
# SISTEMA DE RECONHECIMENTO DE FALA BASEADO EM POCKETSPHINX PARA ACESSIBILIDADE EM ANDROID

Bruno Lima Cardoso

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Fernando Gil Vianna Resende Junior

Rio de Janeiro
Outubro de 2015

SISTEMA DE RECONHECIMENTO DE FALA BASEADO EM POCKETSPHINX PARA ACESSIBILIDADE EM ANDROID

Bruno Lima Cardoso

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Fernando Gil Vianna Resende Junior, Ph.D.

Prof. José Antonio dos Santos Borges, Ph.D.

Prof. Mariane Rembold Petraglia, Ph.D.

Prof. Abraham Alcaim, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
OUTUBRO DE 2015

Cardoso, Bruno Lima

Sistema de Reconhecimento de Fala Baseado em Pocketsphinx para Acessibilidade em Android/Bruno Lima Cardoso. – Rio de Janeiro: UFRJ/COPPE, 2015.

XIV, 69 p.: il.; 29, 7cm.

Orientador: Fernando Gil Vianna Resende Junior

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2015.

Referências Bibliográficas: p. 49 – 51.

1. Reconhecimento de Fala. 2. Pocketsphinx. 3. Acessibilidade. 4. Android. I. Resende Junior, Fernando Gil Vianna. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

*Dedico este trabalho a minha mãe e irmã que muito mais que apoio moral participaram ativamente no seu processo de construção e sem elas certamente seria impossível de completar.*

# Agradecimentos

Agradeço imensamente ao professor Fernando Gil que aceitou esta laboriosa tarefa de me orientar, pois na maior parte do tempo esta foi remota devido as minhas condições de locomoção e em momento algum exitou em me ajudar.

Agradeço ao professor José Antônio Borges, que me acompanha desde o início da graduação com quem tive o privilégio de trabalhar no Projeto Motrix e que, graças a ele, escolhi esta carreira acadêmica. Também agradecê-lo por ter aceito compor a minha banca.

Agradeço a professora Mariane Petraglia por ter aceito fazer parte da minha banca e pelas carinhosas palavras num dos momentos mais críticos do trabalho que atenuaram toda pressão.

Agradeço ao professor Abraham Alcaim por gentilmente também ter aceito participar da minha banca apesar dos meus embaraços e urgência.

Agradeço a COPPE-UFRJ por ter estudado numa das melhores instituições de ensino do mundo.

Agradeço a CAPES pela bolsa de Mestrado durante os dois primeiros anos do curso.

Agradeço a Daniele Cristina O. da Silva, que muitas vezes me salvou com prazos, inscrições e orientações sempre me atendendo com muito carinho, respeito e atenção.

Agradeço a meus familiares e amigos, os quais não citarei por medo de injustamente esquecer de alguns de seus nomes.

E, por fim, gostaria de agradecer a todos que contribuíram de alguma forma para este trabalho.

Muito Obrigado!

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

# SISTEMA DE RECONHECIMENTO DE FALA BASEADO EM POCKETSPHINX PARA ACESSIBILIDADE EM ANDROID

Bruno Lima Cardoso

Outubro/2015

Orientador: Fernando Gil Vianna Resende Junior

Programa: Engenharia Elétrica

Apresenta-se, nesta dissertação, o desenvolvimento de um sistema para navegação por voz em um celular munido de Sistema Operacional (SO) *Android* na versão 4.1 (ou superior) no formato *command-control* baseado em *Pocketsphinx*. Seu objetivo é permitir que pessoas com lesões severas utilizem seus celulares sem auxílio de terceiros.

Após a análise teórica da arquitetura deste SO, foi feito o embarcamento do sistema de reconhecimento de fala e, em seguida, o desenvolvimento da estrutura de navegação com base nos comandos obtidos. Também foi desenvolvido um módulo de notificação responsável por informar ao usuário o último comando compreendido.

Por fim, foram executados testes de performance da ferramenta de reconhecimento onde contabilizou-se a taxa média de acerto e o tempo médio de execução dos comandos, assim como seus respectivos desvios-padrões. Este estudo se deu pela variação de um conjunto de parâmetros oferecidos pela mencionada ferramenta para melhoria de desempenho. A melhor taxa de acerto obtida foi de 94%, com tempo médio de execução de 125 ms.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SPEECH RECOGNITION SYSTEM BASED ON POCKETSPHINX FOR ACCESSIBILITY ON ANDROID

Bruno Lima Cardoso

October/2015

Advisor: Fernando Gil Vianna Resende Junior

Department: Electrical Engineering

In this work, the development of a command-control navigation system for a cellphone with *Android* 4.1 (or greater) as Operation System (OS) based on Pocketsphinx is presented. The main goal of this project is to allow people with severe disability to use theirs cellphones without any help from others.

After theoretical analisys of the OS architecture, the speech engine was embedded, and then, the development of the navigation module based on inputs from the speech engine was performed. Also, a notification module was coded to give feedback to the user about the last recognized command.

Finally, qualitative tests of the speech engine computing the average accuracy and time spent for recognition, as well as the corresponding standard deviation, were executed. The study was based on the variation of a set of parameters provided by the framework to improve performance. The best accuracy obtained was 94% with average execution time of 125 ms.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Siglas

API – Application Programing Interface

ASR – Automatic Speech Recognition

CMN – Cepstral Mean Normalization

CSR – Continuous Speech Recognition

FDP – Função Densidade de Probabilidade

FGV – Fundação Getúlio Vargas

FIFO – First In First Out

HMM – Hidden Markov Models

IDC – International Data Corporation

IME – Input Method Editor

JSGF – Java Speech Grammar Framework

JVM – Java Virtual Machine

MFCC – Mel-Frequency Cepstral Coefficients

NDK – Native Development Kit

OS – Operation System

PCD – Pessoa Com Deficiência

PSD – Pessoa Sem Deficiência

SDK – Software Development Kit

SO – Sistema Operacional

TA – Tecnologia Assistiva

TTS – Text to Speech

# Lista de Variáveis

t – unidade de tempo

$x_t$ – vetor de parâmetros calculado a partir de um segmento de fala

$c_t$ – vetor de MFCCs calculado a partir de um segmento de fala

$\Delta c_t$ – vetor de MFFCs delta de primeira ordem computados a partir dos coeficientes $c_t$

$\Delta\Delta c_t$ – vetor de MFFCs delta de segunda ordem computados a partir dos coeficientes $\Delta c_t$

$\Theta$ – tamanho da janela usada para cálculo dos coeficientes delta de primeira e segunda ordens

M – um modelo oculto de Markov

X – sequência de vetores acústicos

$A = a_{ij}$ – matriz de todas as probabilidades de transição entre um estado i e outro j

$B = b_i$ – matriz de todas as probabilidades de emissão de saída em um determinado estado i

$\prod = \prod_i$ – matriz com as probabilidades de um modelo ser iniciado a partir de um estado i

S – sequência de estados

W – sequência de palavras

TxA – Taxa de commandos reconhecidos corretamente

TxE – Taxa de commandos reconhecidos incorretamente

# Capítulo 1

# Introdução

## 1.1 Motivação

Com o avanço da tecnologia, os sistemas computacionais têm estado cada vez mais presentes no dia a dia das pessoas. Isso se dá tanto pela diminuição dos custos de produção e aquisição de seus diversos componentes quanto pelas facilidades proporcionadas por eles. Tais facilidades, além de promoverem uma maior comodidade e praticidade ao público geral, se bem empregados, propiciam uma melhor qualidade de vida e independência aos portadores de necessidades especiais (Pessoas com Deficiência – PCD). O exemplo mais simples está na utilização de *smartphones*. Hoje estes vão muito além de meros aparelhos de comunicação, servem como agendas, despertadores, bloco de notas, organizadores pessoais e muito mais.

Há um número vertiginoso de aparelhos celulares sendo utilizados por pessoas comuns e, neste universo, há três empresas que lideram o mercado de SOs: *Google* com *Android*, a *Apple* com o *iOS* e a *Microsoft* com o *Windows Phone*. De acordo com *International Data Corporation* (IDC), empresa que acompanha o *Market Share* dos SOs pelo mundo, o *Android* tem ampla vantagem perante aos outros com 78% dos dispositivos, seguido pelo *iOS* com 18% e o *Windows Phone* com 2.7% (Figura 1.1).

Apesar da sua supremacia, o *Android* ainda possui uma deficiência considerável quando o assunto é acessibilidade. Este provê estruturas que permitem desenvolvedores fora da *Google* criarem aplicações neste tema, porém o conjunto de aplicativos desenvolvidos pela empresa para estarem presentes nos celulares com este SO é muito limitado. O exemplo mais conhecido é o *TalkBack* [2] que auxilia cegos a navegarem pelo celular. Entretanto, no que tange a pessoas com limitações severas, sejam elas com mobilidade reduzida ou ausência de mobilidade, nade se encontra. *Smartphones* utilizam telas acionadas por toque, *touchscreens*, e toda interação, ou pelo menos a maior parte dela, se dá desta forma, o que remove qualquer utilização por este grupo. Tendo isto em mente, este trabalho tenta mitigar este problema

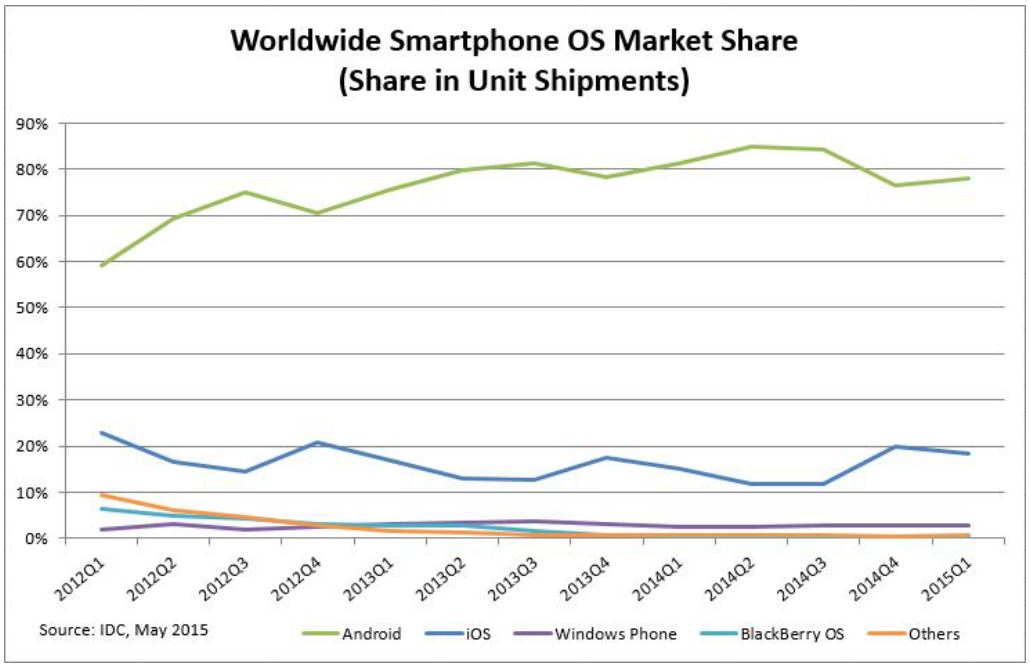


Figura 1.1: IDC *Market Share*

propondo uma nova abordagem de como o usuário PCD opera o aparelho.

A ideia é utilizar um sistema de reconhecimento de voz que emule o acionamento do dispositivo sem a necessidade de interação física. Para tal, é necessário o embarcamento de uma ferramenta de reconhecimento, pois não há nenhuma disponível neste SO por padrão. Há algumas alternativas que podem ser utilizadas e, dentre elas, as mais conhecidas são: *HTK* [3] e o *Sphinx* [4]. Para este trabalho foi escolhido o *Pocketsphinx*, versão compacta do *Sphinx* voltada para dispositivos móveis. Esta escolha se deu pois a *Microsoft* detém os direitos autorais do *HTK* o que impede a livre utilização e comercialização de qualquer aplicação que a utilize.

## 1.2 Objetivos

Os principais objetivos desta dissertação são:

- Analisar a eficiência dos recursos voltados para acessibilidade;
- Desenvolver como prova de conceito um sistema de reconhecimento de voz que permita a um PCD executar o mesmo conjunto de ações básicas (*scroll* e *click*) executadas por um PSD em um *smartphone* usando apenas a voz;
- Executar uma análise de performance do *framework Pocketsphinx* embarcado no dispositivo.

## 1.3 Descrição

Este trabalho está organizado da seguinte forma: o Capítulo 2 contém o referencial teórico no qual este trabalho se baseia. O Capítulo 3 descreve o processo de construção do conhecimento, assim como as etapas da concepção do sistema. No Capítulo 4 estão apresentados os testes comparativos. Por fim, no Capítulo 5, é feito um relato a respeito de todo o processo assim como são apresentadas as propostas de trabalhos futuros.

# Capítulo 2

# Fundamentação Teórica

## 2.1 Design Universal

No início da década de 60 o inglês Selwyn Goldsmith desenvolveu o manual de planejamento arquitetural com intuito de prover orientação ao desenvolvimento de estruturas que garantam acessibilidade a prédios e construções para pessoas com deficiência. Este trabalho ficou conhecido como *Designing for the Disabled*[5]. De todos os seus projetos o mais famosos é o desnível em calçadas para saída de cadeira de rodas. Esta estrutura se tornou padrão para calçadas acessíveis no mundo inteiro e é utilizada até os dias atuais.

Anos depois, na década de 90, o arquiteto Ronald L. Mace, fundador e diretor do *The Center for Universal Design* da *North Caroline State University*, introduziu o conceito Design Universal (*Universal Design*)[6]. Este tem como objetivo central o desenvolvimento de produtos e espaços mais seguros, mais fáceis e mais convenientes para todos, independente de suas limitações físicas.

Junto a um grupo de arquitetos, designers de produtos, engenheiros e designers de ambiente, foram desenvolvidos os sete princípios básicos que suportam o Design Universal. São eles:

- *Equitable Use* - Usabilidade imparcial
- *Flexibility Use* - Usabilidade flexível ou adaptável
- *Simple and Intuitive* - Simples e intuitivo
- *Perceptible Information* - Informações claras e de fácil acesso
- *Tolerance for Error* - Tolerante a erros
- *Low Physical Effort* - Mínimo de esforço possível

- *Size and Space for Approach and Use* - Tamanhos e medidas adequadas para utilização

Com base nestes, o processo de criação e concepção passa a ser incremental onde a cada etapa os recursos se tornam mais universais abrangendo um grupo maior de usuários, como ilustrado na Figura 2.1.

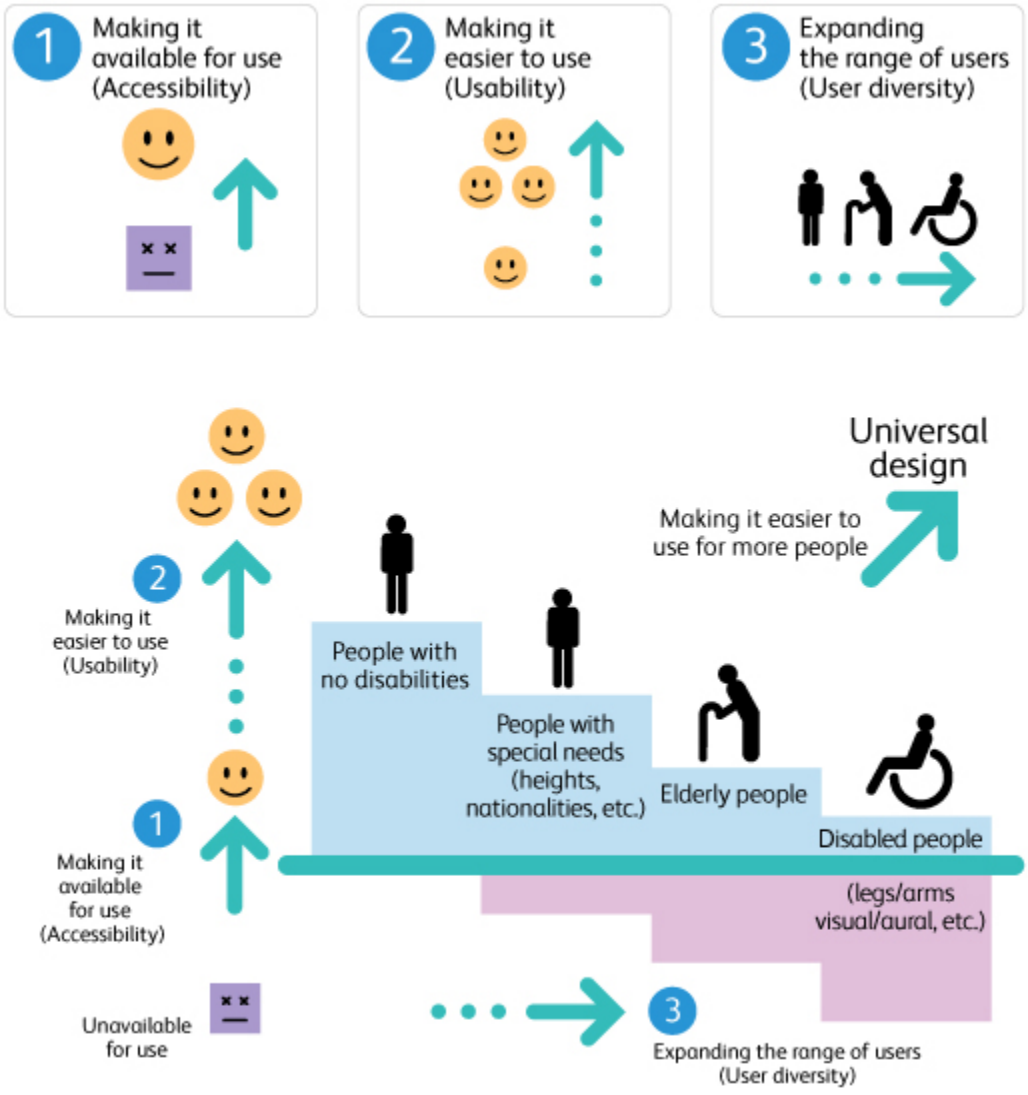


Figura 2.1: *Android* com *feedback* visual ativado[1]

Apesar de sua origem na arquitetura, estes princípios vem sendo utilizados com cada vez mais frequência em outros campos. O exemplo mais claro e que muitas vezes passa despercebido é a utilização do velcro no setor têxtil. Este além de ser

[1]Extraído de [6]

extremamente forte também é de muito fácil utilização por aqueles que possuem limitação, logo muitas marcas o utilizam em seus produtos: roupas, calçados e etc.

Assim como o velcro, é possível ver aplicação do Design Universal no campo da tecnologia da informação. Uma das principais missões do sistema operacional *Android* é: "Organizar as informações do mundo e fazê-las universalmente acessíveis e úteis"[7]. Para tal, os padrões de design deste foram criados de acordo com os sete princípios e, com efeito, a acessibilidade tornou-se a métrica para avaliar o quão bem o produto é capaz de ser utilizado por pessoas com habilidades diferentes.

## 2.2 *Android* e os Sete Princípios

No *Android* a chave para acessibilidade reside em sempre saber onde se está (*"I should always know where I am"*). Conforme o usuário navega pelas telas, ele sempre necessitará de informações (*feedbacks*) para montar um modelo mental de sua atual localização, assim o usuário se beneficiará da sensação de informação estruturada e uma arquitetura que faça sentido. Com isto em mente, os engenheiros da *Google* utilizam um sistema de rótulos onde cada componente da interface com algum significado recebe um rótulo e uma descrição.

