_time()

O primeiro módulo trata-se da obtenção da data e hora atual. Essa função não implementa o reconhecimento de fala, porém foi é utilizada nas aplicações CaptureImage, RecordAudio e CreateSpeaker.

Para este modulo utiliza-se o a biblioteca *datetime*. A classe *datetime.datetime* corresponde a combinação de data e hora, com os atributos: *year, month, day, hour, minute, second, microsecond*. No inicio do *script* a biblioteca deve ser importada.

```
import datetime
```

É utilizado o método *datetime.datetime.now()*, para retorna a data e hora atual. Na variável *now* é atribuída o retorno do método. Essa variável agora é do tipo *datetime.datetime.*

```
now = datetime.datetime.now()
```

Cada um dos atributos é obtido à partir da variável *now*. Em todas as aplicações, data e hora atual são utilizados no nome do arquivo a ser armazenado, desta forma os atributos, que são do tipo *inteiro*, foram convertidos para o tipo *string*. Uma manipulação de adicionar o algarismo ‘0’ na frente de números menores do que 10 foi necessária para que a exibição dos arquivos ocorresse de acordo com a data e hora em que foram salvos.

```
year = str(now.year)
if(now.month<10):
    month = '0' + str(now.month)
else:
    month = str(now.month)

if(now.day<10):
    day = '0' + str(now.day)
else:
    day = str(now.day)

if(now.hour<10):
    hour = '0' + str(now.hour)
else:
```

```
        hour = str(now.hour)

    if(now.minute<10):
        minute = '0' + str(now.minute)
    else:
        minute = str(now.minute)

    if(now.second<10):
        second = '0' + str(now.second)
    else:
        second = str(now.second)
```

Não foi necessário utilizar os milisegundos pois as aplicações demoram mais de um segundo para serem executadas.

## 4.2. Search.py

Uma biblioteca em Python para realizar buscas no site do wikipedia foi utilizado. Sua utilização é relativamente simples. Através da entrada por voz, a aplicação procura na plataforma do Wikipedia o assunto determinado, retornando o sumário da pesquisa. A função wikipedia.summary(name, sentences=2) realiza a pesquisa, onde, *name* corresponde ao assunto a ser buscado e sentences à quantidade de parágrafos iniciais do sumário. O resultado da busca é por fim convertido em áudio.

## 4.3. CaptureImage.py

Pygame é uma biblioteca gratuita e *open source* em Python para desenvolvimento de aplicações multimídia. A biblioteca possui suporte para interface com câmeras, permitindo captura de imagens. E manipulações básicas de visão computacional. O Raspian já contém o Pygame instalado.
O módulo da câmera deve ser importado no inicio do programa.

```
import pygame
import pygame.camera
from pygame.locals import *
```

Como primeiro passo o módulo deve ser inicializado manualmente:

```
pygame.init()
pygame.camera.init()
```

A câmera deve ser aberta e inicializada para que em seguida um frame seja capturado. A câmera é encontrada em /dev/video0, e sua resolução máxima é de 352x288 pixel.

```
cam = pygame.camera.Camera("/dev/video0",(352,288))
cam.start()
image = cam.get_image()
```

A imagem é salva e, ao final, a câmera deve ser liberada. A variável “name” indica com qual nome que a imagem será armazenada.

```
pygame.image.save(image, name)
cam.stop()
```

Nesta aplicação as imagens são salvas com a data e hora da captura do frame, sendo name o retorno da função get_time().

## 4.4. WriteDown.py

Para se iniciar o processo de tomar notas, a palavra-chave “WRITE” deve ser identificada. Inicialmente o sistema pergunta qual será o nome do arquivo, a resposta é escutada por mic.activeListen(), seu conteúdo é convertido para letras minúsculas por padrão, para evitar futuros erros. O arquivo é adicionado na pasta ‘NoteFiles’, contida em ‘/home/pi’. Como o arquivo é aberto no formato ‘write’ (‘w’), caso outro arquivo com o mesmo nome já exista, este será substituído e, toda a informação anteriormente contida será perdida.

```
mic.say('What will be the files name?')
fileName = mic.activeListen()
fileName = fileName.lower()
name = os.path.join('/home/pi/NoteFiles', fileName + ".txt")
f = open(name, "w")
```

O sistema faz uma pergunta sobre o que deve ser escrito e escuta a resposta. Essa resposta, atribuída à *response* é do tipo *unicode* caso algo seja pronunciado. É realizada uma checagem sobre o tipo de *response*. Caso seja unicode, a frase dita pelo usuário será escrita em uma linha do arquivo de texto, caso contrário nada será escrito.

```
mic.say('What do you want me to write down?')
response = mic.activeListen()
```

```
  bool = type(response) is unicode
  if(bool):
     f.write(response + '\n')
  else:
        mic.say('You said nothing')
```

O sistema pergunta se há mais informações a serem escritas no arquivo, se a resposta for unicode o script continua, caso contrário o arquivo é fechado finalizando a escrita.

```
mic.say('Anything else?')
  response = mic.activeListen()
  bool = type(response) is unicode
  if(bool==True):
        .
        .
        .
  else:
     mic.say('Nothing else to write down')
```

Uma resposta afirmativa permite que mais frases sejam adicionadas ao arquivo de texto. A resposta é afirmativa se corresponder a palavra “YES”. Essa identificação é realizada pela função *def yes(text)*, que funciona da mesma forma que a identificação das palavras-chave. Ao identificar que contém a afirmação na frase, a função retorna o valor verdadeiro.

```
def yes(text):
  return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))
```

Com uma resposta afirmativa o script continua, é perguntando o que deve ser adicionado no arquivo, a resposta é escutada e, novamente, se for do tipo *unicode* (significando que foi identificado uma fala) é escrita, caso contrário o sistema informa que nada foi dito. Logo em seguida mais uma vez é perguntado ao usuário se deseja continuar adicionando no arquivo de texto. Para uma resposta afirmativa, o sistema continua o procedimento de escrita. Enquanto a resposta for afirmativa, contendo a palavra “YES”, sistema permanece em *loop* escrevendo no arquivo.

```
     while yes(response):
           mic.say('What else would you like me to write down?')
           response = mic.activeListen()
           bool = type(response) is unicode
           if(bool):
```

```
        f.write(response + '\n')
    else:
        mic.say('You said nothing')
    mic.say('Anything else?')
    response = mic.activeListen()
```

No momento em que o usuário não desejar continuar escrevendo, a resposta não deverá conter a palavra “YES”, assim yes(response) retorna um valor falso. Ao final do *loop* o arquivo é fechado e pode ser lido.

### 4.5. ReadFile.py

Esta função realiza a leitura de um arquivo de texto previamente armazenado. Uma vez identificada a palavra-chave “READ” na fala do usuário, o sistema pergunta o nome do arquivo a ser lido. A fala do usuário deve conter apenas o nome do arquivo, caso a fala pronunciada não corresponda a nenhum arquivo armazenado, o sistema informa que ocorreu um erro em sua operação. A resposta é ouvida através do comando fileName = mic.activeListen(). A variável fileName contém o nome do arquivo buscado, seu conteúdo é convertido para letras minúsculas e então o arquivo é aberto e lido pelo sistema.

```
fileName = fileName.lower()
name = os.path.join('/home/pi/NoteFiles', fileName + ".txt")
f = open(name, "r")
content = f.read()
mic.say(content)
f.close()
mic.say('This is everything written in this file')
```

Ao fim do programa o arquivo é fechado e é informado que todo seu conteúdo foi lido.

### 4.6. Movies.py

A biblioteca IMDbPY é uma biblioteca open source encontrada em imdbpy.sourceforge.net. Nesta aplicação foi utilizada a versão 5.0 dessa biblioteca, porém atualmente já está disponível a versão 5.1. A biblioteca foi implementada para pesquisar um título de filme no banco de dados do IMDB e retornar suas informações.

O módulo deve ser importado no início do script para ser utilizado na aplicação.

```
from imdb import IMDb
```

Para esta pesquisa, a frase dita pelo usuário deve conter uma das palavras-chave: “Movie” ou “Movies”. Após o sistema identificar uma das duas palavras-chave, é requisitado que o usuário informe o filme desejado. Apenas o nome do filme deve ser recebido pela função mic.activeListen(), caso contrário tudo o que for dito será admitido como o título de um filme. A consulta é realizada pela função IMDb().search_movie(name), onde *name* é a variável que armazenou o conteúdo de mic.activeListen(). Essa primeira consulta retorna vetor de cinco filmes cujo título contenham  o que estava armazenado em *movie_name*. O tamanho deste vetor foi definida arbitrariamente.

O sistema retorna por voz um título que contenha o que estava armazenado em *movie_name* e pergunta se este era o filme desejado. Assim como foi feito no script WriteDown.py, a função *def yes(text)* retorna verdadeiro para uma frase que contenha a palavra “YES”, senão retorna valor falso.

```
def yes(text):
  return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))
```

Uma vez confirmado o título, é realizada outra consulta no banco do IMDb para recuperar as informações do filme. Oito informações são recuperadas: título, nota da avaliação, duração em minutos, gênero, sinopse, diretores, produtores e elenco. Como os filmes contém um grande número de integrantes nas suas categorias (diretor, produtor e elenco), foi definido um limite de três pessoas por categoria. Ao final da consulta todas essas informações são convertidas em áudio.