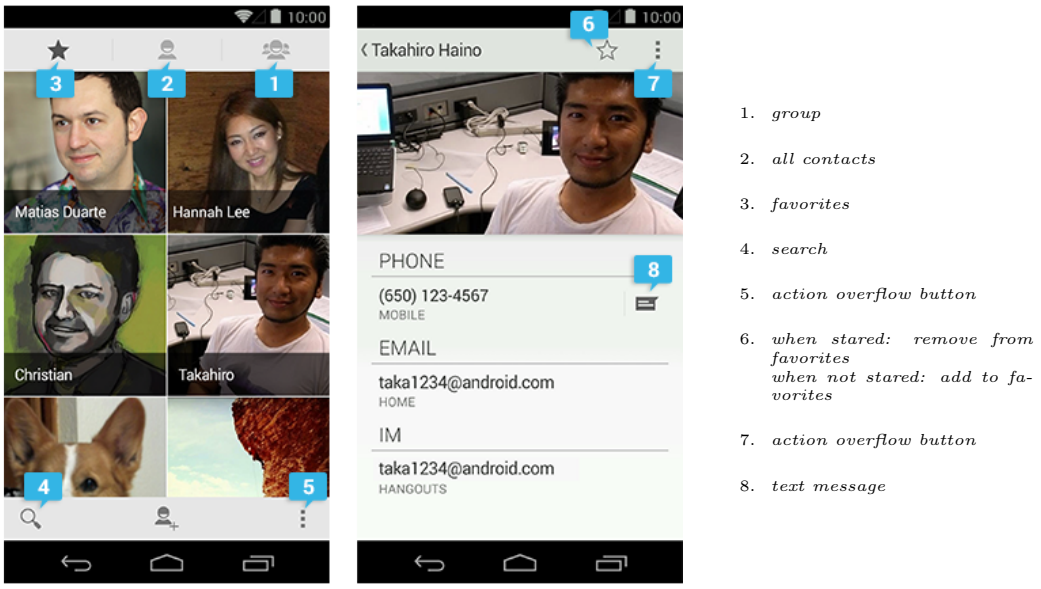


Figura 2.2: Exemplo de rótulos no *Android*[2]

Como pode ser visto na Figura 2.2, todos componentes da interface possuem um rótulo condizente com aquilo que eles fazem, desta forma, as ferramentas de tecnologia assistivas são capazes de informar ao usuário exatamente aonde ele está e qual operação poderá ser executada ao clicá-lo.

[2]Extraído de [7]

(a) *Trackball*


(b) *D-Pad*

Figura 2.3: Dispositivos externos para navegação

O exemplo mais claro de utilização deste sistema vem do aplicativo já citado voltado para deficientes visuais chamado *Talkback*. Sua principal função é permitir a exploração por toque na tela, retornando informações ao usuário referente ao componente clicado. Isto se dá através de uma ferramenta de *Text-to-Speech* (TTS) nativa do SO. Todas as informações emitidas são recuperadas do rótulo e da descrição de cada componente, desta forma o usuário consegue criar um mapa mental de seu aparelho e poderá usá-lo sem receios.

O segundo recurso criado para prover acessibilidade está na navegação com foco. Nesta modalidade é permitido ao usuário navegar pela tela usando comandos direcionais com base no elemento atualmente focado, algo como "próximo", "direita", "esquerda" e assim por diante. Geralmente estes comandos são oriundos de algum dispositivo externo como *D-PADs* (Figura 2.3b) ou *Trackballs* (Figura 2.3a). Tomando por base a Figura 2.4, o campo com borda verde é aquele com foco no momento presente, logo, se um usuário munido de um *D-PAD* clicar na seta para baixo o foco mudará para o campo abaixo seguinte, no caso *Sound Volume*. A partir da versão 4.1 do *Android*, também é possível utilizar gestos direcionais na tela e conseguir o mesmo comportamento supracitado. Para que tudo isso seja possível, os componentes visuais recebem um outro atributo, *focusable*. Este é um atributo booleano que define quais elementos na interface são passíveis de receber foco.

## 2.3 *Android* e Reconhecimento da Fala

Apesar do esforço para fazer os dispositivos mais acessíveis, ainda há um grupo de usuários que foi deixado de lado. Todos os recursos anteriormente citados necessitam de algum movimento, mesmo que reduzido. Logo, PSDs com ausência completa de mobilidades em membros não conseguem utilizar um celular *Android*.

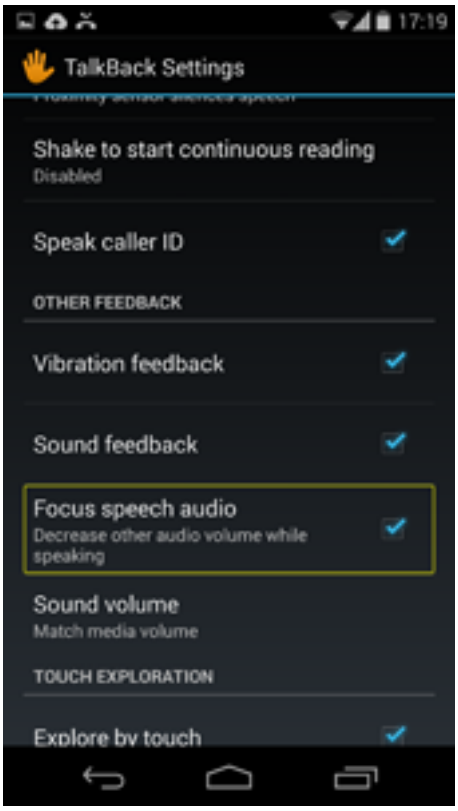


Figura 2.4: *Android* com *feedback* visual ativado

A alternativa é recorrer a sistemas de reconhecimento de fala e assim, através de comandos falados, este grupo será capaz de navegar no telefone. Entretanto, neste grupo, também há aqueles que além das limitações de mobilidades possuem limitações de fala diminuindo o alcance desta abordagem, mas isto não será discutido pois foge do escopo deste trabalho.

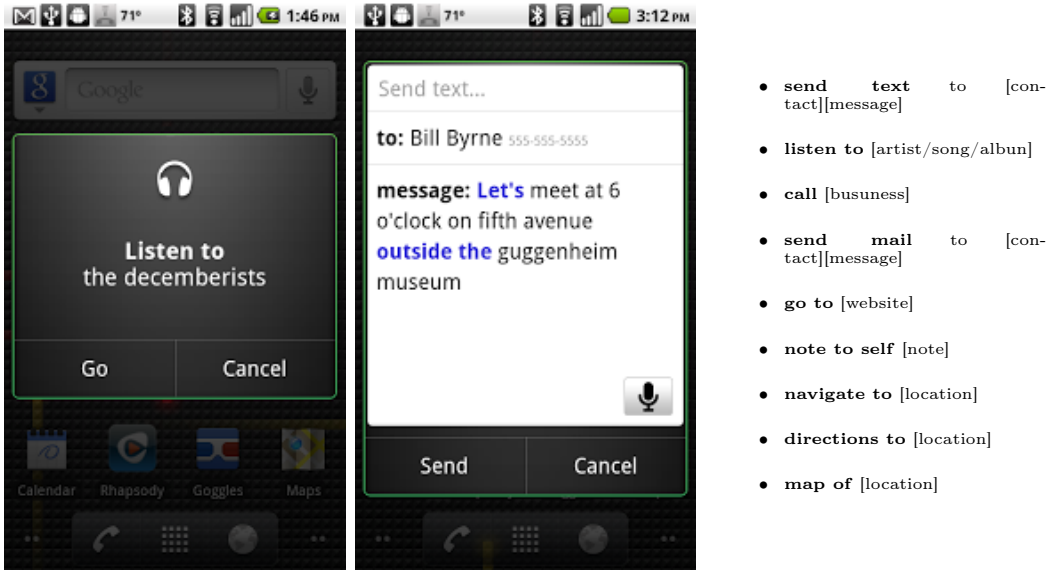


Figura 2.5: *Voice Actions*[1]

A *Google* apresentou em seu *Blog*[1] oficial no ano de 2010 um recurso chamado *Voice Actions*, mostrado na Figura 2.5, que consistia na primeira estrutura de reconhecimento de fala para a plataforma. Infelizmente, o sistema não foi inteiramente embarcado no dispositivo, este apenas serve de ponte para envio do áudio captado para o servidor remoto onde o real processamento ocorre. Daí, este retorna para o dispositivo uma lista com os resultados mais prováveis para então executar alguma ação. Como recurso de acessibilidade esta não é a melhor abordagem, pois, além do aplicativo ficar sujeito ao acesso a internet, para ativar o microfone era necessário clicar num botão e aí então falar o comando.

Desde aquela época o sistema vem evoluindo. Em 2013 no página do projeto *Eyes-free*[8], projeto voltado para o desenvolvimento de tecnologias assistivas para *Android*, foi disponibilizada a descrição do aplicativo chamado *JustSpeak*[9] que basicamente possuia as mesmas funções do *Voice Action*, mas sua ativação não dependia de clicar em um botão, era necessário apenas falar *Launch Google Now* para assim inserir o comando desejado. Hoje o serviço está bem mais robusto e preciso[10], mas o processamento da fala ainda é feito remotamente. E, curiosamente, o aplicativo não está disponível para downloads na loja virtual do *Android*.

O fato do sistema de reconhecimento estar remoto gera implicações que limitam sua utilização como tecnologia assistiva. Isto se baseia em sempre precisar haver acesso à internet para ser usado e, se por algum motivo houver perda de sinal, o usuário ficará ilhado sem nenhum alternativa. Além da necessidade de conexão, um outro problema vem da impossibilidade de treinar o sistema para um locutor específico, logo, com um sistema mais generalista a taxa de acerto tende a diminuir.

## 2.4 Reconhecimento de Fala

A implementação de um Sistema de Reconhecimento de Fala pode ser dividida em duas etapas: treinamento e teste. Previamente à etapa de treinamento é realizada a extração de parâmetros do sinal, ou seja, o áudio presente na base é transformado em coeficientes que servirão de entrada para a etapa de treinamento.

Na etapa de treinamento é realizado o treinamento do modelo acústico e do modelo de linguagem. O primeiro tem como propósito calcular a verossimilhança de uma sequência de vetores acústicos, enquanto o segundo tem por objetivo mapear os relacionamentos entre palavras, estimando a probabilidade de ocorrência de uma palavra em função das anteriores.

A fase de testes compreende o reconhecimento em si, também conhecido como decodificação. Esta fase representa não apenas um problema de reconhecimento de padrões como um problema de busca em um grafo, tendo por objetivo buscar a sequência de palavras que melhor se adapta aos vetores acústicos de entrada. Então,

realiza-se a análise dos resultados, retirando as medidas de interesse. O diagrama apresentado na Figura 2.6 ilustra, de forma resumida, as etapas mencionadas.

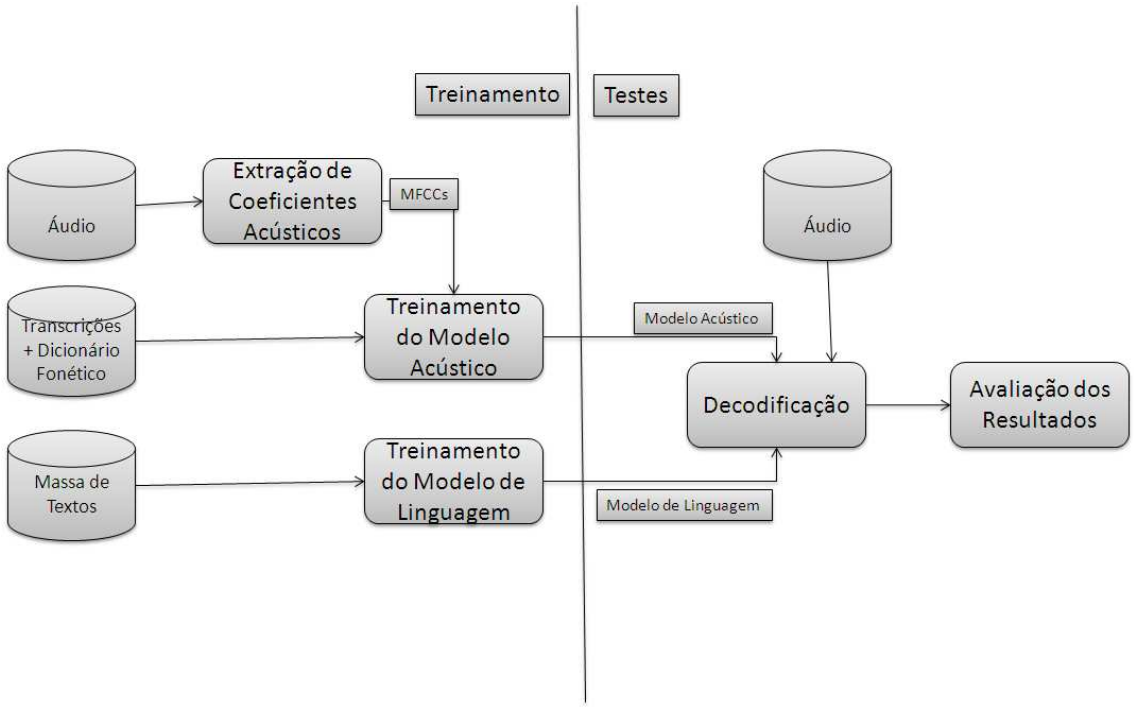


Figura 2.6: Diagrama das etapas envolvidas em um sistema CSR[3]

### 2.4.1 Extração de Parâmetros do Sinal

Para que seja possível processar o sinal de fala para geração do sistema de reconhecimento, torna-se necessária, em uma primeira etapa, a conversão da onda sonora em um sinal digital. O processamento do sinal de fala consiste na amostragem do sinal, janelamento e extração de parâmetros, que serão relevantes para o processo de reconhecimento. A extração dos parâmetros é um dos assuntos mais importantes na área de reconhecimento de fala.

A função principal desta etapa de extração de parâmetros é a da divisão do sinal em blocos – supondo que o sinal de fala pode ser considerado estacionário durante um período de tempo muito pequeno, de alguns milissegundos – e derivação de uma estimativa suavizada do espectro a partir de cada bloco. Em uma configuração considerada padrão, esses blocos – usualmente chamados de janelas – possuem duração de 25ms e são obtidos a cada 10ms (o que é conhecido por *frame shift*). Ainda, o sistema usa blocos sobrepostos de forma a capturar a informação contida nos seus limites.

Após essa fase, é aplicada uma transformada de Fourier a cada bloco, passando os valores obtidos do domínio do tempo para o domínio da frequência. Depois é

[3]Extraído de OLIVEIRA [11]

feita uma filtragem, com o objetivo de extrair os parâmetros que permitem o reconhecimento da fala. A saída é um vetor de valores filtrados, comumente chamados de mel-spectrum [12], cada um correspondendo ao resultado da filtragem do espectro de frequências da entrada por um filtro individual. Então, o comprimento do vetor de saída é igual ao número de filtros criados. As constantes que definem essa filtragem são o número de filtros (cujas frequências centrais estão em uma escala logarítimica, conforme o ouvido humano), a frequência mínima e a frequência máxima. As frequências máxima e mínima dependem da origem do sinal de voz. Para uma fala em um sistema de telefonia, com frequências de corte de 300 e 3700 Hz, o uso de limites fora desses valores significa apenas perda de banda.

Para uma fala clara, a frequência mínima deve estar acima de 100 Hz, para livrar-se de possíveis interferências da corrente alternada (50/60 Hz) e também porque normalmente não há informação útil abaixo desta frequência.

A frequência máxima deve ser menor que a frequência de Nyquist, ou seja, menor que metade da frequência de amostragem. Além disso, acima de 6800 Hz não há muito que possa ser usado para melhorar a separação entre os modelos. Para canais muito ruidosos, uma frequência máxima em torno de 5000 Hz deve ajudar a diminuir o ruído.

Davis e Mermelstein [13] mostraram que os coeficientes de frequência Mel-cepstrais apresentam características robustas para um bom reconhecimento de fala.

A Figura 2.7 ilustra de forma resumida o processo de extração de parâmetros do sinal.

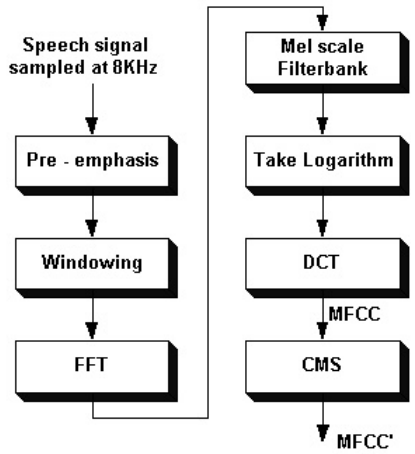


Figura 2.7: Diagrama do processo de extração de parâmetros de um sinal de voz[4]

[4]Extraído de OLIVEIRA [11]

### 2.4.2 Modelagem Acústica

A modelagem acústica consiste em um método para calcular a verossimilhança de uma sequência de vetores X, dado um modelo M. Os Modelos Ocultos de Markov ou *Hidden Markov Models (HMMs)* são considerados o estado da arte para modelagem acústica.

Os HMMs podem modelar uma unidade menor da palavra (como os fonemas ou mesmo fonemas inseridos em contextos diferentes, como difones ou trifones), uma palavra, ou ainda uma frase inteira.

Estes podem ser vistos como máquinas de estado finitas, onde a cada unidade de tempo ocorre uma transição entre estados e cada estado emite um vetor acústico com uma função densidade de probabilidade associada. Um modelo M pode ser escrito como na Equação 2.1.

$$M = A, B, \Pi = a_{ij}, bi, \pi_i, i, j = 1, \ldots, N \tag{2.1}$$

A Figura 2.8 representa um HMM com três estados emissores, onde são adicionados dois estados não emissores no início e no fim do modelo para facilitar a união entre modelos. Na figura, $x_t$ representa um vetor acústico emitido em uma unidade de tempo $t$ e a probabilidade de emissão de um vetor acústico por um estado é representada por uma mistura de gaussianas.

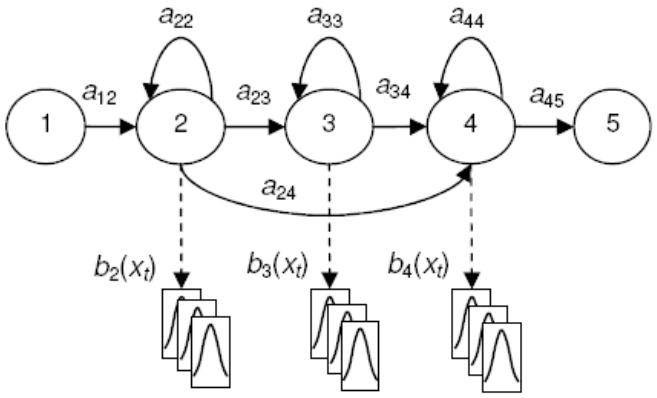


Figura 2.8: Exemplo de HMM com três estados emissores[5]

O objetivo da modelagem é encontrar o ajuste dos parâmetros do modelo que maximize a verossimilhança. Isto é feito através de um algoritmo de estimação de parâmetros. O algoritmo mais usado é o Baum-Welch, também conhecido como *Forward-Backward Algorithm* ou algoritmo de *avanço-retorno*. Além de todos os parâmetros presentes na mistura de gaussianas associada a cada estado, como médias e variâncias, as matrizes de transição entre estados também são consideradas

[5]Extraído de OLIVEIRA [11]

parâmetros do modelo. Também é necessário definir a taxa de convergência do algorítimo a qual, no *toolkit CMU Sphinx* segue a Equação 2.2.

$$convgRatio = \frac{(lkhdperframe - prevlkhd)}{absprev} \tag{2.2}$$

Onde *convgRation* é a taxa de convergência de determinada iteração, *lkhdperframe* é a verossimilhança por janela obtida da iteração corrente, *prevlkhd* é a verossimilhança por janela obtida da iteração anterior e *absprev* é o valor absoluto de *prevlkhd*.

Uma vez que através do modelo não é possível saber a sequência de estados, considera-se uma sequência de estados fixos e calcula-se a probabilidade a priori de todas as possíveis sequências de estados segundo as Equações 2.3 e 2.4.

$$P(X, S|M) = a_{s(0)s(1)} \prod_{t=1}^{T} b_{s(t)} x_t a_{s(t)s(t+1)} \tag{2.3}$$

Onde $a_{s(0)s(1)}$ é a probabilidade de começar no estado $s(1)$, $b_{s(t)}x_t$ é a probabilidade de emissão de um vetor $x_t$ em um estado $s(t)$ no tempo $t$ e $a_{s(t)s(t+1)}$ a probabilidade de transição do estado no tempo $t$ para outro no tempo $t+1$.

$$P(X|M) = \sum_{S} P(X, S|M) \tag{2.4}$$

De modo alternativo, $P(X|M)$ pode ser obtido pela maximização da Equação 2.3 utilizando o Algoritmo de Viterbi. Este também é utilizado no processo de decodificação onde é necessário apontar uma determinada sequência de estados mais provável para uma sequência de palavras, fones ou trifones desconhecidos.

As probabilidades $b_{s(t)}x_t$ são representadas por misturas de gaussianas segundo a Equação 2.5.

$$b_{s(t)}x_t = \sum_{m=1}^{M} c_{s(t)m} N(x_t; \mu_{s(t)m}, \Sigma_{s(t)m}) \tag{2.5}$$

Onde M representa o número de componentes na mistura, $c_{s(t)m}$ o peso do m-ésimo componente da mistura e $N(x_t; \mu_{s(t)m}, \Sigma_{s(t)m})$ é a gaussiana multivariada de acordo com a Equação 2.6.

$$N(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{2.6}$$

Onde o vetor de médias é definido por $\mu$, a matriz de convergência por $\Sigma$ e o número de componentes por $n$.

Ainda, de modo a otimizar esses modelos, pode-se utilizar uma técnica de compartilhamento de estados na ocasião do treinamento de fonemas inseridos em con-

textos diferentes, como no caso dos trifones. Esta técnica pressupõe que alguns fonemas inseridos em diferentes contextos podem não produzir uma variabilidade acústica suficientemente grande para que sejam modelados por HMMs diferentes. Assim, esses trifones podem ser agrupados dentro de um mesmo modelo, através de uma técnica de agrupamento de estados conhecida como *state-tying*. Para tal utilizam-se algoritmos de árvores de decisão de modo que constrói-se uma árvore para cada fonema, onde o nó pai corresponde a todos os estados com todos os diferentes contextos em que aquele fonema pode ser inserido (no caso de trifones, contexto à direita e contexto à esquerda), ou seja, mesmo fonema central. Os arcos dessa árvore correspondem a perguntas fonéticas que servirão para agrupar estados dentro de categorias diferentes e, por fim, os estados que caírem dentro de um mesmo nó folha serão agrupados como um único estado.

### 2.4.3 Modelo de Linguagem

O modelo de linguagem provê uma estrutura que define uma inter-relação entre as palavras. De forma geral, estas estruturas são classificadas entre duas categorias: gramáticas de grafos dirigidos ou modelos estocásticos N-Gram. As gramáticas de grafos dirigidos representam um grafo dirigido de palavras onde cada palavra encontra-se em um vértice e cada arco representa a probabilidade de transição entre estas duas palavras (vértices). Já o modelo estocástico gera probabilidades para as palavras baseado na observação das n-1 palavras anteriores (ver [14]). Como falado anteriormente, neste trabalho será utilizado um sistema de reconhecimento de palavras isoladas onde o modelo de linguagem é classificado como uma gramática de grafo dirigido onde cada aresta tem peso constante, ou seja, a probabilidade de transição entre palavras é constante para qualquer palavra.

### 2.4.4 Decodificação

A etapa de decodificação ocorre durante a fase de testes e consiste em uma busca pela sequência de palavras que melhor se adapta aos vetores acústicos, ou seja, realiza-se uma busca pela sequência de estados que maximiza a probabilidade a posteriori. Esta pode ser calculada através da fórmula de Bayes, como mostrado na Equação 2.7.

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)} \tag{2.7}$$

Uma vez que, para fins de reconhecimento de fala, o elemento observado não corresponde a um único elemento e sim a uma sequência de vetores acústicos, então, para uma sequência de X vetores, considerando a ocorrência de cada observação

como um evento independente, pode-se construir uma regra de decisão $\widehat{W}$ para o problema através da maximização da probabilidade a posteriori, como mostrado na Equação 2.8. Nesta equação, ainda é possível notar a eliminação do termo presente no denominador. A ausência deste termo é justificável, uma vez que ele será o mesmo para todas as classes testadas.

$$\begin{aligned}\widehat{W} &= \arg\max_{W} P(W|X) \\ &= \arg\max_{W} \frac{P(X|W)P(W)}{P(X)} \\ &= \arg\max_{W} P(X|W)P(W)\end{aligned} \tag{2.8}$$

A probabilidade a priori $P(W)$ será fornecida pelo modelo de linguagem e a probabilidade condicional $P(X|W)$, pelo modelo acústico. Logo, onde se lê probabilidade a priori, será lido como a probabilidade de ocorrer uma determinada sequência de palavras. E onde se lê probabilidade condicional, será lido como a probabilidade de observar uma determinada sequência de vetores dada uma sequência de palavras.

O algoritmo mais utilizado nesta etapa de decodificação é conhecido como algoritmo de Viterbi. Ele consiste em um algoritmo de busca síncrono, que procura pelo estado mais provável a cada unidade de tempo. Este algoritmo considera uma aproximação, conhecida por aproximação de Viterbi, como mostrado na Equação 2.9. Esta aproximação considera que o somatório da Equação 2.4 pode ser substituído pelo máximo, de forma a encontrar a melhor sequência de estados.

$$\widehat{W} = \arg\max_{W} P(W|X) \cong \arg\max_{W}[P(W) \max_{s_0^T} P(X, s_0^T|W)] \tag{2.9}$$

De forma a melhorar o desempenho da busca, o decodificador usa duas estratégias: guardar alguns caminhos ótimos intermediários, posto que alguns resultados possam ser usados por outros caminhos sem que haja necessidade de computá-los novamente, e realizar podas (*Pruning*) nos nós que se encontram abaixo de um limiar pré-estabelecido. Desta forma, a cada passo, o decodificador elimina caminhos que são muito provavelmente desnecessários, percorrendo apenas um feixe do espaço de busca. Este tipo de busca é conhecido como busca em feixe.

### 2.4.5 Melhorias

Há casos em que os modelos acústicos não atingem a acurácia desejada ou minimamente necessária para o bom desempenho da aplicação. Para contornar isto existem algumas soluções. A primeira seria fornecer mais dados na fase de treinamento dos modelos, isto é, produzir mais horas de gravações de áudio e por conse-

guinte melhorar o conjunto de características adquirido. A outra seria adaptar os modelos existentes a um único locutor fornecendo gravações apenas deste orador e assim melhorar o reconhecimento para ele. Ambos os casos possuem suas vantagens e desvantagens. No primeiro caso a vantagem reside no fato de se obter um sistema mais generalista sendo possível ser utilizando por qualquer orador com perda na acurácia. Entretanto, para se ter o objetivo atingido de forma satisfatória é necessário um grande número de oradores diferentes, com idade e gênero diversificados. No caso das adaptações, a grande vantagem se dá pela necessidade de um número muito inferior de gravações de áudio quando comparado à primeira o que facilita, sua utilização.

No que tange à adaptação de modelos acústicos, o CMU Sphinx implementa duas técnicas: *Maximum A Posteriori* MAP e *Maximum Likelihood Linear Regression* MLLR.

A adaptação MAP é aplicada para estimar tanto as funções de densidade de gaussianas multivariáveis como o vetor de parâmetros do HMM, sendo assim possível usá-la para modelos dependentes ou independentes de locutor. Para se obter um modelo dependente de locutor, o método toma como entrada o modelo independente e características extraídas de novas gravações do orador. Este é baseado na teoria Bayesiana e sua principal característica está no fato de tomar vantagem da informação a priori gerada no processo de treinamento e assim reduzir a quantidade de dados necessários para se obter um bom resultado para o modelo acústico adaptado. Ela pode ser vista como uma variação do procedimento de treinamento dos modelos acústicos, pois segue o mesmo princípio de maximização a posteriori apresentada na etapa de treinamento, descrita na Seção 2.4.4, onde, na Equação 2.8, o termo $P(W)$ seriam as FDPs dos modelos obtidos no treinamento e $X$ o vetor de características extraídos das gravações de um locutor específico.

Já a adaptação MLLR, faz uso de regressão linear para estimar a melhor linear (ou não linear) relação entre dois conjuntos de dados, consistindo em encontrar os melhores valores $A$ e $b$ da Equação 2.10.

$$y = Ax + b \tag{2.10}$$

Em termos de reconhecimento de voz este método parametriza as médias das GMMs dos modelos acústicos segundo a Equação 2.11.

$$N(x; \mu, \Sigma) => N(x; A\mu + b, \Sigma) \tag{2.11}$$

O efeito desta transformação é ajustar as médias do modelo inicial de modo que o estado do HMM gere o modelo adaptado com maior probabilidade. Com efeto, MLLR encontra a transformação que adiciona maior densidade de probabilidade nos

modelos adaptado. Entretanto, quando a quantidade de dado é muito pequena este pode não representar corretamente o modelo objetivado, ocasionando um *overtraining* do estimador *Maximum Likelihood*. Para evitar este problema é interessante a combinação deste com algum estimador Bayesiano, como o MAP.

# Capítulo 3

# A Aplicação Proposta

## 3.1 Sistema de Reconhecimento de Fala

A primeira etapa no processo de construção do sistema foi embarcar a ferramenta de reconhecimento de fala. Com esse intuito foram analisadas algumas alternativas de possíveis *frameworks* que implementam este método baseado em HMM.

A primeira delas foi o HTK [3]. Esta é uma ferramenta que foi inicialmente desenvolvida no Laboratório Inteligência Computacional (*Machine Intelligence Laboratory*) do Departamento de Engenharia da Universidade de Cambridge e consiste em um conjunto de bibliotecas e programas disponíveis na linguagem C.

A ferramenta provê componentes que servem para o proposito de implementação de todas as etapas de um sistema de reconhecimento de fala, desde a análise do áudio de entrada até a análise dos resultados finais do reconhecimento.

O HTK é uma das ferramentas mais usadas para fins de pesquisa, não somente em reconhecimento de fala como também em algumas outras aplicações onde é necessária a construção e manipulação de HMMs, como síntese de fala, sequenciamento de DNA, entre outras aplicações de reconhecimento, como reconhecimento de caracteres e de gestos.

Além de oferecer suporte a diversas funcionalidades para implementação de sistemas de reconhecimento de fala, o HTK possui vasta documentação com diversos exemplos, o que facilita o seu uso fazendo com seja bastante difundida. No entanto, apesar de possuir código aberto, a *Microsoft* detém o *Copyright* do código original, sendo assim, as aplicações desenvolvidas com o auxílio desta não podem ser comercializadas livremente, logo, esta não se habilita a compor um projeto como o que aqui se apresenta.

Em seguida o *Sphinx* [4] foi analisado. Assim como o HTK, esta é uma ferramenta que oferece diversos componentes para implementar as etapas de um sistema de reconhecimento de fala, provendo flexibilidade na alteração de vários parâmetros.

Desenvolvido na Universidade de Carnegie Mellon, o Sphinx possui uma documentação bastante escassa quando comparado ao seu semelhante, apresentando apenas alguns manuais e tutoriais que são ligeiramente atualizados em sua página, consequentemente, seu uso não é tão difundido quanto o da outra. Entretanto, este utiliza Java, o que facilita o seu porte para *Android*, logo, a melhor opção.

Definida a ferramenta a ser utilizada, se iniciou o seu porte. O foco principal do Sphinx está nas aplicações voltadas para desktop onde a *Java Virtual Machine* (JVM) presente é muito mais robusta e com muito mais recursos. A JVM existente no *Android* utiliza uma versão mais compacta onde nem todo recurso de sua original está presente. Em face disso, o primeiro problema foi a migração do módulo de codificação do áudio de entrada. O *Sphinx* usa por padrão a *Java Sound API*[15] e o *Android* a *Audio Record*[16], o que demandou o desenvolvimento completo de um novo módulo para atender às especificações do projeto. Dada a natureza modular do *Sphinx*, esta tarefa não foi tão árdua quanto parece, bastou apenas a remoção do módulo de entrada e a inserção do novo módulo.

Sanado o problema da biblioteca, foi percebido que a forma utilizada para a leitura dos arquivos que compõem a base de dados também possui limitação. Esta consiste basicamente no tamanho dos arquivos a serem lidos. No *Android* não é possível fazer a leitura de qualquer arquivo diretamente na JVM maior que 1M e, no *Sphinx*, alguns dos arquivos utilizados pelos modelos acústicos são maiores que isto. A melhor forma para resolver esse problema é a fragmentação destes arquivos em arquivos menores de modo a não ultrapassar o limite, porém esta estrutura de leitura de arquivos não está desacoplada do sistema da mesma forma que o módulo de codificação. Logo, dada a necessidade de um estudo mais detalhado para esta operação, foi decidido mudar a abordagem deixando esta para projetos futuros.

O *Sphinx* faz parte de um conjunto de ferramentas desenvolvidas pelo grupo *CMUSphinx* da Universidade Crnegie Mellon. Dentre elas há o *Pocketsphinx* que é uma versão mais compacta da versão 3 do *Sphinx*, hoje na sua versão 4, voltado para dispositivos móveis e embarcados. Este está escrito em C e encontra-se na versão 0.8. Além disso, é disponibilizada uma aplicação de demostração para *Android* com exemplos de utilização de suas funcionalidades no SO, o que facilita sua adaptação para outras aplicações.

Outra vantagem que este apresenta está no fato de rodar fora da JVM, pois utiliza o pacote *Native Development Kit* NDK[17] para interpretar seu código para Java. Desta forma a limitação com o tamanho do arquivo enfrentada anteriormente deixa de existir.

A partir daí, utilizar o *Pocketsphinx* no lugar de *Sphinx* se mostrou a melhor alternativa. Este não foi selecionado logo de início porque, apesar de ser mais rápido, sua acurácia é inferior a do original.

Com *Pocketsphinx*, o desenvolvedor pode criar diversos objetos de busca em diferentes gramáticas e modelos de linguagem. Além disso, é possível mudar qual deles deve ser utilizado no decorrer da execução sem haver a necessidade de reinicialização do aplicativo, e assim, provê ao usuário uma maior experiência de interatividade.

Existem quatro possíveis modos de busca:

- **Palavra Chave** – Busca apenas a frase ou palavra definida e ignora qualquer outra fala.
- **Gramática** – Onde o reconhecimento da fala se dá de acordo com a Java Speech Grammar Framework (JSGF)[18]. Diferentemente da anterior esta não ignora palavras que não estão na gramática, mas tenta reconhecê-las.
- **Modelo de Linguagem** – Utilizada para o reconhecimento de fala contínua. Usa como base um modelo de linguagem concedido no processo de configuração do sistema.
- **Fonética** – Similar ao anterior, porém seu modelo tem por objetivo apenas reconhecer e retornar fonemas.

É possível haver uma combinação de todas elas e dependendo da necessidade escolher a mais adequada. Entretanto, como o objetivo deste trabalho consiste em desenvolver um sistema de comando e controle para navegação, foram utilizados dois modos de busca: o por palavra-chave e o modo de busca em gramática. O primeiro serve para ativação do sistema de navegação. E o segundo para a geração dos comandos em si.

Esta estrutura permite ao usuário ativar ou bloquear o sistema de acordo com sua necessidade sem interação física com o aparelho.

Por ser uma prova de conceito, este trabalho não se dispôs a desenvolver uma base de dados para reconhecimento, portanto, foi utilizanda a disponibilizada pela aplicação de demonstração. Esta consiste em um vocabulário para amplo reconhecimento semi-continuo no inglês americano, o que força o conjunto de comandos ser também em inglês. Durante os testes foi percebido um fraco desempenho desta base e, por conseguinte, esta foi adaptada com o auxílio do *SphinxTrain*, como será visto no Capítulo 4

## 3.2 Sistema de Navegação

Desde a versão 4 do seu SDK, o *Android* possui uma classe chamada *AccessibilityService*. Esta implementa um serviço que roda em *bakground* e recebe chamadas do sistema quando um evento de acessibilidade é disparado. Estes eventos denotam

mudanças de estado na tela do aparelho, tais como: cliques, mudanças de foco, *scroll* e etc. A partir da versão 16 do SO, esta classe passou a poder executar requisições a componentes da interface da aplicação que está visível no momento. Sendo assim, a partir dela, é possível executar cliques, *Scroll*s e mudanças de foco programaticamente.

Em posse deste recurso, buscou-se, inicialmente, seguir o mesmo comportamento do D-Pad, onde o usuário ao apertar algum botão direcional muda o foco para o elemento seguinte mais próximo na direção escolhida. Nesta linha, o sistema de reconhecimento da fala representaria o comandante (D-Pad) que emitiria o comando e o executor, a *AccessibilityService*, executaria a ação. Entretanto, no decorrer do desenvolvimento, foi percebida uma deficiência nesta abordagem. Existem dois tipos de foco, o *AccessibilityFocus* e o *InputFocus*. Todo elemento da interface é passível de receber *AccessibilityFocus*, este comportamento já é pré-definido no SO, porém, somente aqueles elementos que o desenvolvedor da aplicação define como focáveis, *focusable*, podem receber *InputFocus*, o qual é utilizado na navegação com foco. Logo, se o desenvolvedor não atribuí-lo a todos os componentes com significado não será possível navegar pela tela.

Tentando evitar esta dependência, foi decido utilizar um sistema de *Badges*, mostrado na Figura 3.1, onde os elementos focáveis e/ou clicáveis passam a receber um rótulo numerado no canto superior esquerdo e, caso o usuário queira clicar em algum componente, basta dizer o número associado a ele e o sistema executará o clique automaticamente. Estes rótulos são dispostos em uma camada que também é sensível a toque e se sobrepõe a todas as telas do dispositivo *(Overlay)* de modo a não impossibilitar a utilização do aparelho por usuários capazes de manipulação física.