Há a possibilidade de o vetor resultante da primeira consulta não retornar o título desejado, para este caso o usuário não deve confirmar nenhum dos títulos, desta forma o sistema informará que não foi possível encontrar o filme requisitado.

```
  mic.say('Impossible to find the requested movie')
```

### 4.7. RecordAudio.py

Para essa função ser ativada é necessário que duas palavras-chave sejam pronunciadas em seguida.São elas: “RECORD” e “AUDIO”. O sistema informa uma mensagem

de erro caso a fala contenha uma palavra ou mais entre record e audio. Uma vez reconhecida as palavras-chave, o sistema pergunta ao usuário o que deve ser gravado. A resposta é ouvido pelo métod recordListen().

```
record = mic.recordListen()
```

O método mic.recordListen() possui a mesma estrutura do método mic.activeListen(), porém com algumas diferenças em seu código. O tempo que o usuário pode falar é de 180 segundos em recordListen(), isso permite frases maiores e que um pensamento inteiro possa ser armazenado sem quebras. Como os arquivos de áudio devem ser armazenados para posterior consulta, o método get_time() é utilizado, desta forma não há conflito de mais de um áudio com o mesmo nome.

```
path = os.path.join('/home/pi/JasperAudio', (recordedaudio + '.wav'))
shutil.copyfile(f.name, path)
```

Por fim o sistema informa que o áudio foi gravado com sucesso.

```
mic.say('Audio recorded')
```

## 4.8. Recognition.py

Inicialmente deve-se importar o módulo Piwho para poder utilizar seus métodos.

```
from piwho import recognition
```

A palavra-chave que a fala deve conter para acessar essa função é “RECOGNIZE”. Inicialmente o sistema solicita ao usuário pronunciar qualquer frase, esta frase é escutada pelo método *mic.activeListen()*. A frase é salva em um arquivo de áudio no diretório ‘/home/pi/recordings/Reconhecer’ no formato .WAV com o seguinte nome: ‘recordingToRecognize.wav’. A classe *SpeakerRecognizer()* do módulo *recognition* é atribuído à variável *recog*.

```
mic.say('Say something, I will guess who you are')
sentence = mic.activeListen2()
recog = recognition.SpeakerRecognizer()
```

Para determinar a identidade do locutor, é passado como parâmetro o caminho para o arquivo de áudio em que foi salvo a fala do indivíduo. O método *identify_speaker* identifica a pessoa do arquivo de áudio passado como parâmetro de acordo com as identificações contidas em speaker.txt e modelo treinado. O método retorna uma lista com as pessoas cujas características de voz mais se assemelham às características analisadas no arquivo de áudio informado. O usuário com as características mais próximas está na posição zero da lista. À variável *name* é, primeiramente, atribuída uma lista vazia, em seguida *name* recebe a lista de usuários identificados. Caso o indivíduo que não esteja adicionado ao banco de dados em speaker.txt, então o primeiro nome contido na lista será o do usuário do banco cujas características são as mais próximas às do áudio. O nome contido em *name[0]* é então pronunciado pelo sistema, finalizando o script.

```
name = []
name = recog.identify_speaker('/home/pi/recordings/Reconhecer/recordingToRecognize.wav')
person = name[0]
mic.say('You are ' + person)
```

## 4.9. Turn_Light_ON.py Turn_Light_OFF.py

RPi.GPIO é uma biblioteca em Python que permite o controle das porta GPIO da Raspberry Pi. Esta biblioteca já vem inclusa na versão do Raspian.

A porta GPIO (General Purpose Input/Output) são basicamente pinos que permitem a interface entre a Raspberry e o exterior, fazendo a comunicação de entrada e saída de sinais digitais. Estão localizados numa das extremidades da placa, próximos da saída de vídeo e, há um total de 26 pinos. A figura 4 exibe o posicionamento de cada pino.

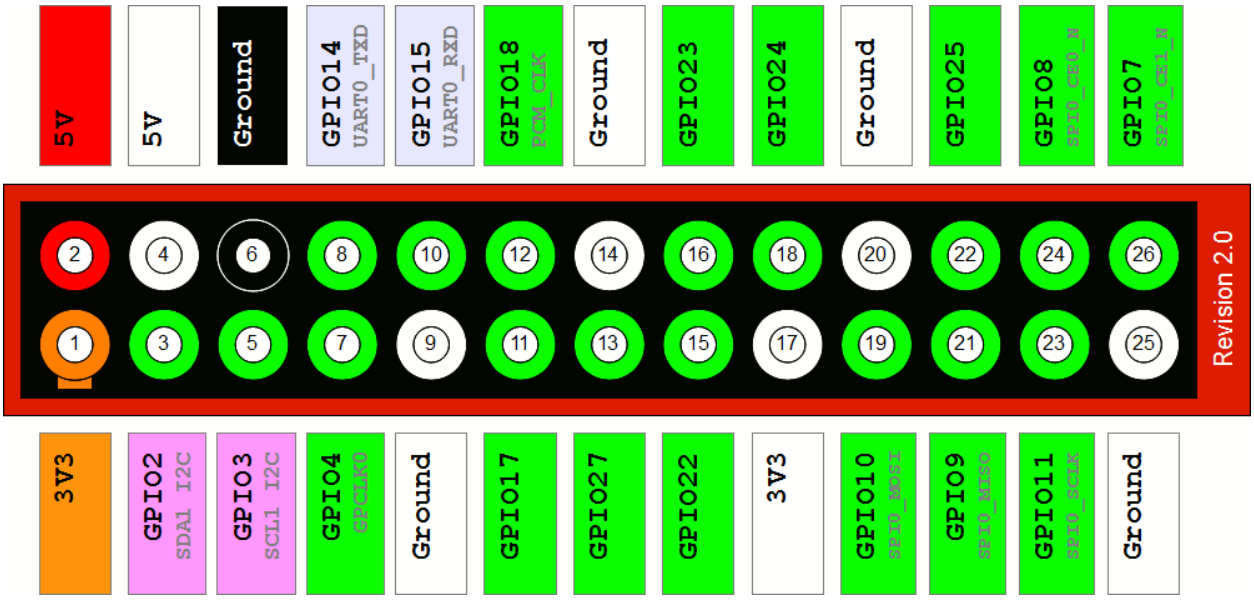


**Figura 5: Pinagem da porta GPIO**

**Fonte: Simple Guide to the RPi GPIO Header and Pins. Disponível em: < http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/> Acesso em: 22 mar. 2017**

Quando definidos como Input, os pinos podem ser ligados ou desligados externamente, quando definidos como Output, os programas da placa que definem quando os pinos estão ligados ou não.

Três funções foram desenvolvidas para interfacear o sistema com um elemento externo, no caso o elemento consiste de uma lâmpada. Um dos programas é utilizado para o acionamento da iluminação, outro para o desligamento da mesma e, o último possibilita tanto acender quanto desligar a lâmpada. O interfaceamento entre a Raspberry e a iluminação ocorre através da porta GPIO conectada a um módulo de relé.O relé suporta cargas de até 10A em 125VAC ou 250VAC.

O módulo de relé possui três pinos de entradas para a conexão com a Raspberry Pi (VCC, GRD, IN), sendo o sinal de acionamento do relé recebido pela entrada IN. Outras três entradas para a conexão com os equipamentos elétricos, que são: NA (Normalmente Aberto), C (Comum), NF (Normalmente Fechado). Um dos terminais do soquete é conectado ao Comum, e pela entrada NA é feita a ligação com a tomada. O outro terminal do soquete também é conectado à tomada. O pino 7 da porta GPIO quando configurado como saída fecha o circuito entre NA e C, acionando a lâmpada. Para a porta configurada como entrada, o circuito abre fazendo a lâmpada se apagar.

Deve-se importar a biblioteca para utilizar RPi.GPIO no script em Python:

```
Import RPi.GPIO as GPIO
```

Inicialmente é definido esquema de numeração da pinagem, são duas possibilidades:

- BOARD: especifica que será utilizada a numeração dos pinos na placa;
- BCM: especifica que será utilizada o sistema de numeração definido pela Broadcom;

A numeração neste aplicação será BOARD, através do comando GPIO.setmode(GPIO.BOARD).

Após esta configuração inicial, foi necessário determinar o modo do pino (entrada ou saída). Foi definido o pino 7 como saída pelo comando: GPIO.set(7, GPIO.OUT). Como último passo dessa função, deve-se escrever o nível lógico no pino (alto ou baixo). Ao setar o pino com GPIO.output(7, HIGH) fornece uma tensão de 3.3V ligando o LED, já o comando GPIO.output(7, LOW) desliga o LED ao enviar 0V.

## 4.10. CreateSpeaker.py

Este módulo trata da criação de um novo usuário, este usuário passará a ter permissão para a leitura dos arquivos de texto salvos. Apenas um usuário principal, pré determinado, possui permissão para registrar novos indivíduos. O sistema inicia o processo perguntando se há permissão para a criação de novos usuários. As características da voz de resposta são comparada com as características salvas de ‘TIAGO’ na função *permissionGranted().* Se o valor da distância entre essas características for menor do que 0,30, a função retorna verdadeiro, se for maior que 0,30, retorna falso.

```
def permissionGranted():
        #Recognize the speaker, if the speaker is TIAGO, then return TRUE otherwise return
FALSE
        recog = recognition.SpeakerRecognizer()
        name = []
        name =
recog.identify_speaker('/home/pi/recordings/Reconhecer/recordingToRecognize.wav')
        while x == False:
        if dist.keys()[i] == 'TIAGO':
                value = dist.values()[i]
                x = True
        else:
```

```
        i = i + 1
if float(value) < 0.30:
        return True
else:
        return False
```

Uma vez obtida a permissão é perguntado o nome do novo usuário e é requisitado que o novo usuário pronuncie duas frases. Essas duas frases correspondem às amostras que a serem utilizadas para o treinamento de suas características. Ao fim deste processo o novo usuário é adicionado no arquivo *speaker.txt* e seu modelo de treinamento ao arquivo .gzbin.

Caso a permissão não tenha sido garantida, o horário da tentative de acesso é adicionada ao arquivo de texto *permissionIntruder* e, uma imagem do indivíduo que requisitou o acesso é capturada. Essas informações podem ser posteriormente conferidas na página Intrusos do website.

## 4.11. Webpage

Neste item sera explicado a interação entre a página web e a Raspberry Pi. Para a programação web foram utilizadas as linguagens PHP e HTML 5. Um *template* foi utilizado como base do código. O servidor utilizado para a webpage foi o Apache, um servidor web livre. Todos os arquivos utilizados na página foram transferidos para o diretório '*/var/www/html*'. Na figura 6 tem-se uma imagem da página principal do site, nela há algumas informações sobre o projeto. Também contém 3 links de redirecionamento para outros endereços, estes links correspondem à: Imagens, Arquivos e Intrusos. Além destes 3 ainda há o link Home que redireciona o usuário para a página principal do site.


**Figura 6: Página inicial do website**

O código em PHP da página Imagens busca todas as imagens contidas no diretório *'/var/www/html/images/JasperImages'* e as exibe em 4 colunas, gerando uma galeria de fotos. É nesse diretório que foram salvas todas os frames capturados pelo script CaptureImage.py. A figura 7 exibe a tela deste link.


**Figura 7: Tela da página Imagens**

A página Arquivos  (ver Figura 8) realiza o mesmo procedimento de Imagens. Os conteúdo dos arquivos de texto armazenados no diretório ‘home/pi/NoteFiles’ são exibidos em sequência, começando do arquivo mais recente.


**Figura 8: Tela da página Arquivos**

Por fim, o último link corresponde à página Intrusos (ver figura 9). Essa página é responsável por exibir todos os indivíduos sem permissão que tentaram acessar algum módulo restrito, como é o caso de ReadFile.py e CreateSpeaker.py.