Para a transição entre as telas através de *Scroll/Swipe*, foram implementados dois comandos: *GO FORWARD* e *GO BACKWARD*. O primeiro executa o movimento de *Scroll* para direita ou para baixo, dependendo da orientação do componente. E o segundo, para esquerda ou para cima. Ao falar um destes comandos é feito na tela uma busca pelos componentes que podem ser *"rolados"*. Se houver mais de um, a estes são atribuídos *Badges* numerados, assim como na ação de clique, e, ao falar o número, a ação é executada. Caso haja apenas um elemento, a ação ocorre sem mais interações.

Toda mudança de estado é reportada à *AccessibilityService* que atualiza a lista de elementos clicáveis e seu respectivos *Badges*, quando necessário. Esta operação é feita com base na natureza arquitetural dos elementos da tela. Estes estão moldados em forma de árvore, onde todo elemento é um nó, seu pai é aquele que o engloba (região na tela) e seus filhos, aqueles englobados por ele. Logo, é possível chegar a qualquer componente (nó) a partir de um anteriormente conhecido. Quando se


Figura 3.1: Sistema de *Badges*

entra numa tela, esta geralmente possui um elemento já com foco (*InputFocus*) e a *AccessibilityService* possui esta referencia e, a partir dele, é criado o mapa com os elementos clicáveis/focáveis e seus *Badges*. Há casos em que não há nenhum item com foco, provavelmente o desenvolvedor não definiu quais elementos são passíveis de receber este atributo, então é utilizado o nó raiz (maior área externa da tela), que é recuperado a partir da *AccessibilityService*, para o inicio da busca.

Esta abordagem se mostrou muito mais interativa e rápida que a anterior, pois, antes, para executar o clique no elemento desejado o usuário teria que navegar até este repetindo quantos comandos direcionais fossem necessários e então falar o comando de clique. E agora, basta falar o valor de seu *Badge* que a ação de clique é executada.

## 3.3 Utilidades

Apesar da maioria dos recursos do aparelho serem acessíveis através de clique e *Scroll*, ainda há comandos cujo acionamento vem do *hardware*. Todos esses comportamentos podem ser replicáveis através da *AccessibilityService*. Logo, para cada um deles foi criado um comando no reconhecedor de fala e uma ação associada como pode servisto na Tabela 3.1.

| **Comando** | **Ação** |
|---|---|
| LOUDER | Aumentar Volume |
| QUIETER | Diminuir Volume |
| GO BACK | Voltar |
| GO HOME | Ir para Tela Principal |

Tabela 3.1: Comandos Auxiliares

Além disso, seguindo o quarto dos Sete Princípios do Design Universal, Seção 2.1, foi desenvolvido um módulo que informa ao usuário o último comando reconhecido. Este tem duas formas de apresentação que podem ser ativadas ou não dependendo da necessidade ou desejo do usuário. A primeira, consiste em expor o comando na barra de notificação do sistema e atualizá-lo a cada nova requisição. Já a segunda, é uma ferramenta de *Text to Speech* (TTS) responsável por falar aquilo que foi requisitado. Para esta última fez-se uso da ferramenta do *Android* para síntese de voz que vem por padrão em todos os dispositivos e funciona sem a presença da internet [19].

De modo a não interferir no funcionamento dos demais módulos este é executado em uma *Thread* a parte segundo o *Design Patter Observer* [20] e uma fila *First In First Out* (FIFO) [21] que ordena a entrada e saída das mensagens.

Apesar de não pertencer ao escopo deste trabalho, também foi desenvolvido um módulo de escrita. Este faz uso da *Google Voice API* para geração dos textos e da área de transferência do dispositivo para inserção daquilo que foi compreendido no campo cabível. Este modo de inserção foi resultado de um teste para avaliar as etapas necessárias para execução da tarefa. A forma adequada de fazê-lo é criar um *Input Method Editor* (IME) [22] a ser ativado quando um campo de entrada de texto receber foco, *InputFocus*, mas isto ficou para trabalhos futuros.

# Capítulo 4

# Testes, Melhorias e Resultados

Com intuito de validar os passos e decisões tomadas no decorrer do desenvolvimento deste trabalho foram executadas duas séries de testes e uma de adaptação. Entretanto, estes foram focados na análise de desempenho (velocidade de processamento dos comandos) do sistema de reconhecimento de fala, pois, como não há nenhuma aplicaçãos similar à apresentada, nenhum teste comparativo pode ser executado. Além disso, testes empíricos com base na usabilidade necessitam da presença dos usuários, mas o resultado atingido não foi suficiente, dado que o modelo acústico é dependente de locutor e nenhum módulo de treinamento/adaptação foi desenvolvido.

O *Pocketsphinx* possui uma série de parâmetros em seu processo de decodificação que podem ser variados a fim de melhorar o sistema de reconhecimento mantendo o equilíbrio entre custo computacional, velocidade e acurácia. Infelizmente, a documentação da ferramenta não descreve o que todos os parâmetros fazem. Portanto, os testes aqui apresentados foram executados apenas com aqueles cujo entendimento foi pleno. São eles: *-ds*, *-topn*, *-maxhmmpf*, *-maxwpf*, *-pl_window*. Suas descrições e comportamentos estão apresentados na Seção 4.3.

Neste capítulo serão vistos os procedimentos tomados para as tarefas já citadas, desde a preparação do ambiente até a exposição dos resultados, assim como a análise dos dados obtidos.

## 4.1 Preparação do Ambiente e Dados de Entrada

Para recriar o mais próximo quanto possível o ambiente em que a ferramenta será utilizada foram gravados, para cada comando, 50 arquivos de áudio contendo apenas uma repetição do mesmo. Estes arquivos foram criados em um celular Galaxy S4 onde o orador fez uso do microfone original do aparelho e suportado pela aplicação *Easy Voice Recorder* [23], esta última dá o poder de configurar o arquivo a ser gerado. Em posse disso, as gravação foram criadas no formato WAV com 16KHz 16-bit PCM

mono, semelhantes às configurações dos modelos acústicos utilizados. Estes arquivos foram agrupados em pastas de modo que cada pasta contenha as gravações de um único comando. Como há 25 comandos, criou-se 25 pastas contendo 50 arquivos de áudio cada. Cada pasta recebeu como rótulo/nome o comando associado a ela de modo a auxiliar a identificação dos seus arquivos.

Em seguida, foi desenvolvida uma aplicação *Android* responsável por executar os testes e contabilizar seus resultados. Esta aplicação recebe como entrada uma pasta contendo um arquivo no formato JSON, que possui uma lista de pares formados pelo parâmetro a ser analisada e seu respectivo valor (Apêndice B), além dos arquivos anteriormente agrupados.

A aplicação de demostração do *Pocketsphinx* vem configurada para ter como entrada de áudio a ser processado a captação oriunda do próprio microfone do aparelho. Portanto, para a execução dos testes, esta aplicação teve que ser alterada para receber arquivos no formato WAV como entrada.

É importante ressaltar que o *Pocketsphinx* processa os dados do áudio grupados em blocos (array) de *SHORT* e a leitura de arquivos no Java é feita em blocos (array) de *BYTE* o que implicou na conversão de *BYTE* para *SHORT*. Esta conversão, se não feita da forma adequada, pode impactar no desempenho da aplicação dada a formatação dos dados de entrada. Este processo foi feito segundo o algorítimo abaixo:

```
for(int i = 0; i < b.length/2; i++) {
    s[i] = (short) ((b[i*2] & 0xff)|(b[i*2 + 1] << 8));
}
```

Onde *s* é o array de *SHORT* e *b* array de *BYTE*.

Durante a execução dos testes a ferramenta de reconhecimento foi reinicializada para cada par parâmetro:valor e, a cada ciclo, as pastas foram processadas uma única vez, assim com todos os seus respectivos arquivos. Estes últimos foram analisados de forma independente e iterativa, isto é, um após o outro sem sobreposição ou paralelismo. Seguindo estas premissas, foram contabilizados os tempos de processamento e o resultado do reconhecimento, correto ou errado, para cada arquivo em um Galaxy Tab 7.0 Plus com Android 4.4 (SDK 16).

## 4.2 Processo de Coleta e Exposição dos Resultados

Para análise dos dados obtidos, foi desenvolvida uma aplicação Web em NodeJS [24] + MongoDB [25] que organiza os dados armazenados no JSON, saído da aplicação de teste, e os coloca no banco de dados com o intuito do manipulá-los de

forma mais simples. Esta aplicação é responsável por gerar os gráficos expositores dos dados estatísticos de acertos e erros.

Há dois tipos de gráficos. O primeiro, em formato de barras verticais, expõe a taxa de Acertos e Erros onde o eixo Y representa o percentual obtido para todos os comandos e o eixo X o valor assumido pelo parâmetro analisado. Já o segundo, formato em linhas, o tempo médio de processamento dos comandos, tanto dos acertos quanto dos erros, de modo que o eixo Y contém o tempo em milissegundos e o eixo X os valores dos parâmetros. Estes valores são obtidos segundo as Equações 4.1 e 4.2 para o primeiro tipo e a Equação 4.3 para o segundo.

$$TxA = \frac{\#acerto}{\#acerto + \#erro} \tag{4.1}$$

$$TxE = \frac{\#erro}{\#acerto + \#erro} \tag{4.2}$$

Onde TxA é a taxa dos comandos reconhecidos corretamente, TxE é a taxa dos comandos reconhecidos incorretamente, #acertos é o número de acertos registrados para um único valor assumido pelo parâmetro em observação e #erro o número de erros contabilizados no mesmo contexto do anterior.

$$\bar{t} = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{4.3}$$

Onde o $n$ é o numero de arquivos processados para um único valor assumido pelo parâmetro em observação e $t$ o tempo de processamento de cada arquivo no mesmo contexto.

Outra tarefa exercida por este último programa é executar o cálculo do desvio padrão de acordo com a Equação 4.4 para cada valor assumido pelo parâmetro em observação.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{4.4}$$

Como dito anteriormente, houve duas levas de testes. Isso se deu pois os resultados obtidos na primeira leva ficaram abaixo do esperado, posto que, em um sistema de reconhecimento de voz com um conjunto pequeno de palavras/comandos a serem reconhecidos sua taxa de acerto costuma ficar acima de 90%, entretanto, esta se apresentou com taxa na casa dos 60%(Seção 4.3). Para resolver este problema havia duas alternativas: criar um modelo acústico para a aplicação ou adaptar o existente.

Caso queira ser criado um modelo independente de locutor é necessário coletas de áudios de diferentes grupos de oradores em estúdio de alta qualidade para daí

preprocessá-los gerando o novo modelo, o que não compõe nenhum dos objetivos deste projeto.

O segundo caso, por sua vez, se mostrou como a melhor alternativa, pois este gera bons resultados mesmo com dados esparsados providos pelo usuário, onde, mesmo que não sejam providos dados suficientes, no pior dos casos, o desempenho será o mesmo que o do modelo original.

Como este trabalho se propõe a apresentar uma prova de conceito, não foi desenvolvido nenhum módulo de adaptação dos modelos acústicos na aplicação, portanto, foi utilizado o *SphixTrain* para executar o processo de adaptação e gerar os novos modelos a serem usados na aplicação. A tarefa de desenvolvimento deste módulo ficou destinada a trabalhos futuros.

Em posse das gravações dos comandos utilizadas na etapa de testes, estes foram reagrupados na escala 6:4, onde 60% deles (30 arquivos por comando) foram usados para adaptar os modelos e 40% (20 arquivos por comando) foram utilizados nos testes dos modelos adaptados.

Seguindo as orientações da documentação do *SphinxTrain*, os modelos passaram pelos dois processos de adaptação, MLLR e MAP, respectivamente. Os passos seguidos estão descritos em Apêndice C. Estes foram feitos em um computador DELL Vostro com 4G de RAM e processador i5.

Em seguida, este novo modelo foi testado segundo descrito no início deste capítulo, porém utilizando os arquivos destinados para testes do modelo adaptado.

## 4.3 Resultados dos Testes

Como já ressaltado, o *Pockesphix* tem em seu processo de decodificação um conjunto de parâmetros independentes que podem ser variados de modo a garantir um melhor desempenho. Entretanto, a computação de todos estes durante a execução da aplicação pode ser custoso em demasia e, consequentemente, afetar a usabilidade da ferramenta.

Para evitar este problema, é requerida a utilização de técnicas que permitem filtrar os estados e as componentes das densidades mais relevantes para cada *frame*, em suma, uma forma de prever quais são aqueles com maior significância. Com efeito, a ferramenta se baseia em três princípio:

1. O conjunto de estados e densidades mais verossímeis muda relativamente pouco de *frame* para *frame*.

2. Se o estado $N$ de um dado HMM não for verossímil então todos os estados $> N$ também não serão nos próximos *frames*.

3. Se o modelo simplificado de um estado não for verossímil então este estado não o será.

Além disso, cada fala é processada de forma síncrona, um *frame* por vez. E a cada *frame* o decodificador possui um número de HMMs ativos que serão utilizados na comparação com o próximo *frame*. Porém, antes disso, o decodificador descarta ou desativa aqueles cujo estado possui verossimilhança abaixo de um determinado parâmetro. Este parâmetro é obtido pelo produto entre a verossimilhança do melhor estado por um valor fixo pré-definido, *threshold*, que varia entre 0 e 1. Esta técnica é chamada de poda (*Pruning*) e pode ser aplicada em outros níveis. Como por exemplo, para determinar o número de palavras a serem reconhecidas ou a densidade em um GMM que melhor representa um vetor de parâmetros.

### 4.3.1 Análise do Parâmetro -maxhmmpf *(Max HMM per Frame)*

Mesmo com a técnica de poda, há momentos em que o número de HMMs que se mantiveram ativos continua muito grande, o que pode ocasionar uma redução da chance de encontrar a palavra correta, além de configurar um custo computacional mais alto. Em situações como essas é possível utilizar a variável **maxhmmpf**. Esta tem por objetivo limitar o número máximo de HMMs a serem computados em cada *frame*.

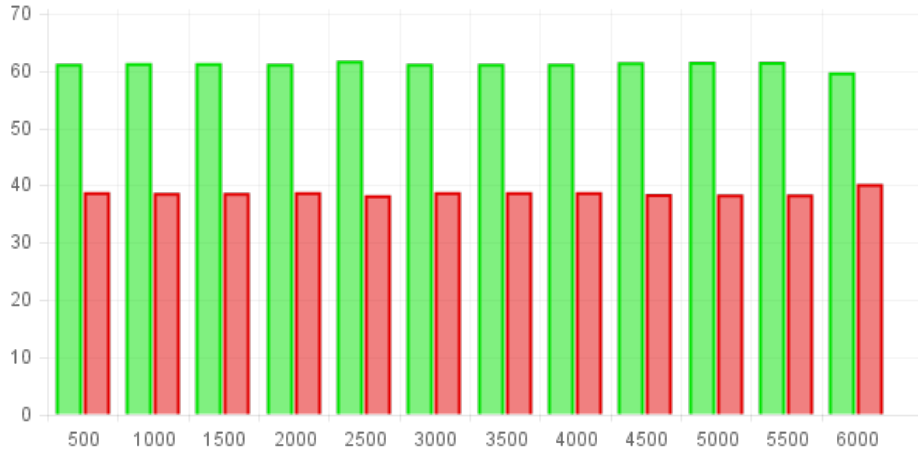


Figura 4.1: Resultados da TxA e TxE para o parâmetro **-maxhmmpf** antes da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

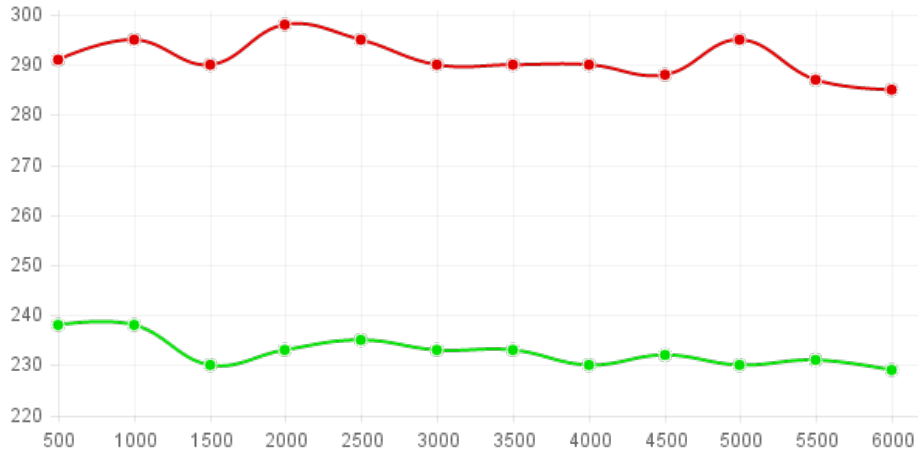


Figura 4.2: Tempo médio de processamento de cada comando para o parâmetro **-maxhmmpf** antes da adaptação

**Eixo Vertical:** Valores correspondentes à $\bar{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.1: Valor médio e desvio padrão do parâmetro **-maxhmmpf** antes da adaptação

| | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 | 4500 | 5000 | 5500 | 6000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 61.28 | 61.28 | 61.04 | 60.96 | 61.28 | 60.16 | 60.56 | 61.04 | 61.2 | 61.12 | 60.88 | 61.75 |
| $\sigma_{TxA}$ | 13.55 | 13.49 | 13.58 | 13.64 | 13.54 | 13.69 | 13.71 | 13.34 | 13.54 | 13.49 | 13.76 | 13.78 |
| $TxE$ | 37.6 | 37.68 | 37.84 | 38.08 | 37.76 | 38.8 | 38.4 | 37.76 | 37.52 | 37.68 | 38.08 | 37.16 |
| $\sigma_{TxE}$ | 13.41 | 14.2 | 14.14 | 14.12 | 14.17 | 14.29 | 14.29 | 13.86 | 14.3 | 14.09 | 14.46 | 14.25 |
| $\bar{t}$ | 238 | 238 | 230 | 233 | 235 | 233 | 233 | 230 | 232 | 230 | 231 | 229 |
| $\sigma_{\bar{t}}$ | 0.09 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 |
| $\overline{te}$ | 291 | 295 | 290 | 298 | 295 | 290 | 290 | 290 | 288 | 295 | 287 | 285 |
| $\sigma_{\overline{te}}$ | 0.10 | 0.10 | 0.10 | 0.10 | 0.11 | 0.10 | 0.10 | 0.10 | 0.10 | 0.10 | 0.09 | 0.10 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxAx} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\bar{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\bar{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

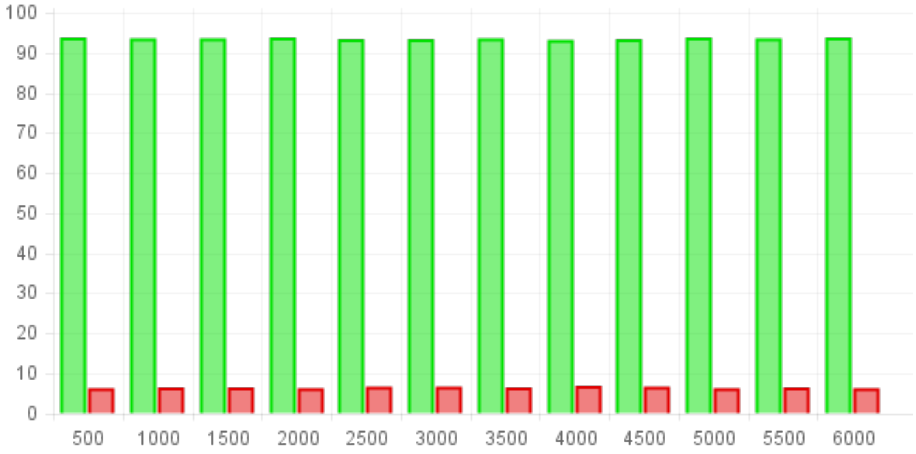


Figura 4.3: Resultados da TxA e TxE para o parâmetro **-maxhmmpf** depois da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

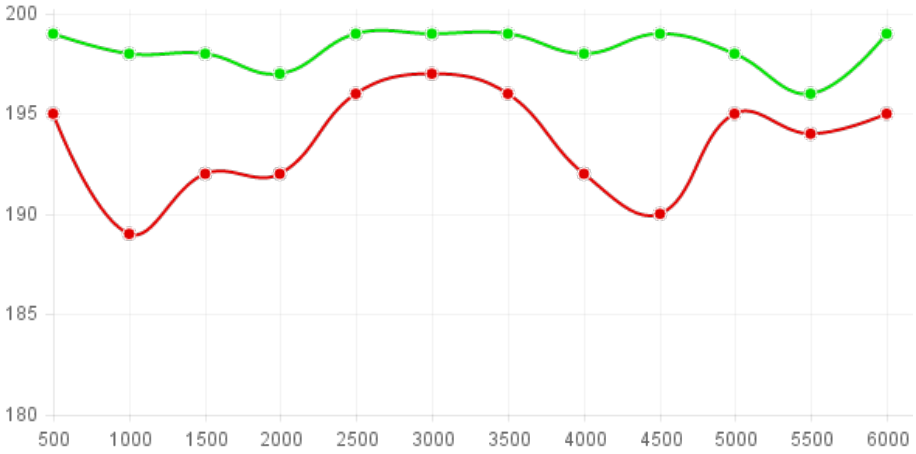


Figura 4.4: Tempo médio de processamento de cada comando para o parâmetro **-maxhmmpf** depois da adaptação

**Eixo Vertical:** Valores correspondentes à $\overline{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

É possível notar que, tanto antes quanto depois da adaptação, não houve ganho nem perda significativa com a variação do parâmetro. Este comportamento se dá pelo fato da aplicação possuir apenas comandos curtos, logo, um conjunto pequeno de HMMs são capazes de representá-los satisfatoriamente.