Essa página se divide em dois segmentos. No primeiro são exibidos os módulos que algum indivíduo sem autorização tentou realizar o acesso e, o horário em que a tentativa ocorreu. Essas informações estavam salvas no arquivo de texto permissionIntruder.txt, adicionadas pelos módulos citados acima. A segunda seção desta página contém as imagens capturadas no momento da tentativa de acesso. O objetivo das informações contidas nessa página o controle do acesso ao sistema. Nada será exibido na tela caso não tenha ocorrido nenhuma tentativa de acesso aos módulos restritos.


**Figura 9: Tela da página Intrusos**

# 5. Resultados

Neste momento cada etapa do projeto é avaliada, desde o reconhecimento de voz com o modulo Jasper, passando pelos scripts desenvolvidos e até a identificação de voz com o Piwho.

## 5.1. Módulos de fala-para-texto e texto-para-fala

Inicialmente o mecanismo de conversão de fala para texto utilizado foi o Google Speech API. Foram realizados testes online no sistema do Google para analisar a resposta e a eficácia do mecanismo. A primeira situação envolvia falas no idioma inglês ao mesmo tempo em que ruídos ocorriam no ambiente. Em nenhum momento eles foram identificados como palavras ou causaram falhas na identificação das palavras. Em uma segunda situação, músicas eram tocadas ao fundo e, novamente, apenas as palavras ditas pela pessoa foram convertidas em texto, a música não causou nenhuma tipo de interferência.

Apesar de apresentar o melhor desempenho em relação aos outros mecanismos, o uso da API foi descontinuado no projeto, isso ocorreu devido à sua limitação. Para se utilizar gratuitamente a API existe um limite de 60 minutos de áudio a ser convertido por mês. Como o projeto requer o uso contínuo desta função, mecanismo Wit.ai. teve que ser implementado.

Percebeu-se que em alguns momentos a palavra “JASPER” não é corretamente reconhecida. Essa falha de reconhecimento ocorre para indivíduos cujo idioma nativo não é o inglês. O sistema do Google foi utilizado para estes testes. Como a voz produzida corresponde à um nativo, não ocorrem falhas no reconhecimento.

O sistema de conversão de texto Flite cumpriu sua função com sucesso em todas as situações em que foi necessitado. Durante todo o desenvolvimento do projeto, o Flite não gerou respostas de voz erradas para nenhum dos módulos desenvolvidos. Em todas as suas chamadas, os textos foram convertidos em áudio corretamente e sem pausas, erros ou alterações na voz definida.

## 5.2. Módulos implementados

Para o método de busca no Wikipedia foram realizados testes para conferir o tempo de resposta da função. É de interesse para o usuário do sistema que o tempo de espera seja o menor possível.

Para uma internet com velocidade de 14Mb/s, os resultados aproximados foram de 18 segundos, 13 segundos.

A demora no tempo de resposta pode ser atribuída à limitações tanto de hardware como da velocidade da internet.

Para o método WriteDown.py, de escrita em arquivos de texto, o sistema reconhece erroneamente a palavra “RIGHT” ao invés de “WRITE”. Como essa falha ocorre regularmente o script WriteDown.py foi alterado para ser admitida qualquer uma das palavras-chave: “WRITE” e “RIGHT”. Além da semelhança na pronúncia das duas palavras, esse erro ocorre também devido ao sotaque de um usuário cuja língua nativa não é inglês.

No método de obtenção de informações sobre um filme em específico, a busca é realizada pela internet, pois essa é a única forma possível de se ter acesso ao banco do IMDb. As estatísticas do próprio IMDb apontam que seu banco contém um total de 4.293.589 títulos, iniciando no ano de 1874. Ao observar este número e, considerando que são realizadas duas vezes a consulta ao banco, já era esperado que a busca e comparação dos títulos não retornasse as informações rapidamente.

A primeira consulta, pelos cinco filmes, demorou 8 minutos e 33 segundos. A consulta pelas informações do filme específico demorou um tempo de 10 minutos e 2 segundos.

A quantidade de informações requisitadas, como gênero, duração, produtores, elenco, etc, não afeta no tempo. O tempo depende da busca no banco e, como esperado, é necessário um tempo em minutos para percorrer um banco com milhões de elementos.

## 5.3. Identificação de usuário

Alguns testes foram realizados a fim de avaliar a eficiência do sistema. Para os testes, foram utilizados dois usuários: ‘TIAGO’ e ‘LUIS’.

Os testes para a identificação do usuário foram realizados utilizando um total de 22 amostras de áudio. Essas 22 amostras se dividem em 5 conjuntos:

- Conjunto 1. 7 arquivos correspondem à falas do usuário ‘TIAGO’;

- Conjunto 2. 3 arquivos correspondem à falas do usuário ‘LUIS’;
- Conjunto 3. 7 arquivos correspondem à falas de indivíduos não registrados;
- Conjunto 4. 3 arquivos correspondem à fragmentos de músicas;
- Conjunto 5. 2 arquivos correspondem à falas em português do usuário ‘TIAGO’

O teste teve como objetivo determinar como a quantidade de modelos treinados para um mesmo usuário afeta sua identificação. 7 situações foram testadas, sendo incremental o número de treinamentos para o usuário ‘TIAGO’ e limitado a 3 modelos treinados para o usuário ‘LUIS. Os testes são os seguintes:

- 1: 1 treinamento para ‘TIAGO’ e 1 treinamento para ‘LUIS’;
- 2: 2 treinamentos para ‘TIAGO’ e 2 treinamentos para ‘LUIS’;
- 3: 3 treinamentos para ‘TIAGO’ e 3 treinamentos para ‘LUIS’;
- 4: 4 treinamentos para ‘TIAGO’ e 3 treinamentos para ‘LUIS’;
- 5: 5 treinamentos para ‘TIAGO’ e 3 treinamentos para ‘LUIS’;
- 6: 6 treinamentos para ‘TIAGO’ e 3 treinamentos para ‘LUIS’;
- 7: 7 treinamentos para ‘TIAGO’ e 3 treinamentos para ‘LUIS’;

Inicialmente, o treinamento realizado utilizou apenas uma amostra para cada usuário. Para cada um dos 5 conjuntos de arquivos de áudio, foi calculada a distância entre as características da amostra treinada do usuário ‘TIAGO’ e as características cada amostras dos conjuntos.

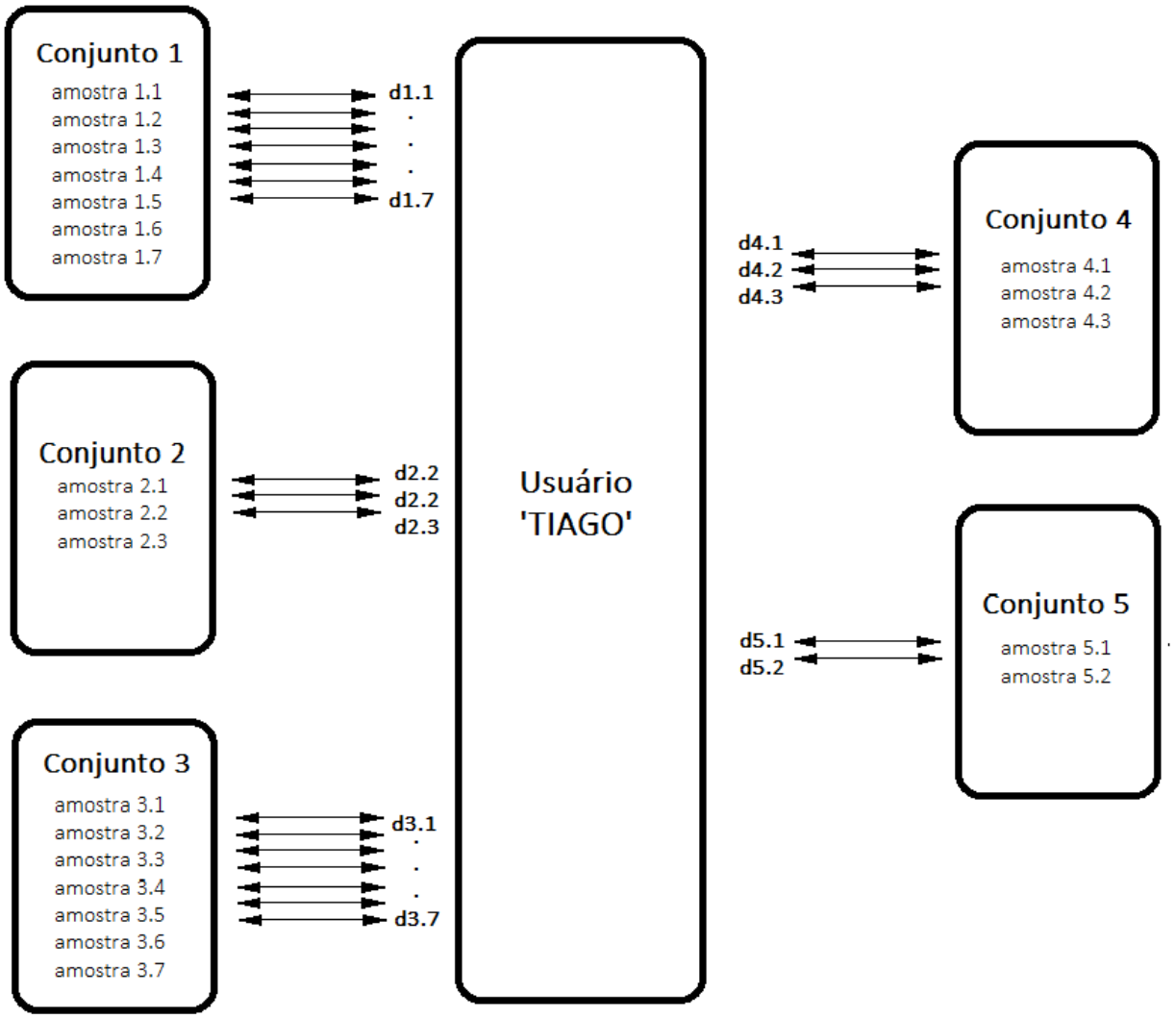


**Figura 10: Distâncias entre as características de cada amostra e as características do usuário**

Em seguida, foi realizada a média entre as distâncias calculadas para as amostras de um mesmo conjunto, ou seja, ao final deste processo obteve-se 5 distâncias entre as características de cada conjunto e as características do usuário 'TIAGO'.

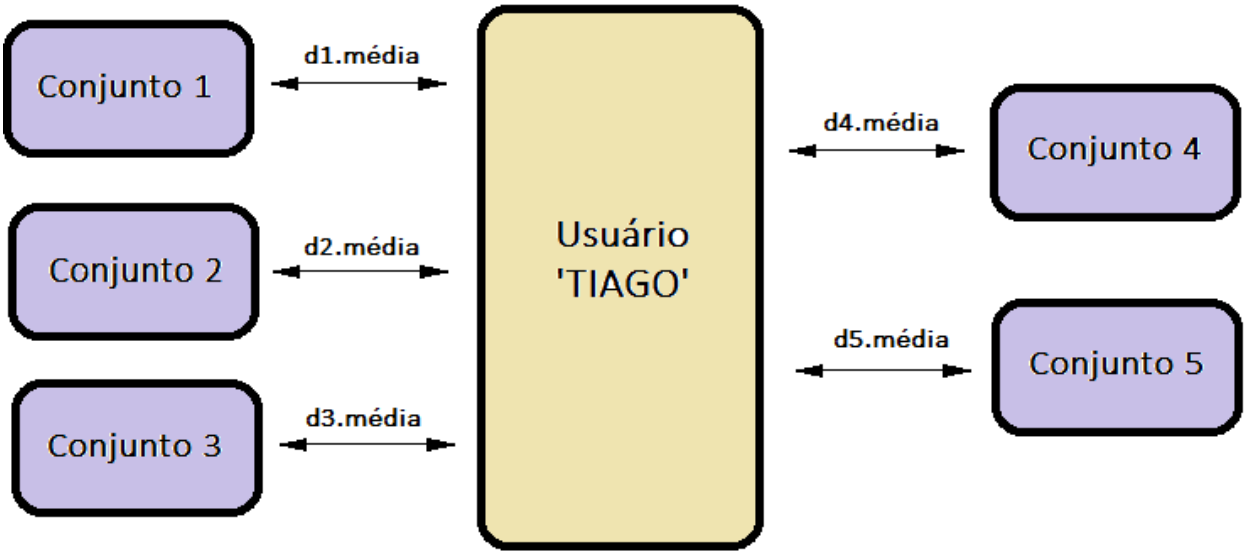


**Figura 11: Distâncias média entre as características de cada amostra e as características do usuário**

Para continuar a análise, foi realizado o segundo teste, em que foram treinados duas amostras para cada usuário. Os mesmo cinco conjuntos de amostras foram utilizados novamente, as distâncias entre as características cada uma de suas amostras de cada conjunto e as características do usuário ‘TIAGO’.

Mais uma vez foi realizada a média entre as distâncias das amostras para cada conjunto, obtendo-se novas 5 distâncias, correspondentes às distâncias entre as características de cada conjunto e as características do usuário.

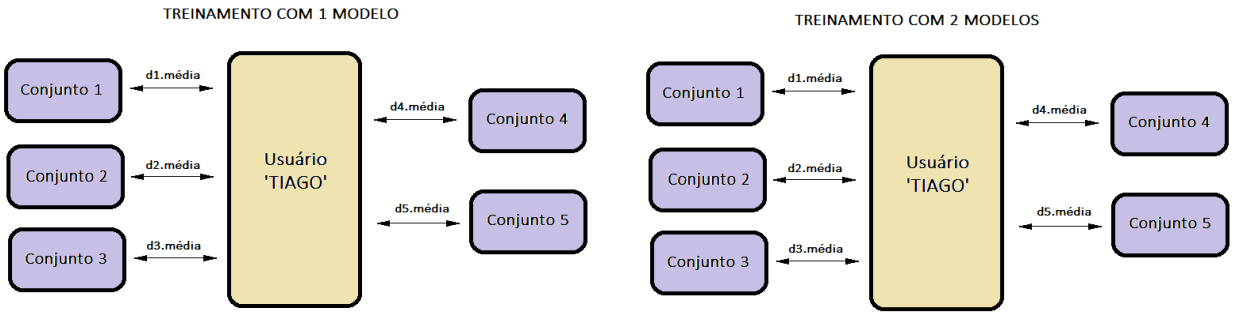


**Figura 12: Distâncias médias entre as características de cada amostra e as características do usuário para treinamento com 1 e com 2 modelos**

As etapas descritas anteriormente foram realizadas até para o restante das 7 situações, incrementando o número de modelos de treinamentos para o usuário ‘TIAGO’.

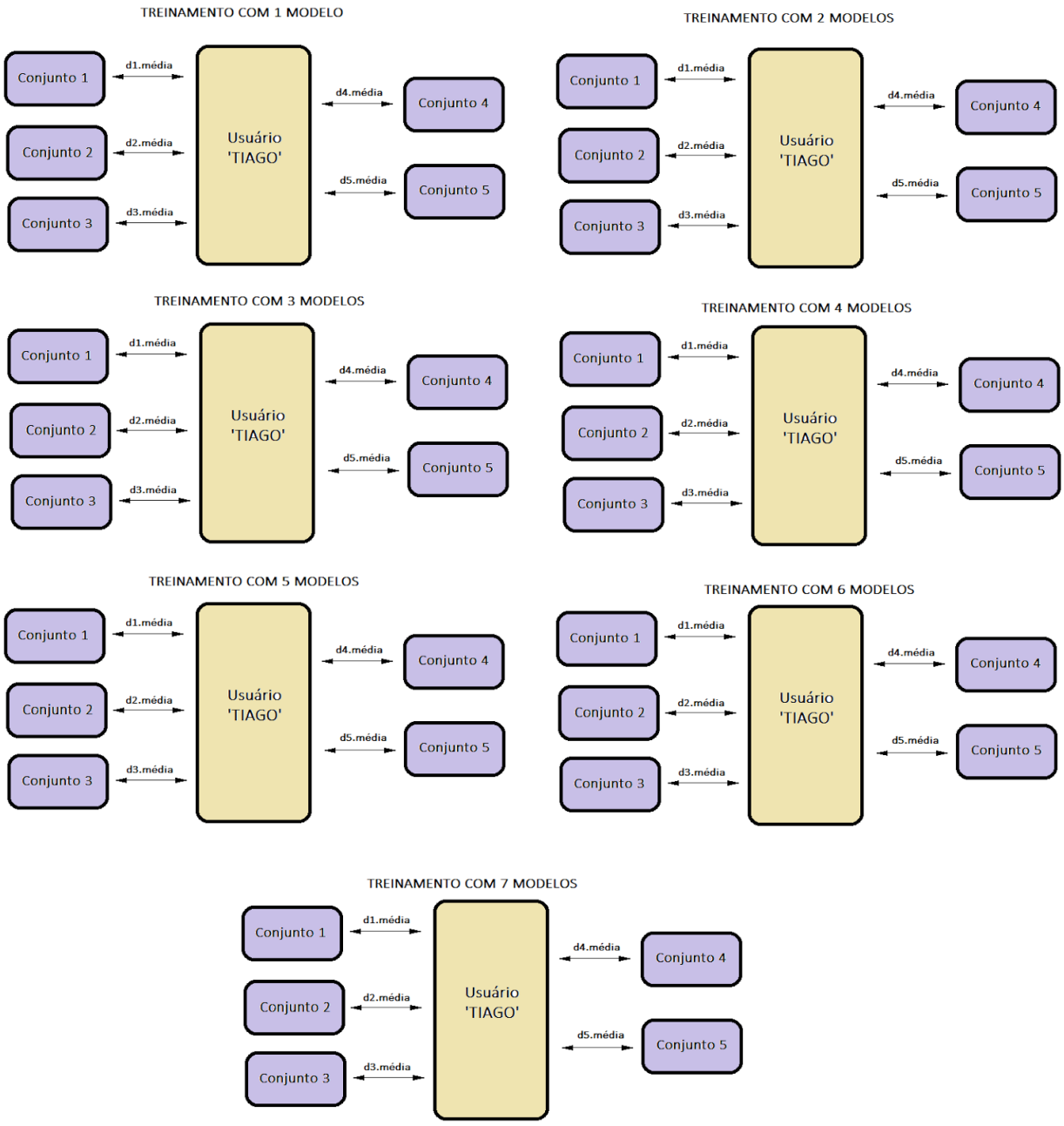


**Figura 13: Distâncias médias entre as características de cada amostra e as características do usuário para treinamentos com de 1 à 7 modelos**

As distâncias médias entre as características de cada amostra e as características do usuário para treinamentos com, de 1 à 7 modelos, estão indicadas na Tabela 1.

**Tabela 1: Distâncias médias para cada conjunto em relação à quantidade de modelos treinados - TIAGO**

| Quantidade de modelos Treinados | Distâncias médias entre as características de cada conjunto e as características do usuário 'TIAGO' | | | | |
|---|---|---|---|---|---|
| | Conjunto 1 | Conjunto 2 | Conjunto 3 | Conjunto 4 | Conjunto 5 |
| 1 | 0.327756 | 0.557485 | 0.606045 | 0.519317 | 0.715846 |
| 2 | 0.248799 | 0.615358 | 0.634606 | 0.51798 | 0.680893 |
| 3 | 0.235117 | 0.657771 | 0.659103 | 0.572082 | 0.705524 |
| 4 | 0.252253 | 0.649032 | 0.662196 | 0.541552 | 0.702064 |
| 5 | 0.250357 | 0.609121 | 0.634495 | 0.517786 | 0.69575 |
| 6 | 0.228216 | 0.616247 | 0.630827 | 0.522595 | 0.68087 |
| 7 | 0.217446 | 0.617566 | 0.628312 | 0.529302 | 0.666175 |

O resultado causado pelo aumento no número de modelos treinados fica mais evidente quando os dados da Tabela 1 são transportados para os gráficos de 1 à 5. Cada gráfico corresponde às distâncias médias de cada conjunto para o usuário 'TIAGO'. No eixo x encontra-se o número de treinamentos e no eixo y encontra-se a média das distâncias entre as características das amostras do conjunto em questão e as características treinadas para o usuário.

Os resultados variam dependendo do conjunto. Para o conjunto de amostras correspondentes à falas do usuário 'TIAGO' (conjunto 1), percebe-se que a distância entre as características do usuário e suas amostras de áudio diminuem conforme o número de modelos treinados aumenta (ver gráfico 1). Para apenas um modelo treinado o valor médio das distâncias foi de 0,3377, com um desvio padrão de 0,0343. Já para sete modelos treinados, esse número decaiu para 0,2174, com um desvio padrão de 0,0348.