Como não há ganhos nem perdas com a variação deste parâmetro, é possível utilizar o seu valor padrão, 3000.

Tabela 4.2: Valor médio e desvio padrão do parâmetro **-maxhmmpf** depois da adaptação

| | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 | 4500 | 5000 | 5500 | 6000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 93.8 | 93.6 | 93.6 | 93.8 | 93.4 | 93.4 | 93.6 | 93.2 | 93.4 | 93.8 | 93.6 | 93.75 |
| $\sigma_{TxA}$ | 1.83 | 1.93 | 4.77 | 1.87 | 2 | 1.84 | 4.77 | 1.97 | 1.76 | 1.74 | 1.84 | 4.7 |
| $TxE$ | 6.2 | 6.4 | 6.4 | 6.2 | 6.6 | 6.6 | 6.4 | 6.8 | 6.6 | 6.2 | 6.4 | 6.25 |
| $\sigma_{TxE}$ | 1.83 | 1.93 | 1.99 | 1.87 | 2.01 | 1.84 | 2 | 1.98 | 1.77 | 1.75 | 1.84 | 1.84 |
| $\overline{t}$ | 199 | 198 | 198 | 197 | 199 | 199 | 199 | 198 | 199 | 198 | 196 | 199 |
| $\sigma_{\overline{t}}$ | 0.06 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.06 |
| $\overline{te}$ | 195 | 189 | 192 | 192 | 196 | 197 | 196 | 192 | 190 | 195 | 194 | 195 |
| $\sigma_{\overline{te}}$ | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.04 | 0.05 | 0.05 | 0.04 | 0.05 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\overline{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\overline{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

### 4.3.2 Análise do Parâmetro -maxwpf *(Max Words per Frame)*

Assim como a **-maxhmmpf**, a **-maxwpf** é uma técnica que tem seu foco na poda, onde este limita o número máximo de palavras esperadas para cada *frame*. A documentação sugere valores em torno de 40, porém, para esta aplicação, não há comandos com mais de 2 palavras. Portanto, para analisar o comportamento deste parâmetro, a ele foram atribuídos valores de 1 a 10.

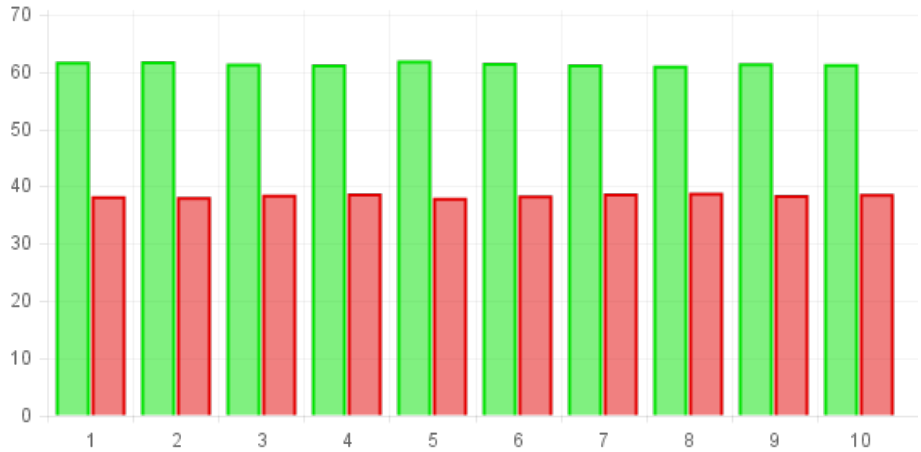


Figura 4.5: Resultados da TxA e TxE para o parâmetro **-maxwpf** antes da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

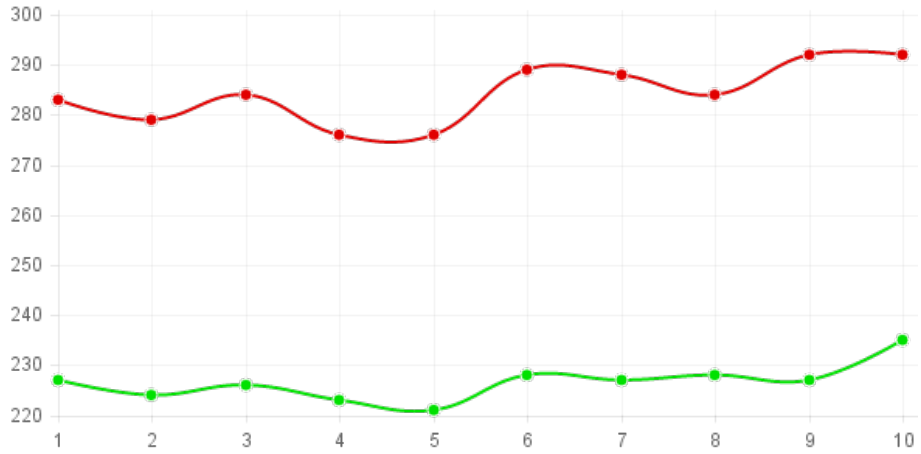


Figura 4.6: Tempo médio de processamento de cada comando para o parâmetro **-maxwpf** antes da adaptação

**Eixo Vertical:** Valores correspondentes à $\bar{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.3: Valor médio e desvio padrão do parâmetro **-maxwpf** antes da adaptação

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 61.36 | 61.44 | 60.96 | 61.04 | 60.64 | 61.52 | 61.28 | 61.04 | 61.76 | 61.36 |
| $\sigma_{TxA}$ | 13.72 | 13.55 | 13.53 | 13.74 | 13.65 | 13.71 | 13.42 | 13.53 | 13.55 | 13.42 |
| $TxE$ | 37.52 | 37.52 | 38 | 37.84 | 38.24 | 37.52 | 37.68 | 38.16 | 37.12 | 37.6 |
| $\sigma_{TxE}$ | 14.09 | 14.07 | 14.03 | 14.4 | 14.25 | 14.38 | 13.77 | 14.33 | 14.02 | 13.93 |
| $\bar{t}$ | 227 | 224 | 226 | 223 | 221 | 228 | 227 | 228 | 227 | 235 |
| $\sigma_{\bar{t}}$ | 0.08 | 0.07 | 0.07 | 0.07 | 0.07 | 0.08 | 0.07 | 0.08 | 0.08 | 0.08 |
| $\overline{te}$ | 283 | 279 | 284 | 276 | 276 | 289 | 288 | 284 | 292 | 292 |
| $\sigma_{\overline{te}}$ | 0.10 | 0.09 | 0.10 | 0.09 | 0.09 | 0.11 | 0.10 | 0.10 | 0.10 | 0.10 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\bar{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\bar{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

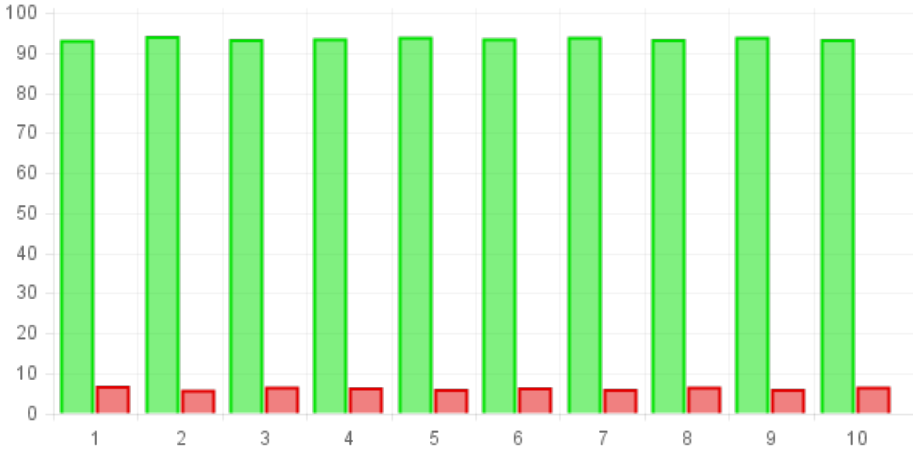


Figura 4.7: Resultados da TxA e TxE para o parâmetro **-maxwpf** depois da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

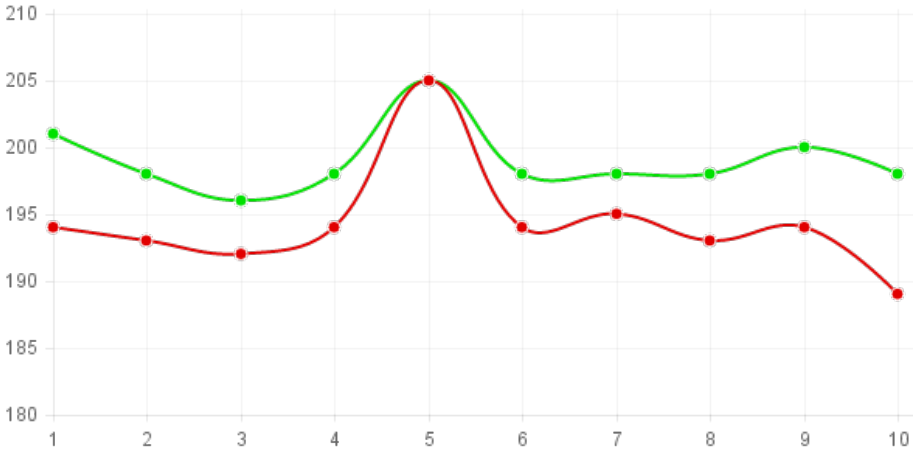


Figura 4.8: Tempo médio de processamento de cada comando para o parâmetro **-maxwpf** depois da adaptação

**Eixo Vertical:** Valores correspondentes à $\overline{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.4: Valor médio e desvio padrão do parâmetro **-maxwpf** depois da adaptação

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 93.2 | 94.2 | 93.4 | 93.6 | 94 | 93.6 | 94 | 93.4 | 94 | 93.4 |
| $\sigma_{TxA}$ | 1.98 | 1.81 | 1.84 | 1.95 | 4.79 | 1.95 | 4.72 | 1.97 | 1.79 | 1.86 |
| $TxE$ | 6.8 | 5.8 | 6.6 | 6.4 | 6 | 6.4 | 6 | 6.6 | 6 | 6.6 |
| $\sigma_{TxE}$ | 1.98 | 1.81 | 1.84 | 1.95 | 1.99 | 1.95 | 1.82 | 1.98 | 1.79 | 1.87 |
| $\overline{t}$ | 201 | 198 | 196 | 198 | 205 | 198 | 198 | 198 | 200 | 198 |
| $\sigma_{\overline{t}}$ | 0.06 | 0.05 | 0.05 | 0.05 | 0.06 | 0.05 | 0.05 | 0.05 | 0.06 | 0.06 |
| $\overline{te}$ | 194 | 193 | 192 | 194 | 205 | 194 | 195 | 193 | 194 | 189 |
| $\sigma_{\overline{te}}$ | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.04 | 0.05 | 0.05 | 0.04 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\overline{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\overline{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

Assim como o antecessor, sua variação apresentou pouca alteração. Como o conjunto de comandos desta aplicação possui no máximo 2 palavras, não faz sentido utilizar valores superiores a este, dado que não há nenhum impacto na acurácia ou na velocidade de processamento.

### 4.3.3 Análise do Parâmetro -topn

Diferentemente das anteriores, este técnica de poda restringe ao sistema a computação das $n$ primeiras densidades de uma dada GMM com base em seus pontos, como já discutido.

A documentação define como padrão o valor 4, mas sugere o valor 2 com ressalvas, pois pode gerar queda na acurácia.

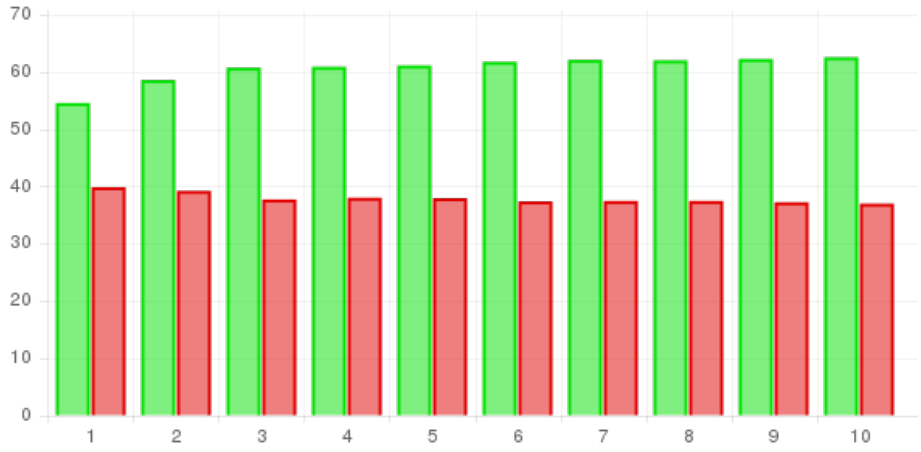


Figura 4.9: Resultados da TxA e TxE para o parâmetro **-topjn** antes da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

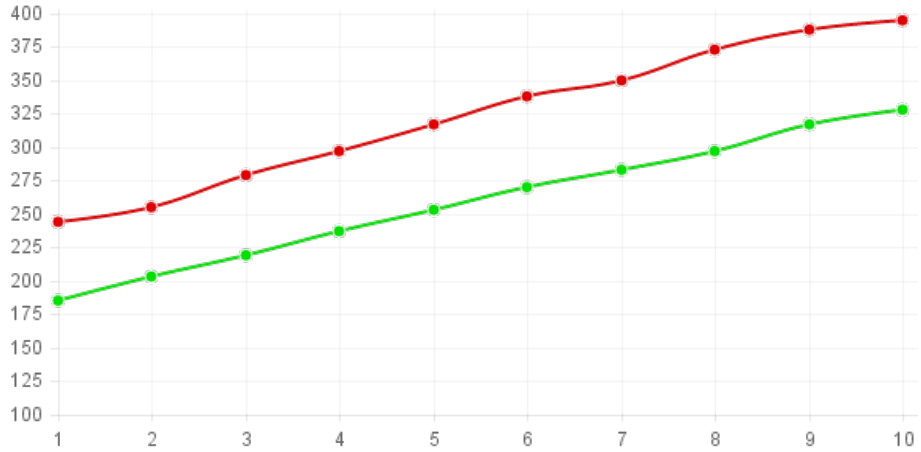


Figura 4.10: Tempo médio de processamento de cada comando para o parâmetro **-topn** antes da adaptação

**Eixo Vertical:** Valores correspondentes à $\bar{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.5: Valor médio e desvio padrão do parâmetro **-topn** antes da adaptação

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 54.56 | 58.56 | 60.72 | 60.88 | 61.12 | 61.76 | 62.08 | 62 | 62.24 | 62.56 |
| $\sigma_{TxA}$ | 15.52 | 14.67 | 14.25 | 13.72 | 13.77 | 13.92 | 13.81 | 13.66 | 13.6 | 13.4 |
| $TxE$ | 39.84 | 39.2 | 37.68 | 38 | 37.92 | 37.36 | 37.44 | 37.44 | 37.2 | 36.96 |
| $\sigma_{TxE}$ | 15.81 | 15.11 | 14.53 | 13.96 | 13.95 | 13.98 | 14.01 | 13.87 | 13.9 | 13.72 |
| $\overline{t}$ | 185 | 203 | 219 | 237 | 253 | 270 | 283 | 297 | 317 | 328 |
| $\sigma_{\overline{t}}$ | 0.07 | 0.08 | 0.08 | 0.08 | 0.09 | 0.09 | 0.09 | 0.09 | 0.10 | 0.10 |
| $\overline{te}$ | 244 | 255 | 279 | 297 | 317 | 338 | 350 | 373 | 388 | 395 |
| $\sigma_{\overline{te}}$ | 0.09 | 0.09 | 0.10 | 0.11 | 0.11 | 0.12 | 0.12 | 0.13 | 0.13 | 0.13 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\overline{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\overline{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

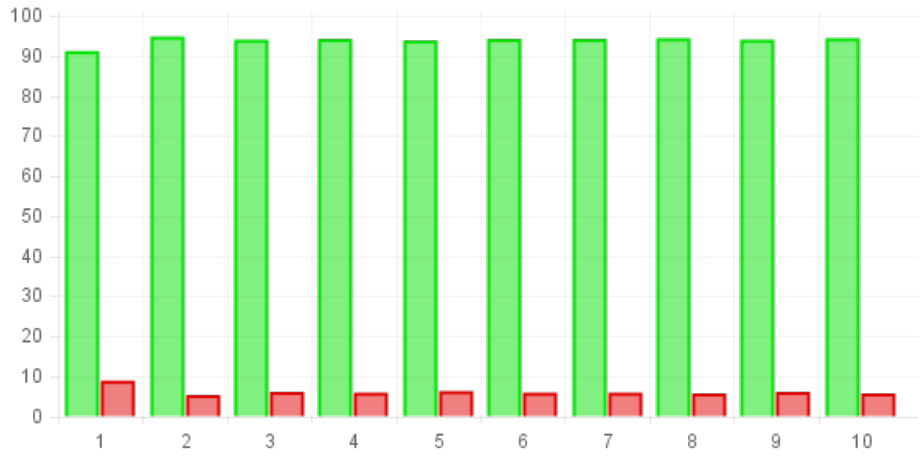


Figura 4.11: Resultados da TxA e TxE para o parâmetro **-topn** depois da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

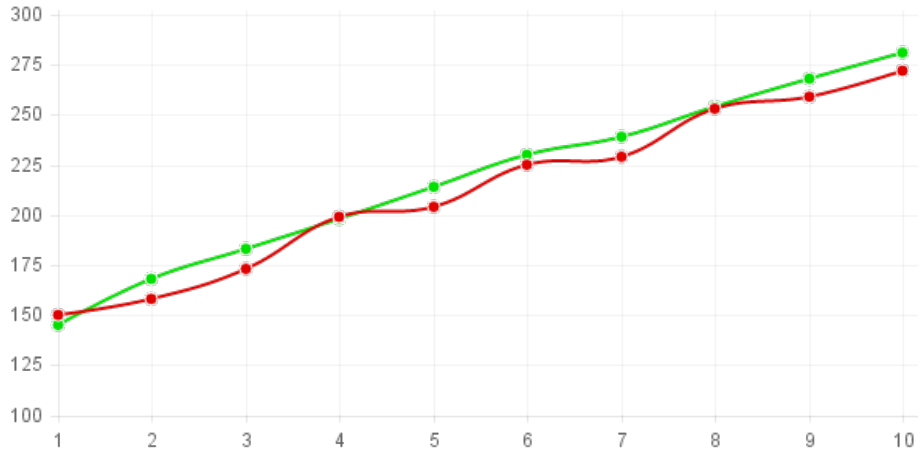


Figura 4.12: Tempo médio de processamento de cada comando para o parâmetro **-topn** depois da adaptação

**Eixo Vertical:** Valores correspondentes à $\bar{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.6: Valor médio e desvio padrão do parâmetro **-topn** depois da adaptação

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 91.2 | 94.8 | 94 | 94.2 | 93.8 | 94.2 | 94.2 | 94.4 | 94 | 94.4 |
| $\sigma_{TxA}$ | 2.38 | 4.78 | 1.83 | 4.87 | 1.93 | 1.96 | 1.96 | 4.87 | 2 | 1.84 |
| $TxE$ | 8.8 | 5.2 | 6 | 5.8 | 6.2 | 5.8 | 5.8 | 5.6 | 6 | 5.6 |
| $\sigma_{TxE}$ | 2.37 | 1.79 | 1.83 | 2.03 | 1.94 | 1.97 | 1.97 | 2.04 | 2.01 | 1.85 |
| $\bar{t}$ | 145 | 168 | 183 | 198 | 214 | 230 | 239 | 254 | 268 | 281 |
| $\sigma_{\bar{t}}$ | 0.04 | 0.05 | 0.05 | 0.05 | 0.06 | 0.07 | 0.06 | 0.07 | 0.07 | 0.07 |
| $\overline{te}$ | 150 | 158 | 173 | 199 | 204 | 225 | 229 | 253 | 259 | 272 |
| $\sigma_{\overline{te}}$ | 0.04 | 0.04 | 0.04 | 0.05 | 0.05 | 0.07 | 0.06 | 0.07 | 0.08 | 0.08 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\bar{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\bar{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

Apesar da preocupação apresentada com a possível perda da acurácia, isso não se confirma. É possível perceber que mesmo variando o valor para 1 o sistema manteve a acurácia apresentada nas demais, tanto antes quanto depois da adaptação, e apresentado ganhos significativos em tempo de processamento. Logo, este foi considerado como ótimo.

### 4.3.4 Análise do Parâmetro -ds *(FrameDownsampling)*

Esta técnica é considerada, segundo a documentação, como a arma nuclear dos processos de poda e deve ser utilizada com cautela. Em um primeiro olhar pode indicar que seu uso não funciona, mas segundo os teste se mostra bem eficaz. Esta consiste em computar somente as gaussianas a cada $N$ *frames* e ignorar os demais, assim, ganhar o tempo que seria gasto com estes.

Ela apresenta resultados favoráveis onde o discurso a ser reconhecido é longo e expressado com cuidado, desta forma, as características mudam pouco de *frame* para *frame*. Este comportamento permite que o sistema ignore alguns *frames* sem grandes perdas na acurácia e com ganhos significativos em velocidade.

A documentação sugere valores não superiores a 3 onde a melhor alternativa se encontra onde este vale 2.

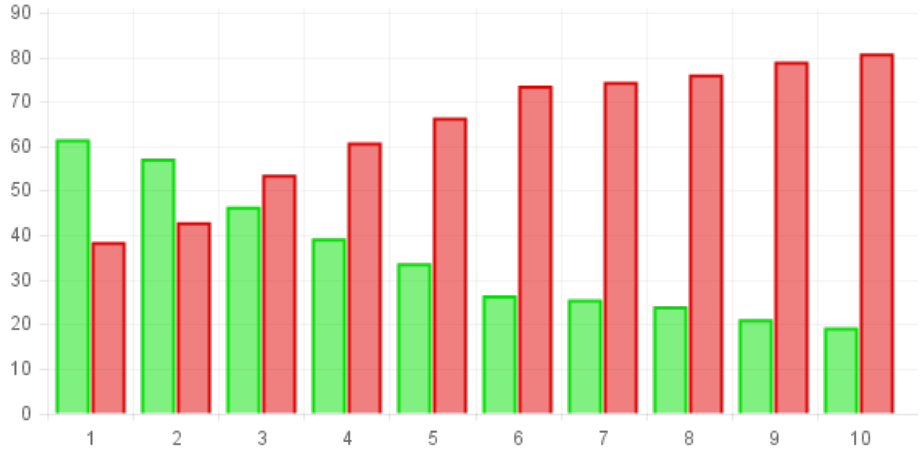


Figura 4.13: Resultados da TxA e TxE para o parâmetro **-ds** antes da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

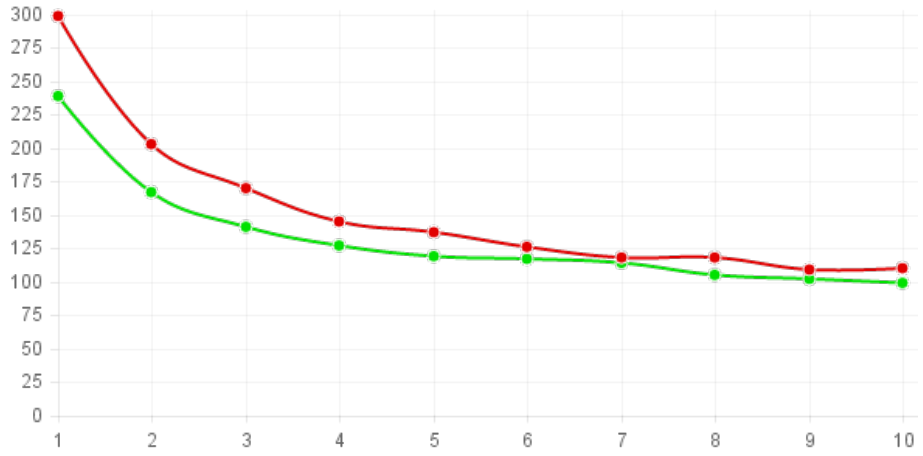


Figura 4.14: Tempo médio de processamento de cada comando para o parâmetro **-ds** antes da adaptação

**Eixo Vertical:** Valores correspondentes à $\bar{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.7: Valor médio e desvio padrão do parâmetro **-ds** antes da adaptação

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 61.28 | 56.08 | 42.88 | 34.8 | 25.84 | 20.72 | 17.36 | 14.96 | 14.16 | 13.12 |
| $\sigma_{TxA}$ | 13.48 | 13.36 | 16.45 | 18.82 | 22.04 | 23.25 | 24.05 | 24.54 | 24.81 | 25.4 |
| $TxE$ | 37.52 | 42.8 | 53.36 | 60.64 | 68.8 | 75.68 | 77.52 | 77.68 | 80.96 | 81.04 |
| $\sigma_{TxE}$ | 14.14 | 15.18 | 18.95 | 21.31 | 24.12 | 26.03 | 25.31 | 25.15 | 26.07 | 25.96 |
| $\bar{t}$ | 239 | 167 | 141 | 127 | 119 | 117 | 114 | 105 | 102 | 99 |
| $\sigma_{\bar{t}}$ | 0.08 | 0.05 | 0.04 | 0.03 | 0.03 | 0.04 | 0.04 | 0.03 | 0.03 | 0.03 |
| $\overline{te}$ | 299 | 203 | 170 | 145 | 137 | 126 | 118 | 118 | 109 | 110 |
| $\sigma_{\overline{te}}$ | 0.11 | 0.06 | 0.04 | 0.04 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\bar{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\bar{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

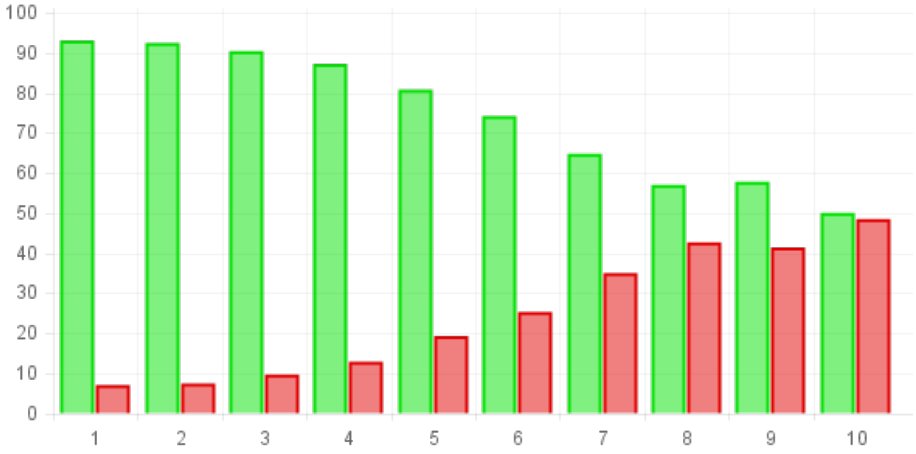


Figura 4.15: Resultados da TxA e TxE para o parâmetro **-ds** depois da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

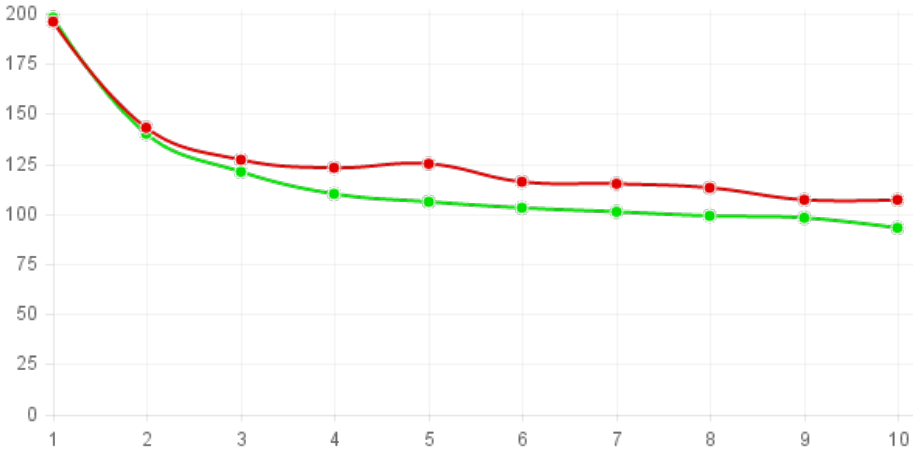


Figura 4.16: Tempo médio de processamento de cada comando para o parâmetro **-ds** depois da adaptação

**Eixo Vertical:** Valores correspondentes à $\overline{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.8: Valor médio e desvio padrão do parâmetro **-ds** depois da adaptação

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 93 | 92.4 | 90.4 | 87.2 | 80.8 | 74.2 | 64.8 | 57 | 57.8 | 50 |
| $\sigma_{TxA}$ | 1.96 | 4.8 | 4.99 | 5.27 | 4.1 | 5.09 | 7.16 | 8.57 | 8.56 | 10.34 |
| $TxE$ | 7 | 7.4 | 9.6 | 12.8 | 19.2 | 25.2 | 35 | 42.6 | 41.4 | 48.4 |
| $\sigma_{TxE}$ | 1.96 | 2 | 2.41 | 2.95 | 4.08 | 5.1 | 7.1 | 8.6 | 8.46 | 10.07 |
| $\overline{t}$ | 198 | 140 | 121 | 110 | 106 | 103 | 101 | 99 | 98 | 93 |
| $\sigma_{\overline{t}}$ | 0.05 | 0.03 | 0.03 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| $\overline{te}$ | 196 | 143 | 127 | 123 | 125 | 116 | 115 | 113 | 107 | 107 |
| $\sigma_{\overline{te}}$ | 0.05 | 0.03 | 0.03 | 0.02 | 0.03 | 0.03 | 0.03 | 0.03 | 0.02 | 0.03 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\overline{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\overline{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

O mesmo comportamento deste parâmetro pode ser visto tanto antes quanto depois dos modelos adaptados. E, de fato, a melhor taxa de acerto se encontra onde este possui valores inferiores a 3, caindo vertiginosamente para os demais.

Um outro ponto importante se revela no gráfico que computa o tempo médio de processamento. É possível ver o aumento da inclinação da curva após $ds = 3$ onde o tempo médio salta de 121ms para 198ms, $\sim$40%, enquanto a taxa de acerto oscila muito pouco, entre $TxA_{ds=1} = 93\%$ e $TxA_{ds=3} = 90\%$.

Sendo assim, é possível escolher $ds = 3$ sem grandes perdas para a aplicação.

### 4.3.5 Análise do Parâmetro -pl_window *(Phonetic Lookahead)*

Nesta técnica de poda, as contabilizações dos pontos se dão pela busca fonética livre de contexto, i.e., pela probabilidade de ocorrência de um determinado fonema sem considerar os estados anteriores.

A variável define quantos *frames* a frente do atual passarão por este processo antes de serem realmente computados. Esta busca é chamada de Busca Rápida e tipicamente seus valores variam de 0, sem busca rápida, até 10, 10 *frames* a frente. Obviamente, valores muito altos inferem na possível queda da acurácia.

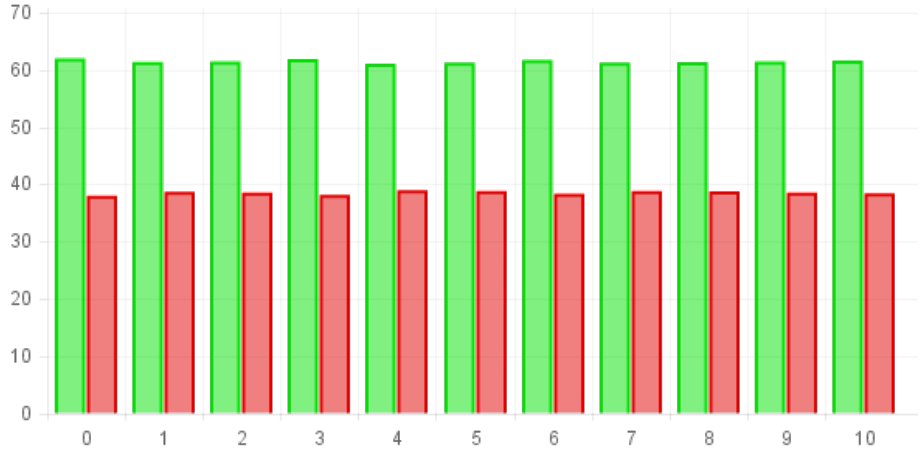


Figura 4.17: Resultados da TxA e TxE para o parâmetro **-pl_window** antes da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

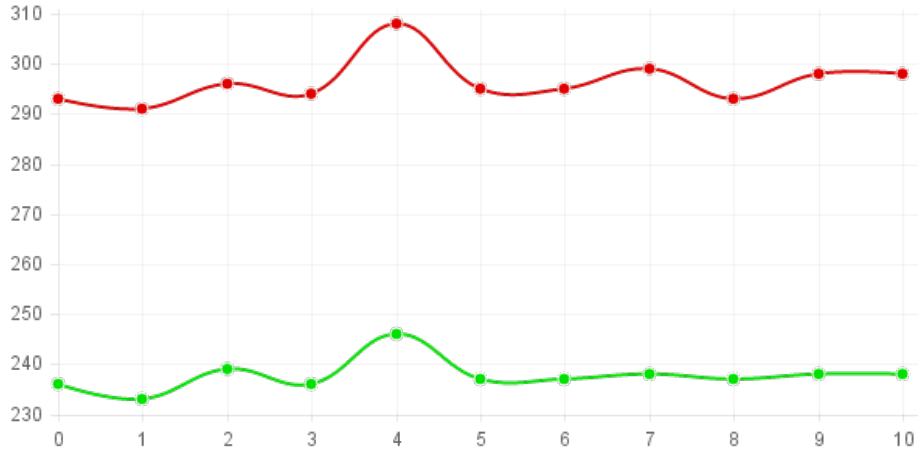


Figura 4.18: Tempo médio de processamento de cada comando para o parâmetro **-pl_window** antes da adaptação

**Eixo Vertical:** Valores correspondentes à $\overline{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.9: Valor médio e desvio padrão do parâmetro **-pl_window** antes da adaptação

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 61.36 | 60.96 | 60.72 | 61.2 | 61.84 | 61.6 | 61.44 | 60.96 | 61.28 | 60.88 | 60.8 |
| $\sigma_{TxA}$ | 13.68 | 13.57 | 13.5 | 13.57 | 13.2 | 13.32 | 13.76 | 13.68 | 13.8 | 13.57 | 13.69 |
| $TxE$ | 37.76 | 38 | 38.32 | 37.92 | 37.2 | 37.12 | 37.6 | 37.92 | 37.76 | 38.08 | 38 |
| $\sigma_{TxE}$ | 14.19 | 14.32 | 14.29 | 14.37 | 13.72 | 13.92 | 14.11 | 14.16 | 14.22 | 14.25 | 14.26 |
| $\overline{t}$ | 236 | 233 | 239 | 236 | 246 | 237 | 237 | 238 | 237 | 238 | 238 |
| $\sigma_{\overline{t}}$ | 0.09 | 0.08 | 0.09 | 0.08 | 0.09 | 0.09 | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 |
| $\overline{te}$ | 293 | 291 | 296 | 294 | 308 | 295 | 295 | 299 | 293 | 298 | 298 |
| $\sigma_{\overline{te}}$ | 0.10 | 0.10 | 0.11 | 0.11 | 0.12 | 0.10 | 0.10 | 0.11 | 0.10 | 0.11 | 0.11 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\overline{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\overline{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