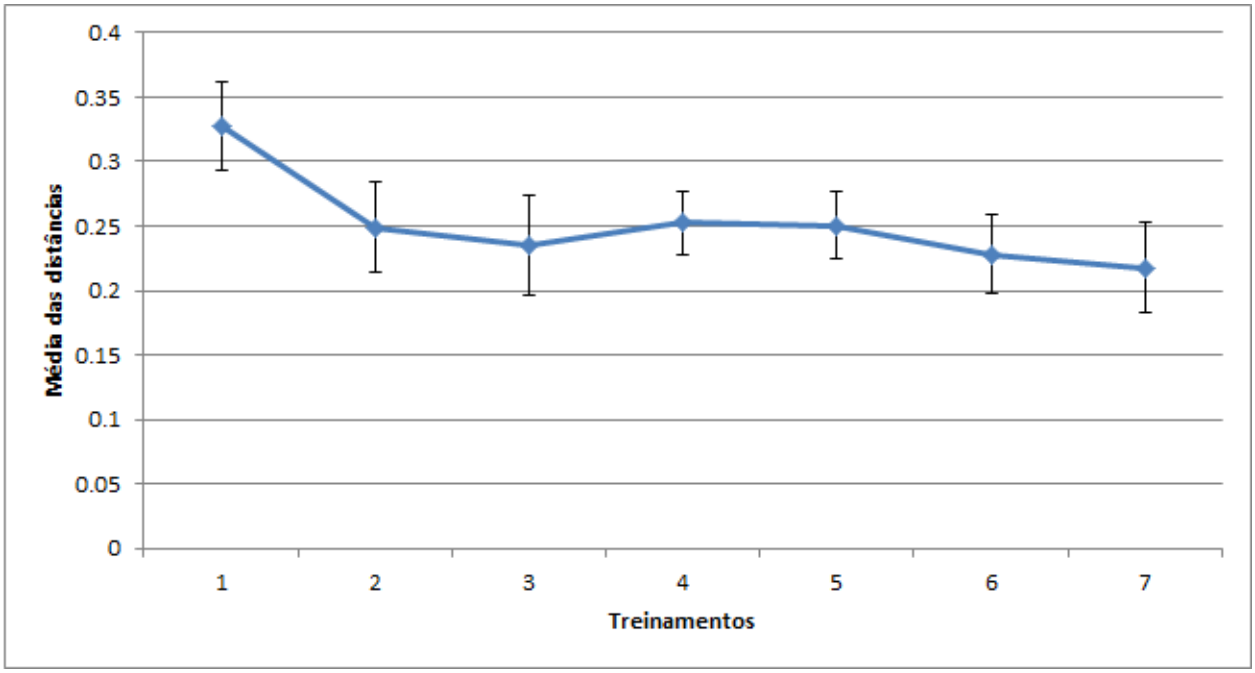


**Gráfico 1: Média das distâncias para conjunto 1 x treinamentos para usuário 'TIAGO'**

Baseado nos valores obtidos e apresentados na tabela 1 e gráfico 1, calculou-se a taxa que a distância média diminui em relação à distância média do primeiro treinamento, conforme aumenta do número de modelos treinados. A tabela 3 apresenta os resultados deste cálculo.

Observando os dados nota-se que já no segundo modelo treinado, a distância média diminuiu em 24,0902%. E, depois de 7 treinamentos essa distância média caiu em 33,6563% se aproximando mais ainda.

Apesar de a taxa ter aumentado novamente para 4 e 5 modelos treinados, ela volta a diminuir a partir do sexto treinamento.

**Tabela 2: Taxa eficiência do reconhecimento de acordo com o número de modelos treinados**

| Quantidade de modelos Treinados | Taxa de diminuição (%) |
|---|---|
| 2 | 24.0902 |
| 3 | 28.2645 |
| 4 | 23.0363 |
| 5 | 23.5691 |
| 6 | 30.3702 |
| 7 | 33.6263 |

Os gráficos 2 e 3 exibem os resultados para os conjuntos 2 (amostras do usuário ‘LUIS’) e 3 (amostras de indivíduos não registrados), respectivamente. Sabendo que a origem das amostras não corresponde ao usuário em análise (TIAGO), percebe-se que fica com valores elevados, acima de 0,5. O conjunto 3 apresentou valores variados, alguns muito afastados, como 0,908529 e, alguns valores pouco mais próximos como 0,412989. Uma vez que foram utilizadas amostras de diferentes indivíduos, algumas falas foram identificadas com características muito diferentes do usuário e outros um pouco mais próximas.

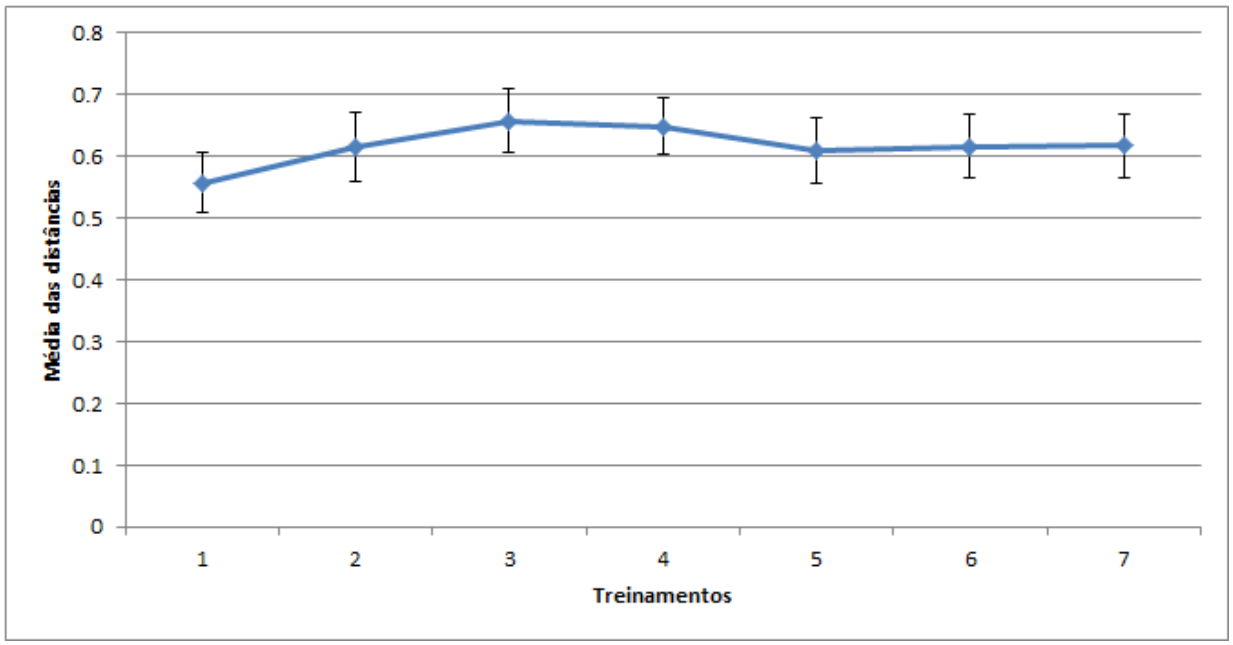


**Gráfico 2: Média das distâncias para conjunto 2 x treinamentos para usuário 'TIAGO'**

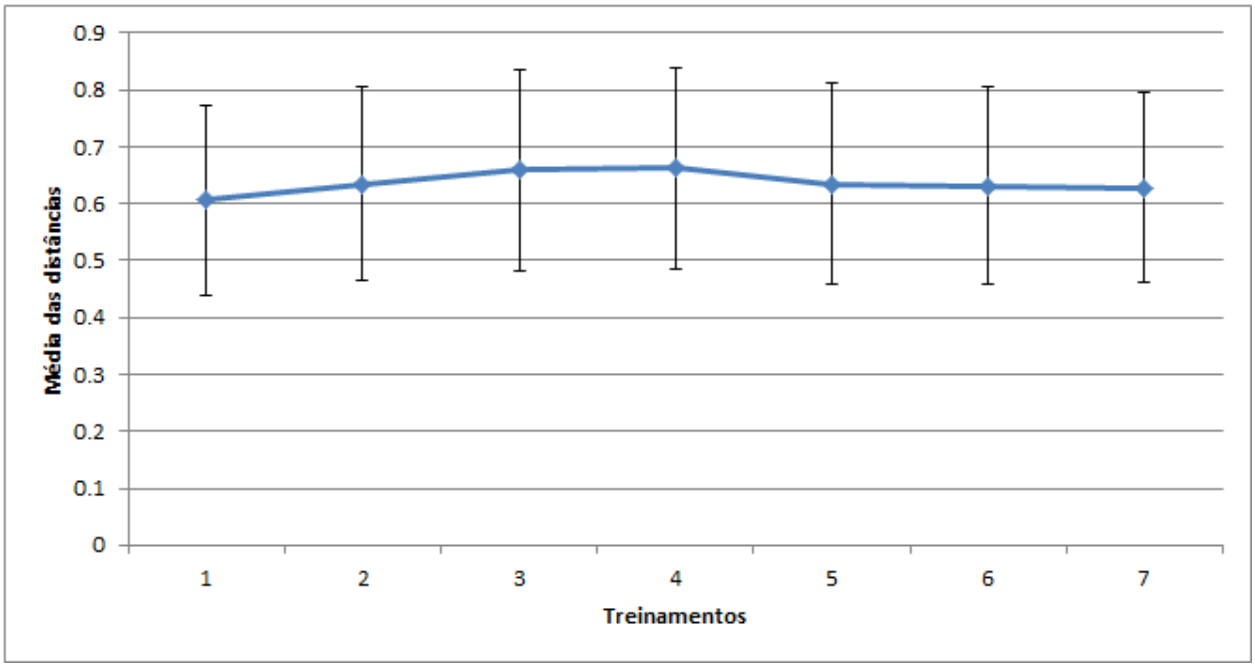


**Gráfico 3: Média das distâncias para conjunto 3 x treinamentos para usuário 'TIAGO'**

As amostras do conjunto 4 são aquelas que correspondem à fragmentos de músicas, especificamente de momentos em que o vocalista canta. Essas amostras também apresentaram resultados afastados do usuário em análise (ver gráfico 4), com o desvio padrão

as distâncias não foram menores que 0,384726. Mesmo estando distantes de zero, as amostras musicais apresentaram valores mais próximos do que falas dos conjuntos 2 e 3.

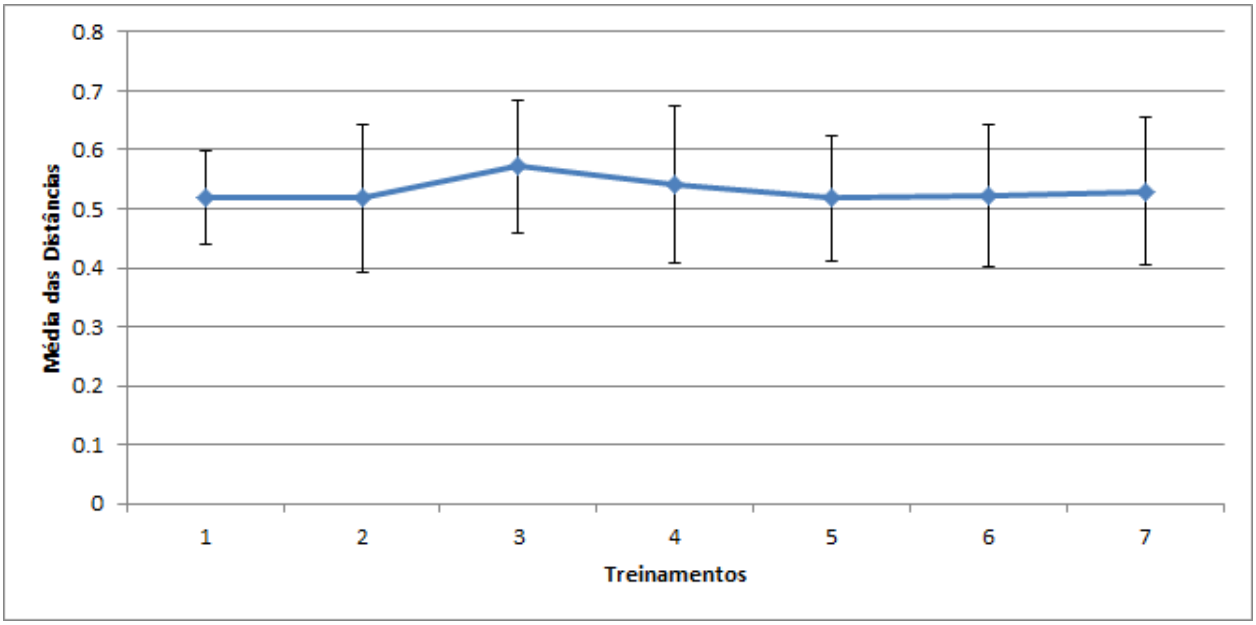


**Gráfico 4: Média das distâncias para conjunto 4 x treinamentos para usuário 'TIAGO'**

Por fim, os resultados das amostras do conjunto 5 mostraram-se em valores próximos de 0,7, com mínimo de 0,666175 e máximo de 0,702064. Deve-se reparar que este conjunto corresponde a amostras de fala no idioma português do mesmo usuário que está sendo comparado. Percebe-se que mesmo que as amostras venham da mesma origem, as características mudam drasticamente quando o idioma da fala é alterado.

Considerando todo o conjunto de distâncias obtidas, foram divididos em: 48 valores como verdadeiros positivos, 91 valores como verdadeiros negativos, 0 como falso positivos e 1 como falso negativo. Sensibilidade é definida como o total de verdadeiros positivos dividido pela soma do número de verdadeiros positivos com o número de falso negativo. Desta forma, a sensibilidade do sistema é de 97,959%.

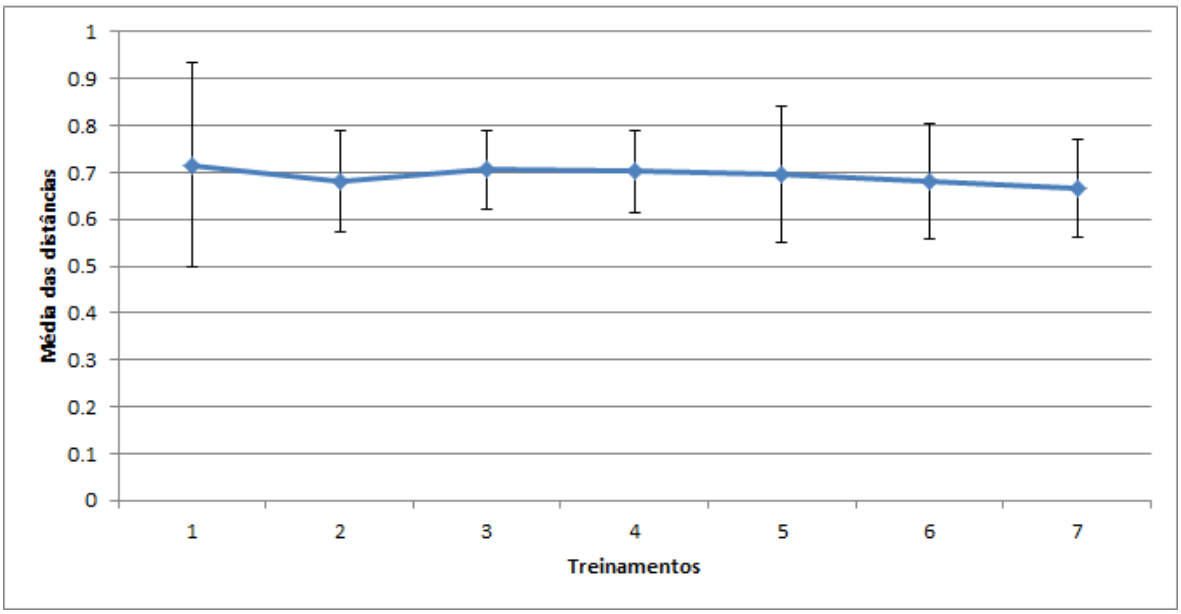


**Gráfico 5: Média das distâncias para conjunto 5 x treinamentos para usuário 'TIAGO'**

Ao se treinar um mesmo usuário com uma nova amostra, o modelo será atualizado, melhorando sua capacidade de reconhecimento. Porém não altera a distância das características quando um indivíduo não registrado tenta ser reconhecido.

O mesmo procedimento foi realizado para o usuário 'LUIS', porém para estes testes não se utilizou as amostras do conjunto 5. A tabela 3 apresenta os resultados obtidos.

**Tabela 3: Distâncias médias para cada conjunto em relação à quantidade de modelos treinados - LUIS**

| Quantidade de modelos Treinados | Distâncias médias entre as características de cada conjunto e as características do usuário 'LUIS' | | | |
|---|---|---|---|---|
| | Conjunto 1 | Conjunto 2 | Conjunto 3 | Conjunto 4 |
| 1 | 0.710351 | 0.201491 | 0.599247 | 0.627361 |
| 2 | 0.691238 | 0.176917 | 0.572085 | 0.571949 |
| 3 | 0.659828 | 0.163991 | 0.566504 | 0.561764 |

Para melhor visualização e comparação dos resultados, os dados foram transportados para representação gráfica. Os gráficos de 6 a 9 exibem os resultados para os conjuntos de 1 a 4, respectivamente.

Enquanto o gráfico 3 indicava os resultados das características de amostras do usuário 'LUIS' em relação as treinadas do usuário 'TIAGO', o gráfico 6 apresenta a situação invertida, as características de amostras do 'TIAGO' são comparadas com as treinadas de 'LUIS'. Os valores encontrados para ambas situações são semelhantes, nos dois casos as distâncias ficam no intervalo de 0,5 à 0,75.

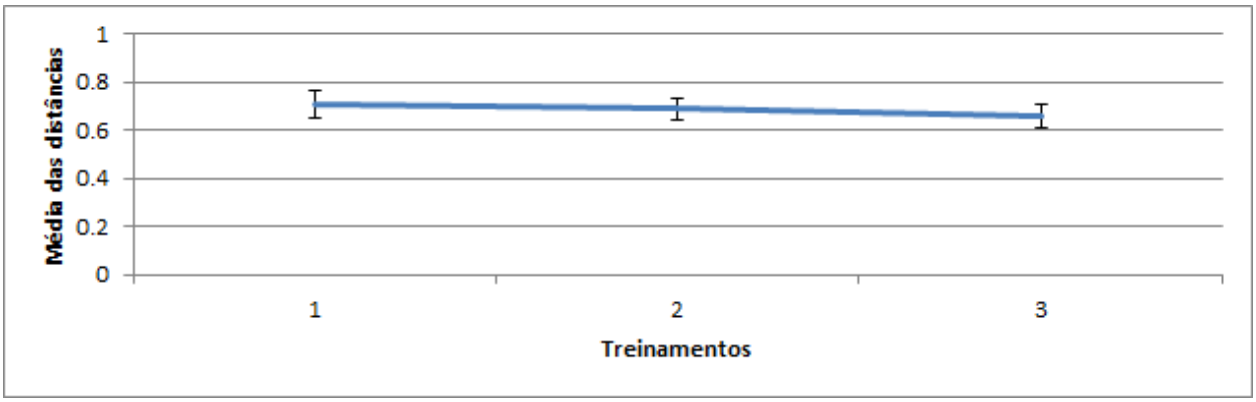


**Gráfico 6: Média das distâncias para conjunto 1 x treinamentos para usuário 'LUIS'**

O conjunto 2 corresponde à amostras do próprio usuário em análise. Como esperado as distâncias entre as características extraídas dos arquivos de áudio e as características do usuário são menores do que para os demais conjuntos. As médias correspondem a um intervalo de 0,172917 e 0,201491. Os desvios também apresentam valores muito pequenos, de 0,174505, 0,043506 e 0,0227587, para treinamentos de 1 a 3, respectivamente.

Conforme se aumentou o número de modelos treinados, as distâncias diminuíram bem como os desvios, garantindo que após o terceiro treinamento a maior distância não passou de 0,20.

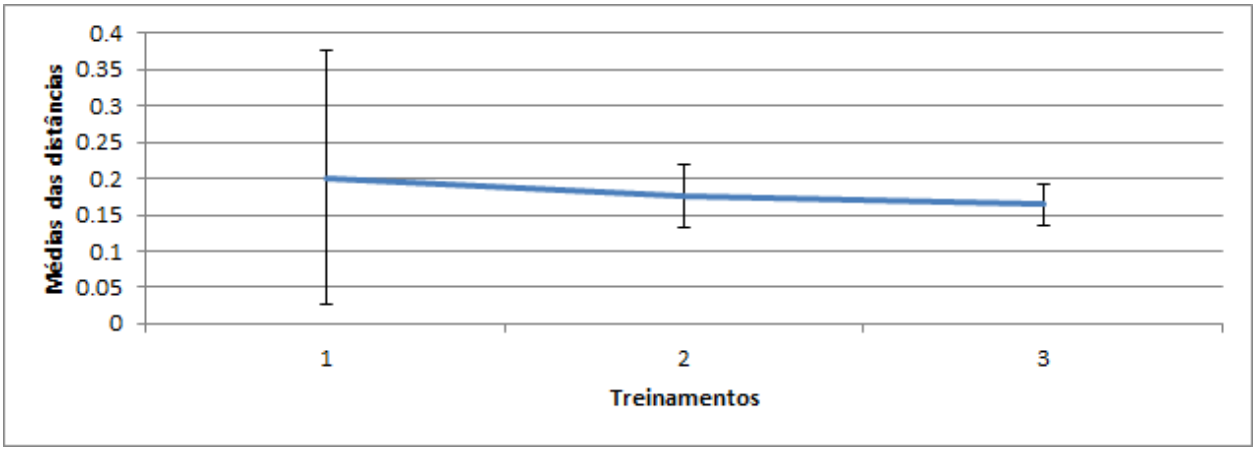


**Gráfico 7: Média das distâncias para conjunto 2 x treinamentos para usuário 'LUIS'**

Da mesma forma que ocorreu no gráfico 3, indivíduos não registrados apresentaram características mais próximas do usuário em análise do que as características do outro usuário cadastrado. As médias aparecem no intervalo entre 0,566504 e 0,599247, porém o desvio padrão resulta em um valor mínimo de 0,312423 e, valor máximo de 0,941806. Sugere-se que alguns indivíduos comparados possuem características mais próximas do usuário em análise e, outros mais afastadas.