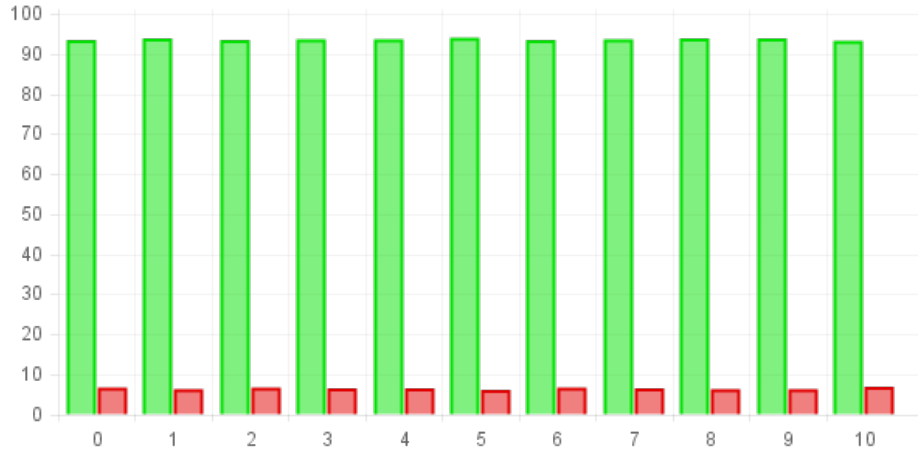


Figura 4.19: Resultados da TxA e TxE para o parâmetro **-pl_window** depois da adaptação

**Eixo Vertical:** Valores correspondentes à TxA (curva em verde) e TxE (curva em vermelho) em percentil.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

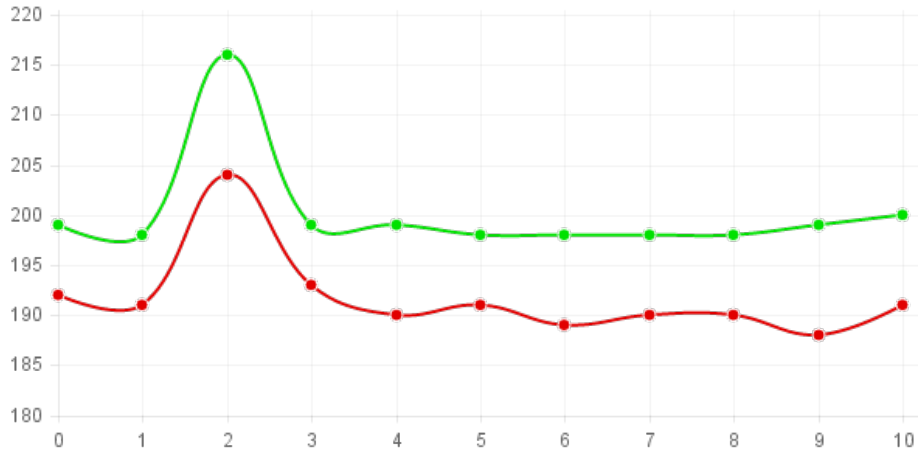


Figura 4.20: Tempo médio de processamento de cada comando para o parâmetro **-pl_window** depois da adaptação

**Eixo Vertical:** Valores correspondentes à $\bar{t}$ (curva em verde) e $\overline{te}$ (curva em vermelho) em ms.
**Eixo Horizontal:** Valores assumidos pelo parâmetro

Tabela 4.10: Valor médio e desvio padrão do parâmetro **-pl_window** depois da adaptação

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $TxA$ | 93.4 | 93.8 | 93.4 | 93.6 | 93.6 | 94 | 93.4 | 93.6 | 93.8 | 93.8 | 93.2 |
| $\sigma_{TxA}$ | 1.82 | 1.83 | 1.95 | 1.72 | 1.84 | 1.78 | 1.99 | 1.97 | 1.96 | 1.96 | 1.84 |
| $TxE$ | 6.6 | 6.2 | 6.6 | 6.4 | 6.4 | 6 | 6.6 | 6.4 | 6.2 | 6.2 | 6.8 |
| $\sigma_{TxE}$ | 1.82 | 1.84 | 1.95 | 1.72 | 1.84 | 1.79 | 2 | 1.97 | 1.97 | 1.97 | 1.85 |
| $\bar{t}$ | 199 | 198 | 216 | 199 | 199 | 198 | 198 | 198 | 198 | 199 | 200 |
| $\sigma_{\bar{t}}$ | 0.06 | 0.05 | 0.07 | 0.06 | 0.06 | 0.05 | 0.05 | 0.05 | 0.05 | 0.06 | 0.05 |
| $\overline{te}$ | 192 | 191 | 204 | 193 | 190 | 191 | 189 | 190 | 190 | 188 | 191 |
| $\sigma_{\overline{te}}$ | 0.06 | 0.05 | 0.07 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 |

$TxA \rightarrow$ taxa média de acertos; $\sigma_{TxA} \rightarrow$ desvio padrão da taxa de acerto;
$TxE \rightarrow$ como taxa média de erro; $\sigma_{TxE} \rightarrow$ desvio padrão da taxa de erro;
$\bar{t} \rightarrow$ tempo médio de processamento de um comando correto; $\sigma_{\bar{t}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando correto;
$\overline{te} \rightarrow$ tempo médio de processamento de um comando errado; $\sigma_{\overline{te}} \rightarrow$ desvio padrão do tempo médio de processamento de um comando errado;

Seguindo a documentação, estes foram variados de 0 a 10, tanto antes quanto depois da adaptação. Como pode ser visto, sua variação pouco influiu na acurácia ou no tempo de processamento. É visto uma alteração significativa da curva no momento em que a variável assume o valor 2 depois da adaptação, mas ao observar com mais atenção a granularidade é possível ver que este salto não configura grande impacto. Sua variação foi de 198ms para 216ms, portanto um aumento

de aproximadamente 9% apenas. Para garantir melhor performance, escolheu-se $pl\_window = 1$.

## 4.4 Resultado Final

Seguindo as análises feitas na seção anterior (Sessão 4.3) foi montado um experimento teste com os valores encontrados, Tabela 4.11 e, em seguida, comparados com os valores padrões.

Tabela 4.11: Parâmetros e seus valores otimizados

| | **-ds** | **-maxhmmpf** | **-maxwpf** | **-pl_window** | **-topn** |
|---|---|---|---|---|---|
| Padrão | 2 | 3000 | 5 | 0 | 4 |
| Melhorado | 3 | 3000 | 2 | 1 | 1 |

O experimento foi rodado nos mesmos moldes dos testes com os modelos adaptados. O resultado pode ser visto na Tabela 4.12, tanto para os valores padrões quanto para os melhorados.

Tabela 4.12: Resultado para os parâmetros otimizados

| | TxA (%) | $\sigma_{TxA}$ | TxE (%) | $\sigma_{TxE}$ | $\bar{t}$ (ms) | $\sigma_{\bar{t}}$ | $\overline{te}$ (ms) | $\sigma_{\overline{te}}$ |
|---|---|---|---|---|---|---|---|---|
| Padrão | 93.12 | 2.12 | 6.87 | 2.12 | 283 | 0.19 | 373 | 0.32 |
| Melhorado | 89.58 | 2.31 | 10.20 | 2.23 | 134 | 0.11 | 137 | 0.08 |

É importante notar uma queda na acurácia de 89% para os parâmetros melhorados contra 93% para os padrões, configurando uma queda de cerca de ∼4% neste quesito. Entretanto, esta mudança apresenta uma melhora no tempo de processamento, de 283 ms para 134 ms, caracterizando uma diminuição de aproximadamente 52% compensando a perda de acurácia. Este comportamento permite que sejam feitos ajustes na aplicação dependendo do dispositivo, isto é, em aparelhos mais potentes e com mais recursos de *hardware* é possível buscar um aumento na acurácia em detrimento do custo computacional, então utiliza-se a configuração padrão da ferramenta. Já, em aparelhos mais simples, busca-se um custo computacional menor em detrimento da acurácia, usando a configuração aqui apresentada. A via de regra, essa escolha é destinada ao usuário na área de configuração da aplicação. Este trabalho não se propõe a implementar esta funcionalidade limitando-se apenas à análise aqui apresentada.

Durante a preparação da estrutura de testes onde testes empíricos foram realizados afim de modelar as melhores práticas e caminhos foi identificada uma debilidade da ferramenta. Esta não implementa nenhuma técnica de detecção de silêncio ou de término de fala, portanto, caso o sistema seja ativado, *START_LISTENING*, e por algum motivo não for desativado, *STOP_LISTENING*, este tende a reconhecer comandos aleatórios e executá-los sem a possibilidade de bloqueios o que pode gerar efeitos indesejáveis ao usuário. Este trabalho não se propõe a analisar este comportamento pois a solução mais adequada é a utilização do *Sphinx4* no lugar do *Pocketsphinx* o que já está mapeado como trabalho futuro.

Da mesma forma, o *Pocketsphinx* apresenta um comportamento atípico em ambientes ruidosos por não implementar nenhuma técnica de redução de ruído. Quando isto ocorre, este não consegue encontrar um resultado final para o áudio de entrada e bloqueia a utilização da aplicação. Para retomar suas funções normais é necessário que o aparelho seja reiniciado. Assim como o anterior, este problema pode ser resolvido com a utilização do *Sphinx4*, logo, não será analisada.

# Capítulo 5

# Conclusões e Trabalhos Futuros

Neste trabalho foi feita uma análise teórica dos recursos voltados para acessibilidade no *Android* e, como resultado, constatou-se que estes são satisfatórios e seguem os princípios do Design Universal. Entretanto, sua documentação é escassa, caso busque-se desenvolver uma ferramente para tecnologia assistiva. Também foi desenvolvida uma aplicação que garante a seu usuário a capacidade de navegação pelas telas do *Android* através da emulação de cliques e *scrolls* acionados por comandos de voz. E por fim, foi analisado o desempenho do *Pocketsphinx* ao ser embarcado em um dispositivo com base na variação de parâmetros sugeridos pela ferramenta, contudo, este possui, assim como o *Android*, uma documentação limitada onde são sugeridos valores para os parâmetros, salvo por algumas exceções. Com base neste ponto, foi realizada uma bateria de testes de performance apenas dos parâmetros que foram completamente compreendidos, **maxhmmph**, **maxwpf**, **topn**, **ds** e **pl_window**.

Os testes executados permitiram a identificação da necessidade de adaptação dos modelos acústicos para o locutor que pretende utilizar o aplicativo. E também foi demonstrada a regularidade comportamental da ferramenta de reconhecimento antes e depois da adaptação com base na variação dos parâmetros. Estes dois pontos por si só já corroboram a utilização do *Pocketsphinx* neste trabalho, porém, o estudo foi além e resultou numa combinação de parâmetros capazes de melhorar o tempo de processamento de cada comando, queda de $\sim$52%, com baixo impacto na acurácia, queda de $\sim$4%, que permite usuários de dispositivos menos potentes possam ter uma boa experiência.

Propõe-se como trabalho futuro, a criação de um repositório contendo instruções de desenvolvimento de tecnologias assistivas para *Android* em exemplos dado a escassez de documentação já citada. Basicamente, o código gerado neste projeto pode ser fragmentado em módulos e agrupados de forma a apresentar seu conteúdo didaticamente. Hoje este já se encontra em um repositório público e aberto para qualquer um que queira acessá-lo aguardando apenas sua reestruturação.

Um outro ponto está no desenvolvimento de funcionalidades a fim de atingir um nível de maturidade que permita a disponibilização do aplicativo. Para isso, será necessário concluir o porte do *Sphinx4* para *Android*, dado que este, além de ter melhor desempenho que o *Pocketsphinx*, também possui uma estrutura de treinamento de seus modelos que os tornam mais adequados ao locutor. Também é requerida, a implementação de um IME para edição/criação textual. E por fim, a preparação de uma base de dados para o português brasileiro.

# Referências Bibliográficas


[1] GOOGLE. "Just speak it: introducing Voice Actions for Android". http://googlemobile.blogspot.com.br/2010/08/just-speak-it-introducing-voice-actions.html, 2010.

[2] GOOGLE. "Google TalkBack". https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback&hl=pt_BR, .

[3] YOUNG, S. "HTK - The Hidden Markov Model Toolkit". http://htk.eng.cam.ac.uk/, 2015.

[4] LEE, K.-F., HON, H.-W., REDDY, R. "An Overview of the SPHINX Speech Recognition System", *IEEE Transactions on Acoustcs Speech and Signal Processing*, v. 38, n. 1, January .

[5] GOLDSMITH, S., OF ARCHITECTURE, A. A. G. B. S. *Designing for the disabled: a manual of technical information*. Royal Institute of British Architects, Technical Information Service, 1963. Disponível em: <https://books.google.com.br/books?id=aotqAAAAMAAJ>.

[6] UNIVERSALDESIGN.COM. "What is Uneversal Design?" http://www.universaldesign.com/about-universal-design.html.

[7] GOOGLE. "Accessibility". http://developer.android.com/design/patterns/accessibility.html, 2015.

[8] GOOGLE. "Eyes Free Project". https://code.google.com/p/eyes-free/, 2013.

[9] GOOGLE. "JustSpeak". http://eyes-free.googlecode.com/svn/trunk/accessibilityServices/justspeak/docs/overview.html, 2013.

[10] ZHONG, Y., RAMAN, T., BURKHARDT, C., et al. "JustSpeak: Enabling Universal Voice Control on Android", 2014.

[11] OLIVEIRA, V. *Reconhecimento de Fala Contínua Para O Português Brasileiro Baseado Em HTK e SPHINX*. Bsc dissertação, Escole Politécnica/COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2010.

[12] SKINNER, D. G. C. D. P., KEMERAIT, R. C. "The Cepstrum: A Guide to Processing", *Proceedings of the IEEE*, v. 65, n. 10, October 1977.

[13] DAVIS, MERMELSTEIN. *Comparison of Parametric Representations for Monosyllable Word Recognition in Continuously Spoken Sentences*. IEEE Transactions on Acoustic, Speech and Signal Processing, 1980.

[14] RABINER, L., JUANG, B.-H. *Fundamentals of Speech Recognition*. Pentice-Hall International, Inc, 1993.

[15] ORACLE. "Java Sound API". http://www.oracle.com/technetwork/java/index-jsp-140234.html, .

[16] GOOGLE. "Audio Record API". http://www.oracle.com/technetwork/java/index-jsp-140234.html, .

[17] GOOGLE. "Android NDK". https://developer.android.com/tools/sdk/ndk/index.html, .

[18] ORACLE. "Java Speech Grammar Framework". http://cmusphinx.sourceforge.net/doc/sphinx4/edu/cmu/sphinx/jsgf/JSGFGrammar.html, .

[19] GOOGLE. "Text To Speech (TTS)". http://developer.android.com/reference/android/speech/tts/TextToSpeech.html, .

[20] ROBSON, E., BATES, B., FREEMAN, E. "Head First Design Patters". 1 ed., cap. 2, O'Reilly Media, 2014.

[21] CORMEN, T. H., STAIN, C., RIVEST, R. L., et al. "Introduction to Algorithms". 3 ed., cap. 10, The MIT Press, 2009.

[22] GOOGLE. "Input Method Editor (IME)". http://developer.android.com/guide/topics/text/creating-input-method.html, .

[23] INC, D. "Easy Voice Recorder". https://play.google.com/store/apps/details?id=com.coffeebeanventures.easyvoicerecorder&hl=pt_BR, 2015.

[24] FUNDATION, N. "NodeJS". https://nodejs.org/, 2015.

[25] MONGODB, I. "MongoDB". https://www.mongodb.org/, 2015.

[26] MELLON, C. “Biblioteca Pocketsphinx para Android”. https://github.com/cmusphinx/pocketsphinx-android, 2015.

[27] INC, G. “Gradle”. http://gradle.org/, 2015.

[28] SWIG. “Simplified Wrapper and Interface Generator”. http://www.swig.org/, 2015.

[29] GOOGLE. “Android SDK”. http://developer.android.com/intl/pt-br/sdk/index.html, 2015.

[30] MELLON, C. “Biblioteca base do CMUSphinx”. https://github.com/cmusphinx/sphinxbase, 2015.

[31] MELLON, C. “Biblioteca Pocketsphinx”. https://github.com/cmusphinx/pocketsphinx-android, 2015.

# Apêndice A

# Os Comandos e suas Funções

## A.1 Comandos Globais

Aqui se encontra a lista de comandos que se dispõe a substituir/emular os botões físicos do aparelho.

**GO HOME:** Redireciona o usuário para a tela inicial

**GO BACK:** Redireciona o usuário para tela anterior

**SETTINGS:** Abre a área de configurações do aparelho

**POWER:** Abre as opções de gerenciamento de energia, desligar/reiniciar

**LOUDER:** Aumenta o volume do aparelho

**QUITER:** Diminui o voluma do aparelho

## A.2 Comandos de Controle da Ferramenta

**START LISTENING:** Ativa o sistema para entrada de novos comandos

**STOP LISTENING:** Bloqueia a entrada de outros comandos

**CLEAR:** Remove todos os Badges da tela

**WRITE:** Entra em modo de escrita

**REFRESH:** Reposiciona todos Badges na tela

## A.3 Comandos de Navegação

**GO FORWARD:** Busca todos os elementos passíveis de serem "rolados" na tela atual, se encontra mais de um é atribuído um rótulo numéico a eles de modo que ao usuário soletrar o número de um deles este e rolado para "frente" (direita ou para cima). Caso tenha apenas 1 item "rolável", este é exdecutado automaticamente.

**GO BACKWARD** Similar ao anterior, porém no sentido oposto "trás" (esquerda ou para baixo).

**HISTORY:** Abre a lista dos últimos aplicativos utilizados

**NOTIFICATION:** Expõe a área de notificações do aparelho

Lista de comandos para filtragem dos Badges.

**ONE**

**TWO**

**THREE**

**FOUR**

**FIVE**

**SIX**

**SEVEN**

**EIGHT**

**NINE**

**ZERO**

# Apêndice B

# Arquivos Utilizados nos Testes

## B.1 JSON de Entrada

A seguir pode ser visto a estrutura do arquivo JSON utilizado para alimentar o aplicativo de execução de testes. Como este é muito extenso, porém repetitivo, o apresentado abaixo é uma forma reduzida do original.

```
[
    {
        ‘‘scenario’’: [
            {
                ‘‘integerInputs’’: {
                    ‘‘-maxwpf’’: 1
                }
            },
            {
                ‘‘integerInputs’’: {
                    ‘‘-maxwpf’’: 2
                }
            }
        ],
        ‘‘scenario’’: [
            {
                ‘‘integerInputs’’: {
                    ‘‘-ds’’: 1
                }
            },
            {
                ‘‘integerInputs’’: {
```

```
                    ‘‘-ds’’: 2
                }
            }
        ]
    }
]
```

Os *scenarios* consistem em grupos de *inputs* baseados em tipo que configuram o reconhecedor a cada inicialização. Esta estrutura foi pensada com o intuito de, no futuro, inserir um conjunto de parâmetros de uma única vez.

## B.2 JSON de Saída

Assim como o anterior, aqui se encontra apenas uma pequena amostra do formato original com dados fictícios.

```
[
    {
        variableName: ‘‘-ds’’,
        variableValue: 5,
        command: ‘‘go home’’,
        right: 3,
        wrong: 2,
        durationOfRight: [185, 328, 253],
        durationOfWrong: [405, 678],
        mapOfWrong: {one: 1, seven: 1}
    },
    {
        variableName: ‘‘-topn’’,
        variableValue: 3,
        command: ‘‘go home’’,
        right: 4,
        wrong: 1,
        durationOfRight: [185, 328, 253, 179],
        durationOfWrong: [405],
        mapOfWrong: {two: 1}
    }
]
```

Desta forma, podem ser encontrados, para cada execução par variável:valor, o número de acertos (*right*), número de erros (*wrong*), tempo de processamento de

cada arquivo em ms tanto dos acertos (*durationOfRight*) quanto dos erros (*durationOfWrong*) e o mapa de erros (*mapOfWrong*) que contabiliza quais comandos foram encontrados quando houve um erro e quantas vezes isto ocorreu.