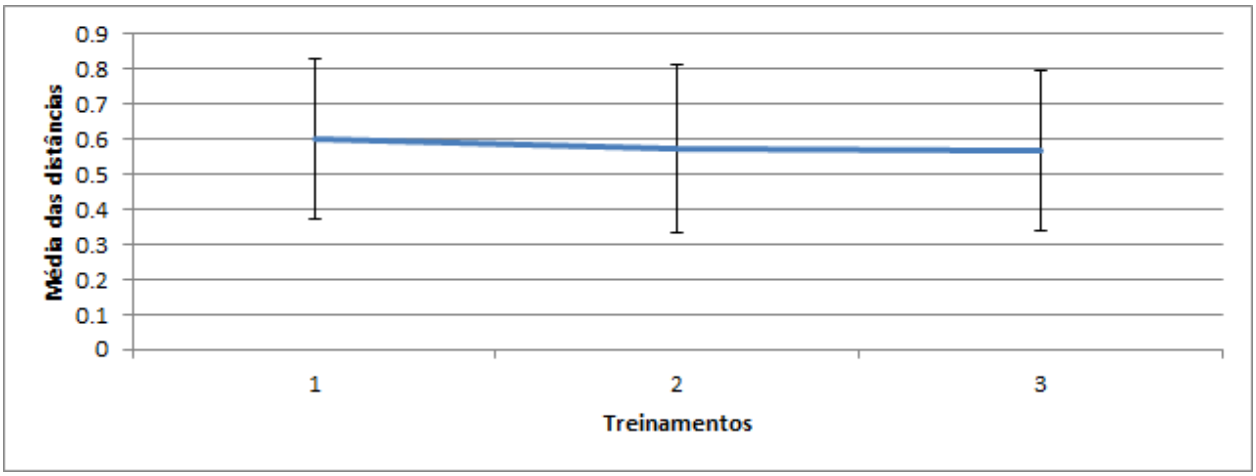


**Gráfico 8: Média das distâncias para conjunto 3 x treinamentos para usuário 'LUIS'**

As amostras de fragmentos musicais apresentam médias semelhantes à amostras de áudio de falas dos conjuntos 1 e 3.

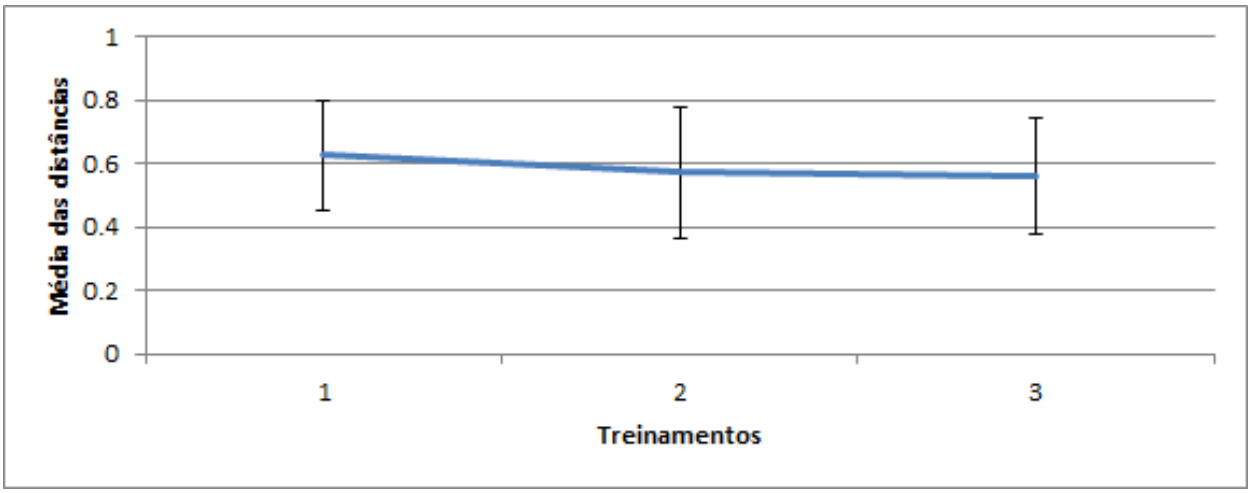


**Gráfico 9: Média das distâncias para conjunto 4 x treinamentos para usuário 'LUIS'**

A partir de observações de todos os resultados obtidos dos dois usuários, pode-se garantir que, de fato, a quantidade de modelos treinados para um usuário influencia no seu reconhecimento. Conforme mais amostras são treinadas, a distância entre as características extraídas da fala da pessoa cadastrada e as suas características armazenadas no treinamento ficam cada vez menores.

Dois números podem ser retirados dessa análise:

- A distância média entre as características extraídas de um indivíduo treinado e, as características de amostras do mesmo indivíduo não foi, em nenhuma situação, maior que 0,26;
- A distância média entre as características extraídas de um indivíduo treinado e, as características de amostras que não correspondem ao indivíduo não foi, em nenhuma situação, menor que 0,35;

Define-se que, para uma distância de limite de 0,30 garante que o usuário será reconhecido corretamente. Baseado neste resultado, este valor foi adicionado à função de reconhecimento de usuário no método CreateSpeaker.py.

# 6. Conclusões

A plataforma Raspberry Pi tem demonstrado eficiência, superando as expectativos e sem apresentar nenhuma dificuldade de processamento.

Os mecanismos conversores de texto para fala ainda possuem um certo grau de limitação devido ao atual conhecimento de linguística. Já para os mecanismos de reconhecimento de fala, os recursos computacionais e a aplicação dos conhecimentos de linguística são fatores limitantes.

O processamento fornecido pelo Wit.ai é, por sua vez, eficiente na função a que é atribuido. As dificuldades e diferenças na conversão de fala para texto são atribuídas tanto ao sotaque do indivíduo como do sistema Wit.ai. Quando os testes utilizaram uma voz capaz de produzir palavras na lingua inglesa sem vícios, o reconhecimento foi um sucesso. A dificuldade no reconhecimento do comando de acionamento “Jasper” é também atribuída ao Wit.ai. Uma solução seria a utilização do outro conversor de fala para texto, como o Google Speech API. Apesar deste ter se provado um dos mais eficientes mecanismos de conversão de fala-para-texto devido a sua alta precisão, sua utilização é limitada por conta do próprio desenvolvedor, sua versão gratuita possui um número máximo de minutos para a implementação de seu serviço. Uma opção para diminuir falhas é a modificação do comando de acionamento “Jasper”, alterando para uma palavra que resulte em menos erros para falantes cujo idioma nativo não seja o inglês.

O ajuste do Piwho utilizando na configuração: pré-processamento com algorítmo Endpoint, extração de características por LPC e classificação por Distância de Chebyshev, demostrou um ótimo reconhecimento, identificando corretamente cada usuário de acordo com suas respectivas amostras de voz. Mostrou-se também que a quantidade de módulos treinados afeta a classificação das distâncias. Com o aumento do número de treinamentos, aumenta a aproximação entre as características da voz de um indivíduo e as características de uma amostra de audio do mesmo. Esse resultado possibilitou a implementação de um método que identifica com precisão o usuário, sem aproximar para o indivíduo cadastrado com as características mais próximas.

Sobre a área de controle de sistemas por meio da voz, pode-se concluir que os avanços continuarão seguindo os estudos em linguística e aprimoramentos de técnicas de inteligência artificial. Percebe-se que grandes empresas no ramo de desenvolvimento de novas tecnologias têm apostado nessa área, com projetos ainda em desenvolvimento, que podem num fututo mudar o modo como a humanidade interage com o ambiente a sua volta.

## Referências


Admane, M. P.; RATNAPRABHA, J. **Speech to Text and Accelerometer based SmartPhone Interaction System**. ICICES2014 - S.A.Engineering College, Chennai, Tamil Nadu, India 2014.

ASH, M. **How Cortana Comes to Life in Windows 10.** Disponível em: <https://blogs.windows.com/windowsexperience/2015/02/10/how-cortana-comes-to-life-in-windows-10/#9kSs4udLFil6hhep.97> Acesso em: 22 mar. 2017.

BECHARA, E. **Moderna Gramática Portuguesa**. Rio de Janeiro 2009.

BLACK, A. W.; LENZO, K. A. **Flite: a small, fast speech synthesis engine**. 2014 Disponível em: <http://www.festvox.org/flite/doc/flite.pdf> Acesso em: 26 nov. 2016.
Google Speech API Disponível em: <https://cloud.google.com/speech/> Acesso em: 26 nov. 2016.

BOSKER, B **Siri Rising: The Inside Story Of Siri's Origins (And Why She Could Overshadow The iPhone).** <http://www.huffpostbrasil.com/entry/siri-do-engine-apple-iphone_n_2499165> Acesso em: 22 mar. 2017.

Carnegie Mellon University. **Flite**
Disponível em: <http://www.festvox.org/flite/doc/index.html> Acesso em: 26 nov. 2016.

CMUSphinx, **Basic Concepts of speech recognition** Disponível em: <http://cmusphinx.sourceforge.net/wiki/tutorialconcepts> Acesso em: 30 nov. 2016.

CMUSphinx, **Building Language Model** Disponível em: <http://cmusphinx.sourceforge.net/wiki/tutoriallm?s[]=cmuclmtk> Acesso em: 29 nov. 2016.

IMDb. **IMDb Database Statistics**, Disponível em: <http://www.imdb.com/stats> Acesso em: 02 março de 2017.

Khandkar, A. **Piwho.** Disponível em: <https://github.com/Adirockzz95/Piwho/>Acesso em: 20 mar. 2017

Kleback, M. **Tutorial: Raspberry Pi GPIO Pins and Python.** Disponível em: <http://makezine.com/projects/tutorial-raspberry-pi-gpio-pins-and-python/> Acesso em: 22 mar. 2017.

MARSH, C.; SHUBHRO S. **Jasper Project.** 2014 Disponível em: < http://jasperproject.github.io/> Acesso em: 01 jan. 2017.

MICROSOFT, **Cortana.** Disponível em: <https://www.microsoft.com/en-us/windows/cortana> Acesso em: 22 mar. 2017.

O'Malley, M. H. **Text-To-Speech Conversion Technology.** Agosto 1990.

PEREIRA, A. P. **Como funciona o reconhecimento de voz**. Disponível em: <https://www.tecmundo.com.br/curiosidade/3144-como-funciona-o-reconhecimento-de-voz-.htm>Acesso em: 20 nov 2016.

SEABROOK, J. **Hello, Hal.** Disponível em: <http://www.newyorker.com/magazine/2008/06/23/hello-hal> Acesso em: 22 mar. 2017.

The MARF Research and Development Group Modular. **Audio Recognition Framework v.0.3.0.6 (0.3.0 final) and its Applications**. Montréal, Québec, Canada. Dezembro 2007.

The Raspberry Pi Foundation, Disponível em: <https://www.raspberrypi.org> Acesso em: 21 nov. 2016.

The Raspberry Pi Foundation, **Raspbeerry Pi Software Guide.** Disponível em: <https://www.raspberrypi.org/learning/software-guide/> Acesso em: 21 nov. 2016.

The Raspberry Pi Foundation, **Getting Started With you Raspberry Pi.** Disponível em: <https://www.raspberrypi.org/help/videos/#noobs-setup> Acesso em: 21 nov. 2016.

WIT.AI. **WIT.AI** 2015 Disponível em: < https://wit.ai/ > Acesso em 22. Mar. 2017.

Xbox Wire Staff**. The Ultimate Assistant: Halo's Cortana Coming to Windows Phone**. Disponível em: <https://news.xbox.com/2014/04/03/games-cortana-companion-blog-post> Acesso em: 22 mar. 2017.

ZUCKERBERG, M. **Building Jarvis.** Disponível em: <https://www.facebook.com/notes/mark-zuckerberg/building-jarvis/10154361492931634>. Acesso em: 19 de dez. 2016.

## Apêndices

### CaptureImage.py

```
import re
import os
import datetime
import pygame
import pygame.camera
from pygame.locals import *

WORDS = ["PICTURE", "YES"]

def yes(text):
    return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))

def isValid(text):
    return bool(re.search(r'\b(picture)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    now = datetime.datetime.now()
    year = str(now.year)
    if(now.month<10):
        month = '0' + str(now.month)
    else:
        month = str(now.month)

    if(now.day<10):
        day = '0' + str(now.day)
    else:
        day = str(now.day)

    if(now.hour<10):
        hour = '0' + str(now.hour)
```

```
    else:
        hour = str(now.hour)

    if(now.minute<10):
        minute = '0' + str(now.minute)
    else:
        minute = str(now.minute)

    if(now.second<10):
        second = '0' + str(now.second)
    else:
        second = str(now.second)

    fileName = year + month + day + '_' + hour + '-' + minute + '-' + second
   fileName = fileName.lower()
    name = os.path.join('/var/www/html/images/JasperImages', fileName + ".jpg")
    pygame.init()
    pygame.camera.init()
    cam = pygame.camera.Camera("/dev/video0",(300,250))
    cam.start()
    image = cam.get_image()
    pygame.image.save(image, name)
    cam.stop()
    mic.say('Image captured successfully')
```

**CreateNewUser.py**

```
import re
import os
import shutil
from piwho import recognition

import datetime
```

```
import pygame
import pygame.camera
from pygame.locals import *


WORDS = ["SPEAKER", "YES"]


def yes(text):
    return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))

def isValid(text):
    return bool(re.search(r'\b(speaker)\b', text, re.IGNORECASE))

def handle(text, mic, profile):

        now = datetime.datetime.now()
        year = str(now.year)
        if(now.month<10):
            month = '0' + str(now.month)
        else:
                month = str(now.month)

        if(now.day<10):
                day = '0' + str(now.day)
        else:
                day = str(now.day)

        if(now.hour<10):
                hour = '0' + str(now.hour)
        else:
                hour = str(now.hour)

        if(now.minute<10):
```

```
        minute = '0' + str(now.minute)
    else:
        minute = str(now.minute)

    if(now.second<10):
        second = '0' + str(now.second)
    else:
        second = str(now.second)

    fileName = year + month + day + '_' + hour + '-' + minute + '-' + second
      print fileName



      #Ask for permission
      mic.say('Do you have permission to create a new user?')
    user = mic.activeListen2()
      #Call function to identify_speaker and check permission
      recog = recognition.SpeakerRecognizer()
      name = []
      name =
recog.identify_speaker('/home/pi/recordings/Reconhecer/recordingToRecognize.wav')
      x=False
      i=0
      print(name[0])
      dist = recog.get_speaker_scores()
      while x == False:
        if dist.keys()[i] == 'TIAGO':
          value = dist.values()[i]
            x = True
        else:
            i = i + 1
      print(value)
      if float(value) < 0.37:
        permission = True
```

```
        else:
            permission = False
        if permission==True:
            #New User's name
            mic.say('What is the new user name?')
            new_user = mic.activeListen()
            #First sentence
            mic.say('You have to say two sentences. Go ahead say the first one after the beep')
            user = mic.activeListen2()

shutil.move("/home/pi/recordings/Reconhecer/recordingToRecognize.wav","/home/pi/recordings
/speaker_NewUser/new_user1.wav")
            mic.say('Now the second sentence please')
            #Second sentence
            user = mic.activeListen2()

shutil.move("/home/pi/recordings/Reconhecer/recordingToRecognize.wav","/home/pi/recordings
/speaker_NewUser/new_user2.wav")
            #Ask if can continue
            mic.say('If everything is correct, and with that I mean, if the new user said correctely
the two sentences, say YES, otherwise you will have to start everything again')
            response = mic.activeListen()
            if yes(response):
                #Train the model
                    recog = recognition.SpeakerRecognizer('/home/pi/recordings/speaker_NewUser')
                    recog.speaker_name = new_user
                    recog.train_new_data()
                mic.say('New user created, your name is bla')
            else:
                    mic.say('You did everything wrong, start it again!')
        #Permission not granted, which means name[0] != 'TIAGO', create .txt file to inform the
intruder
        else:
            mic.say('You are not allowed to perform this action')
```

```
            mic.say('Informing intruder')
            name = os.path.join('/home/pi/NoteFiles', "permissionIntruder.txt")
            #name = os.path.join('/home/pi/JasperIntruder', "permissionIntruder" + ".txt")
            f = open(name, "a")
            print('OPEN')
            f.write(fileName + ' \n')
          name = os.path.join('/var/www/html/images/JasperIntruder/', fileName + ".jpg")
            os.system('fswebcam --no-banner -S 3 --jpeg 50 --save ' + name)
            f.close()
        mic.say('DONE')
```

**LED_ON.py**

```
import re
import RPi.GPIO as GPIO

WORDS = ["LIGHT"]

def isValid(text):
    return bool(re.search(r'\b(light)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)
    GPIO.setup(7, GPIO.OUT)
    GPIO.output(7, GPIO.LOW)
    mic.say('light is on')
```

**LED_OFF.py**

```
import re
import RPi.GPIO as GPIO

WORDS = ["DARK"]
```

```
def isValid(text):
    return bool(re.search(r'\b(dark)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)
    GPIO.setup(7, GPIO.IN)
    mic.say('light is off')
```

**Light_ON.py**

```
import re
import RPi.GPIO as GPIO

WORDS = ["LAMP"]

def on(text):
    return bool(re.search(r'\b(on)\b', text, re.IGNORECASE))

def isValid(text):
    return bool(re.search(r'\b(lamp)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    mic.say('Should I turn it on or off?')
    response = mic.activeListen()
    GPIO.setmode(GPIO.BOARD)
    if on(response):
        GPIO.setup(7, GPIO.OUT)
            mic.say('led is on')
    else:
            GPIO.output(7, GPIO.IN)
        mic.say('led is off')
```

**Recognition.py**

```
import re
import os
from piwho import recognition

WORDS = ["RECOGNIZE", "YES"]

def yes(text):
    return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))

def isValid(text):
    return bool(re.search(r'\b(recognize)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    mic.say('Say something, I will guess who you are')
    sentence = mic.activeListen2()
    print('mic ok')
    recog = recognition.SpeakerRecognizer()
    print('recog ok')
    name = []
    print('name ok')
    name =
recog.identify_speaker('/home/pi/recordings/Reconhecer/recordingToRecognize.wav')
    print('name = ok')
    print(name[0])
    person = name[0]
    mic.say(person)
```

**RecordAudio.py**

```
import re
import os
from piwho import recognition
```

```
WORDS = ["RECORD"]

def isValid(text):
    return bool(re.search(r'\b(record)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    mic.say('What do you want me to record?')
    record = mic.activeListen3()
    mic.say('Audio recorded')
```

**Search.py**

```
import re
import wikipedia

WORDS = ["SEARCH", "SET", "YES"]



def yes(text):
    return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))

def isValid(text):
    return bool(re.search(r'\b(search|set)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    mic.say('What would you like me to search about?')
    movie_name = mic.activeListen()
    mic.say('Searching about.  %s' %movie_name)
    final = wikipedia.summary(movie_name, sentences=2)
    mic.say(final)
```

**WriteDown.py**

```
import re
import os

WORDS = ["WRITE", "RIGHT", "YES"]

def yes(text):
    return bool(re.search(r'\b(yes)\b', text, re.IGNORECASE))

def isValid(text):
    return bool(re.search(r'\b(write|right)\b', text, re.IGNORECASE))

def handle(text, mic, profile):
    mic.say('What will be the files name?')
    fileName = mic.activeListen()
    fileName = fileName.lower()
    print(fileName)
    name = os.path.join('/home/pi/NoteFiles', fileName + ".txt")
    f = open(name, "w")
    mic.say('What do you want me to write down?')
    response = mic.activeListen()
    #f = open("notes.txt", "w")
    bool = type(response) is unicode
    print bool
    if(bool):
        f.write(response + '\n')
            print(response)
    else:
            mic.say('You said nothing')

    mic.say('Anything else?')
    response = mic.activeListen()
    bool = type(response) is unicode
    print bool
    if(bool==True):
```

```
    while yes(response):
            mic.say('What else would you like me to write down?')
            response = mic.activeListen()
            bool = type(response) is unicode
        if(bool):
            f.write(response + '\n')
                print(response)
        else:
                mic.say('You said nothing')
            mic.say('Anything else?')
            response = mic.activeListen()
else:
    mic.say('Nothing else to write down')
f.close()
mic.say('Everything was wrote')
```