# Apêndice C

# Adaptação de Modelos Acústicos

## C.1 Preparação do Ambiente

Para adaptação dos modelos acústicos com o Sphinx é necessário a instalação dos seguintes pacotes:

- SphinxTrain
- SphinxBase
- PocketSphinx

Estes fazem uso dos interpretadores Perl e Python. Este último na versão 2, caso contrário algumas funcionalidades não irão funcionar adequadamente.

Neste projeto os modelos adaptados foram aqueles fornecidos pelo PocketSphinx para reconhecimento contínuo do inglês americano (**en-us-ptm**) os quais serão usados neste tutorial.

Em seguida é necessário criar uma pasta onde todo o processo de adaptação ocorrerá. Dentro desta, os modelos a serem adaptados devem ser adicionados, no caso, **en-us-ptm**.

## C.2 Preparação dos Dados para Adaptação

Para esta etapa, o primeiro passo remete a preparação dos arquivos de áudio. Estes precisam ter as mesmas configurações dos utilizados nos modelos originais. Neste contexto, os arquivos de áudio foram gravados em formato WAV 16KHz em MONO com Single Chanel. E armazenados numa pasta **wav**.

Depois são criados dois arquivos:

- *models.fileids* - arquivo com o nome dos arquivos de áudios que serão utilizados.

wav/audio1
wav/audio2
wav/audio3
...

- *models.transcription* - transcrição dos arquivos de áudio referenciados no arquivo acima

<s> start listening </s> (wav/audio1)
<s> start listening </s> (wav/audio2)
<s> stop listening </s> (wav/audio3)
...

## C.3 Geração dos Arquivos de Características Acústicas (MFCC)

Pare adaptar os modelos acústicos será necessário a extração das características do áudios a serem utilizados. Isto pode ser feito com o auxílio do *sphinx_fe* provido pelo SphinxBase como demonstrado a seguir:

```
sphinx_fe -argfile en-us-ptm/feat.params \
        -samplarate -1600 -c models.fileids \
        -di . -do . -ei wav -eo mfc -mswav yes
```

Desta forma as características de cada áudio será armazenada junto aos originais, mesmo diretório, em um outro arquivo com o mesmo nome, porém com extensão mfc.

## C.4 Descompressão dos Modelos em Binário

Em alguns casos os modelos acústicos são distribuídos em arquivos no formato binário, *mdef*. As ferramentas de adaptação esperam arquivos textuais para serem processados, logo, uma conversão é requerida. Esta pode ser feita com uso da ferramenta *pocketsphnx_mdef_converter*, como mostrado a baixo:

```
pocketsphinx_mdef_converter -text en-us-ptm/mdef en-us-ptm/mdef.txt
```

## C.5 Coleta de Estatísticas

O próximo passo consiste em coletar estatísticas dos dados a serem utilizados na adaptação. Isto é feito através da ferramenta *bw* pertencente ao *SphinxTrain*. Esta operação é apresentada a seguir:

```
bw \
-hmmdir en-us-ptm \
-moddeffn en-us-ptm/mdef.txt \
-ts2cbfn .ptm. \
-feat 1s_c_d_dd \
-svspec 0-12/13-25/26-38 \
-cmn current \
-agc none \
-dictfn cmudict-en-us.dict \
-ctlfn models.fileids \
-lsnfn models.transcription \
-accumdir .
```

Os argumentos neste comando são obtidos no arquivo *en-us-ptm/feat.params* e é imprescindível que sejam os mesmos, caso contrário o programa não será executado. Também é importante ressaltar que nem todo parâmetro contido no arquivo citado é suportado pelo *bw*, portanto, é necessário atenção ao defini-los.

## C.6 Criação da Matriz de Transformação com *MLLR*

Através da ferramenta *mllr_solve* do *SphinxTrain* essa operação se torna possível. Basta executar o seguinte comando:

```
mllr_solve \
-meanfn en-us-ptm/means \
-varfn en-us-ptm/variances \
-outmllrfn mllr_matrix -accumdir .
```

Este comado gerará um arquivo de adaptação chamado *mllr_matrix*, referenciado no comando por *-outmllrfn*. Para utilizá-lo faça referência na configuração do reconhecedor *Pocketsphinx* como:

```
-mllr caminho/para/mllr_matrix
```

## C.7 Adaptação dos Modelos Acústicos com *MAP*

Diferentemente do anterior o *MAP* não cria um arquivo a parte para ser utilizado, mas modifica cada parâmetro existente no modelo. Sendo assim, inicialmente cria-se uma cópia dos modelos no mesmo diretório do original contendo exatamente os mesmos arquivos. Esta será referenciada como *en-us-ptm-adapt*.

Em seguida, utiliza-se o programa *map_adapt* do *SphinxTrain* da seguinte forma:

```
map_adapt \
-moddeffn en-us-ptm/mdef.txt \
-ts2cbfn .ptm. \
-meanfn en-us-ptm/means \
-varfn en-us-ptm/variances \
-mixwfn en-us-ptm/mixture_weights \
-tmatfn en-us-ptm/transition_matrices \
-accumdir . \
-mapmeanfn en-us-ptm-adapt/means \
-mapvarfn en-us-ptm-adapt/variances \
-mapmixwfn en-us-ptm-adapt/mixture_weights \
-maptmatfn en-us-ptm-adapt/transition_matrices
```

Para utilizar, substitua o *en-us-ptm* pelo *en-us-ptm-adapt* no sistema de reconhecimento de voz.

## C.8 Compressão dos Modelos para Binário

Caso haja necessidade de redução de tamanho do espaço de armazenagem dos modelos é possível transformá-los em binário com o auxílio do *mk_s2sendump*:

```
mk_s2sendump \
-pocketsphinx yes \
-moddeffn en-us-ptm-adapt/mdef.txt \
-mixwfn en-us-ptm-adapt/mixture_weights \
-sendumpfn en-us-ptm-adapt/sendump
```

Agora é possível remover os arquivos *mdef.txt* e *mixture_weights* pois estes não serão mais utilizados pelo decodificador.

# Apêndice D

# Implementação do Pocketsphinx no Android

O Sphinx, hoje em sua versão 4, apesar de estar escrito em Java, tem seu foco centrado em aplicações para Desktops fazendo uso de ferramentas e bibliotecas pertencentes as JVM deste grupo. O exemplo mais claro está na biblioteca para manipulação de áudio, base do reconhecimento, JavaSound. Esta não está disponível para Android, logo impossibilitando o uso da mesma neste ambiente.

Tendo isso em mente, o time de desenvolvimento do conjunto de ferramentas CMUSphinx adaptou a versão anterior do framework, Sphinx 3, para dispositivos móveis. Dado que este foi escrito em C, utilizou-se a ferramenta NDK do Android que permite a injeção e utilização de códigos escritos em C/C++ como nativos da plataforma.

Afim de prover um guia para implementação do Pocketsphinx no Android, aqui serão apresentadas as etapas para tal, posto que todas estas foram testadas em um computador DELL Vostro, 4G de RAM e com SO ArchLinux.

## D.1 Preparação do Ambiente

A biblioteca Pocketsphinx pode ser gerada através da compilação do código fonte hospedado no GitHub [26]. Para que isto seja possível é necessário ter instalado no ambiente local o Gradle [27], Swig [28], Android NDK [17] e Android SDK [29]. E, além do código fonte da biblioteca, também se faz necessário o fonte do Sphinxbase [30] e do Pocketsphinx [31] armazenados na mesma pasta na seguinte estrutura:

```
/path/para/projeto/sphinxbase
/path/para/projeto/pocketsphinx
/path/para/projeto/pocketsphinx-android
```

## D.2 Geração da Biblioteca

O primeiro passo é definir onde o Android SDK e Android NDK estão instalados. Isso é feito no arquivo de propriedades do pocketsphinx-android, gradle.properties, da seguinte forma:

```
sdkDir = /path/para/android/sdk
ndkDir = /pth/para/android/ndk
sdkVersion = 22
ndkExt =
```

Estes dados serão utilizados pelo *gradle* no processo de construção da biblioteca. É importante ressaltar a utilização dos parâmetros *sdkVersion* e *ndkExt*. O primeiro define a versão do *Android SDK* que será utilizada. Já o segundo, é utilizado para atribuir a extensão do arquivo de execução do *NDK*. Este arquivo não possui nenhuma extensão se o SO for *Linux* ou *Mac*, então esta linha deve ser definida como descrita acima. Entretanto, se o SO for *Windows* esta deve ser acrescida de *.cmd* ao final.

```
ndkExt = .cmd
```

Por fim, basta compilar executando o comando abaixo na pasta do *pocketsphinx-android*.

```
gradle build
```

Com isso será gerado o arquive *pocketsphinx-android-5prealph-nolib.jar* na pasta */path/para/projeto/pocketsphinx-android/build/libs* e os arquivos com extensão *.so* referentes a cada arquitetura na pasta */path/para/projeto/pocketsphinx-android/libs*.

## D.3 Configuração da Aplicação

Para utilizar a biblioteca numa aplicação basta copiar o arquivo *pocketsphinx-android-5prealpha-nolib.jar* para a pasta *app/libs* e os arquivos com extensão *.so* para a pasta *app/src/main/jniLibs*, esta última geralmente precisa ser criada posto que o *Android Studio* não o faz por padrão.

Em seguida é preciso definir no arquivo de configurações do aplicativo as arquiteturas objetivadas para que o *NDK* possa interpretar seu código e portá-lo para *Java*. Isto é feito adicionando o conjunto *productFlavor* no arquivo de construção, *app/build.gradle*, como descrito a seguir:

```
android {
    ...
    productFlavors {
        x86 {
            ndk {
                abiFilters(‘‘x86’’, ‘‘x86_64’’)
            }
        }
        mips {
            ndk {
                abiFilters(‘‘mips’’, ‘‘mips64’’)
            }
        }
        arm {
            ndk {
                abiFilters(‘‘armeabi-v7a’’, ‘‘armeabi’’, ‘‘arm64-v8a’’)
            }
        }
    }
}
```

Para que tudo acima possua o efeito desejado é necessário estar instalado no *Android Studio* o *plugin* de suporte ao *NDK*. Este está disponível na lista de *plugins* com o nome *Android NDK Support*.

O *Pocketsphinx* utiliza um conjunto de arquivos contendo a base de dados para o processamento da fala. Esses arquivos são verificados através da comparação de sues respectivos arquivos *Hash MD5* antes de serem utilizados e na ausência destes é gerado uma exceção bloqueando o uso da ferramenta. Para contornar isto, convenciona-se que os modelos serão armazenados na pasta *app/src/main/assets/models* e assim gerar dinamicamente os arquivos *MD5* durante o processo de compilação com o auxílio do arquivo *XML* armazenado na pasta *app/assets.xml* apresentado abaixo:

```
<?xml version=‘‘1.0’’ encoding=‘‘UTF-8’’?>
<project name=‘‘assets’’>
    <property name=‘‘assets.list.name’’ value=‘‘assets.lst’’/>
    <property name=‘‘assets.dir’’ value=‘‘src/main/assets/sync’’/>
    <property name=‘‘assets.hash.type’’ value=‘‘md5’’/>
    <property name=‘‘assets.ctl.files’’
    value=‘‘**/*.${assets.hash.type},${assets.list.name}’’/>
```

```
    <fileset id=‘‘assets’’ dir=‘‘${assets.dir}’’
    excludes=‘‘${assets.ctl.files}’’/>

    <target name=‘‘clean_assets’’>
        <delete>
            <fileset dir=‘‘${assets.dir}’’
            includes=‘‘${assets.ctl.files}’’/>
        </delete>
    </target>

    <target name=‘‘list’’>
        <pathconvert dirsep=‘‘/’’
        pathsep=‘‘${line.separator}’’
        refid=‘‘assets’’ property=‘‘asset.list’’>
            <map from=‘‘${basedir}/${assets.dir}/’’ to=‘‘/>
        </pathconvert>
        <echo message=’’${asset.list}‘‘
        file=’’${assets.dir}/${assets.list.name}‘‘/>
    </target>

    <target name=’’checksum‘‘>
        <checksum algorithm=’’${assets.hash.type}‘‘>
            <fileset refid=’’assets‘‘/>
        </checksum>
    </target>
</project>
```

Para executá-lo adiciona-se o seguinte comando ao arquivo de compilação *app/build.gradle*:

```
ant.importBuild 'assets.xml'
preBuild.dependsOn(list, checksum)
clean.dependsOn(clean_assets)
```

Desta forma, toda vez que o projeto for compilado os arquivos *MD5* serão gerados.

## D.4 Utilização da Biblioteca

Para fazer uso da biblioteca, basta que a classe responsável por gerenciar as requisições e processamento dos resultados obtidos pelo *Pocketsphinx* implementar a interface *RecognitionListener*.

```
public class MyRecognizer implements RecognitionListener
```

É importante ressaltar que no *Android SDK* existe uma interface com o mesmo nome, então deve-se ter certeza de que a implementada pertence ao pacote *edu.cmu.pocketsphinx*.

Esta interface possui 5 métodos a serem sobrescritos cuja a descrição segue a baixo:

- *onPartialResult* – Método chamado quando uma parte do áudio dado como entrada foi reconhecido.
- *onResult* – Método chamado quando o áudio dado como entrada foi reconhecido.
- *onBeginningOfSpeech* – Método chamado quando a ferramenta detecta o início da fala no áudio dado como entrada.
- *onEndOfSpeech* – Método chamado quando a ferramenta detecta o final da fala no áudio dado como entrada.
- *onTimeout* – Método chamado quando a ferramenta não consegue encontrar um resultado em definitivo dentro do tempo esperado.

Em seguida é necessário preparar o reconhecedor antes de começar o processamento. O primeiro passo é recuperar os modelos armazenados na pasta *assets* e associá-los ao reconhecedor. Este processo precisa ser feito em uma *Thread* diferente da corrente para que não haja *Exception*, logo pode-se utilizar a classe *AsyncTask* para facilitar esta operação.

```
public setup() {
    new AsyncTask<Void, Void, Exception>() {
        @Override
        protected Exception doInBackground(Void... params) {
            try {
                // Recuperação do diretório assets
                Assets assets = new Assets(context);
                File assetDir = assets.syncAssets();
```

```
                // Método chamado para configurar o reconhecedor
                setupRecognizer(assetDir);
            } catch (IOException e) {
                Log.e(TAG, e.getMessage());
                return e;
            }
            return null;
        }

        @Override
        protected void onPostExecute(Exception result) {
            if (result != null) {
                Log.e(TAG, ‘‘Failed to init recognizer’’ + result.getMessage());
            } else {
                // Método usado para inicializar a busca por resultados
                switchSearch(KWS_SEARCH);
                currentSearch = KWS_SEARCH;
            }
        }
    }.execute();
}
```

Após a a recuperação dos modelos, ainda na *Thread* de recuperação, é criada a instância do reconhecedor assim como sua configuração. No exemplo esta etapa foi executada no método *setupRecognizer*. O *Pocketsphinx* utiliza o padrão de projeto *Builder* para criação e configuração o que facilita seu processo.

```
...
import static edu.cmu.pocketsphinx.SpeechRecognizerSetup.defaultSetup;
...

private void setupRecognizer(File assetsDir) {
    File modelsDir = new File(assetsDir, ‘‘models’’);
    try {
        // Padrão Builder implementado pela ferramenta
        recognizer = defaultSetup()
        .setAcousticModel(new File(modelsDir, ‘‘hmm/en-us-ptm-adapt’’))
        .setDictionary(new File(modelsDir, ‘‘dict/cmudict-en-us.dict’’))
```

```
        // Threshold to tune for keyphrase
        // Limiar utilizado nas técnicas de poda
        .setKeywordThreshold(1e-40f)

        // Use context-independent phonetic search,
        // context-dependent is too slow for mobile
        .setBoolean(‘‘-allphone_ci’’, true)

        / Retorna uma instância da classe SpeechRecognizer
        .getRecognizer();
    } catch (IOException e) {
        Log.e(TAG, e.getMessage());
    }
    recognizer.addListener(this);

    // Definição dos tipos de buscas/reconhecimento
    recognizer.addKeyphraseSearch(KWS_SEARCH, KEYPHRASE);

    File navigationGrammar = new File(modelsDir, ‘‘grammar/navigation.gram’’);
    recognizer.addGrammarSearch(NAVIGATION_SEARCH, navigationGrammar);
}
```

A definição do tipo de busca se dá por um par chave:valor em que a sua ativação só ocorre quando sua chave é chamada, exemplo *recognizer.startListening(CHAVE)*, dado que no processo de configuração do reconhecedor este tipo tenha sido adicionado. No bloco de código acima, as variáveis *KWS_SEARCH* e *KEYPHRASE* são duas *Strings* onde, a primeira é o identificador do tipo de busca e a segunda a palavra a ser buscada. Nesta modalidade o reconhecedor somente gerará resultados (*onResult* e *onPartialResult*) caso a *KEYPHRASE* seja identificada, este tipo é chamado de busca por palavra-chave. Assim como a anterior, na penúltima linha do código, é adicionado outro módulo de busca, porém, nesta modalidade, a busca será feita numa gramática na estrutura da JGSF.

A inicialização do processo de reconhecimento ocorre com a chamada do método *switchSearch* ao final da inicialização já mostrado anteriormente. A implementação deste método segue a baixo:

```
private void switchSearch(String searchName) {
    recognizer.stop();
```

```
    if (searchName.equals(KWS_SEARCH))
        recognizer.startListening(searchName);
    else
        recognizer.startListening(searchName, 10000);

}
```

Como pode ser visto, o reconhecedor precisa ser parado a cada mudança de tipo de busca e reinicializado. Além disso, buscas que não sejam por palavras-chave precisam receber um segundo parâmetro ao serem inicializadas. Este configura o intervalo de tempo de áudio a ser processado, no código este é de 10000 milisegundos ou 10 segundos. Ao término deste intervalo o método *onResult* é chamado. Neste momento o desenvolvedor deve escolher o que fazer. De modo geral, costuma-se manter o tipo de busca como pode ser visto a baixo:

```
@Override
public void onResult(Hypothesis hypothesis) {
    if (hypothesis != null) {
        // Recupera o texto encontrado
        String text = hypothesis.getHypstr();

        // Continua no tipo de busca atual
        if (!recognizer.getSearchName.equals(KWS_SEARCH))
            switchSearch(currentSearch);
    }
}
```

É muito comum em sistemas *command-control* que a mudança de tipos de busca ocorra sem que a busca esteja completa, i.e., chamar o método *onResult*. Se os comandos são pequenos a mudança pode ser feita no método *onPartialResult*.

```
@Override
public void onPartialResult(Hypothesis hypothesis) {
    if (hypothesis != null) {
        // Recupera o texto encontrado
        String text = hypothesis.getHypstr();

        // Muda o tipo de busca
        if (!recognizer.getSearchName.equals(KWS_SEARCH)) {
            if(text.equals(KEYPHRASE)
                switchSearch(NAVIGATION_MENU);
```

```
        }
    }
}
```

Por fim, sempre ao encerrar a aplicação o reconhecedor deve ser finalizado e sua infraestrutura desligada.

```
public void finishRecognizer() {
    recognizer.cancel();
    recognizer.shutdown();
}
```