UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

JESSE NAZARIO DE MEDEIROS

AMBIENTE DE RECONHECIMENTO DE FALA PARA ESCREVER CÓDIGOS DE PROGRAMAÇÃO INTEGRADO AO GNU EMACS

CRICIÚMA

2018

JESSE NAZARIO DE MEDEIROS

AMBIENTE DE RECONHECIMENTO DE FALA PARA ESCREVER CÓDIGOS DE PROGRAMAÇÃO INTEGRADO AO GNU EMACS

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Esp. Gilberto Vieira da Silva

CRICIÚMA

2018

JESSE NAZARIO DE MEDEIROS

AMBIENTE DE RECONHECIMENTO DE FALA PARA ESCREVER CÓDIGOS DE PROGRAMAÇÃO INTEGRADO AO GNU EMACS

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Desenvolvimento de Software

Criciúma, 25 de junho de 2018.

BANCA EXAMINADORA

Prof. Gilberto Vieira da Silva- Esp - UNESC - Orientador

Prof. Anderson Luis Furlan - Me - CEDUP

Prof. Matheus Leandro Ferreira - Esp - UNESC

“Dedico este trabalho a meus familiares, namorada e amigos.”

AGRADECIMENTOS

Gostaria de agradecer primeiramente meus familiares que tanto me apoiaram durante esta etapa da minha vida. Quero agradecer também minha namorada, amigos, e colegas, que me ajudaram a suportar todas as dificuldades.

Por último, mas não menos importante, agradeço ao meu orientador, Gilberto Vieira da Silva, que não mediu esforços para me auxiliar durante o desenvolvimento deste trabalho.

“A essência de uma mente independente não está no que ela pensa, mas no modo como ela pensa.”

Christopher Hitchens

RESUMO

Existe uma crescente demanda por profissionais da área de Tecnologia da Informação no mundo inteiro. Entretanto, várias pesquisas concluem que o uso em excesso de computador, aliado a má postura, podem causar vários tipos de lesões, muitas vezes difíceis de tratar. Este trabalho propõe utilizar comandos de voz como um meio de interação alternativo com o computador para programar, buscando reduzir ou substituir o uso de teclado e mouse para escrever códigos de programação, auxiliando tanto na prevenção de lesões quanto na inclusão de pessoas com deficiências físicas. Para isso, foi desenvolvido, utilizando apenas com ferramentas gratuitas e software livre, um protótipo para escrever códigos de programação através de comandos de voz. Este protótipo utiliza a engine de reconhecimento de fala CMU Sphinx para realizar a conversão de áudio para texto e executa as ações correspondentes aos comandos falados no editor de texto GNU Emacs por meio de uma conexão TCP/IP. Através deste protótipo foi possível escrever, editar, manipular e executar códigos de programação. Entretanto, foi necessário utilizar um dicionário de palavras limitado para obter-se uma melhor precisão com o CMU Sphinx, visto que, utilizando um dicionário com todas as palavras do idioma, a probabilidade de retornar falsos positivos é bem maior, atrapalhando o fluxo dos comandos. Este ambiente desenvolvido foi escrito em Python e otimizado para trabalhar com a linguagem de programação Ruby, podendo também ser otimizado para outras linguagens.

Palavras-chave: CMU Sphinx, GNU Emacs, Reconhecimento de fala.

ABSTRACT

There is a growing demand of jobs for the Information Technology field in the entire world. However, many studies concluded that excessive computer usage, along with a bad posture, may cause a variety of injuries, which are usually hard to treat. This paper's goal is to use voice commands as an alternative interaction method to program, aiming to reduce or replace the keyboard and mouse usage to write computer code, helping to prevent injuries and providing opportunity to people with physical disabilities. With that in mind, a prototype to write computer code was developed, using only free software tools. This prototype uses the speech recognition engine CMU Sphinx as the backend to perform the speech-to-text conversion and executes the corresponding actions to the spoken commands in the GNU Emacs text editor over a TCP/IP connection. Through this prototype it was possible to write, edit, manipulate and run computer code. However, it was necessary to utilize a limited word dictionary to achieve a better precision under CMU Sphinx, since, when using a full language dictionary, there is a much higher probability of false positives, interrupting the command flow. This environment was written in Python and optimized to work with the Ruby programming language, while still being possible to be optimized for other languages.

Keywords: CMU Sphinx, GNU Emacs, Speech recognition

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interfaces*
DORT	Distúrbios Osteomusculares Relacionadas ao Trabalho
HMM	*Hidden Markov Mode*
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	*Integrated Development Environments*
LER	Lesões por Esforço Repetitivo
MR	Módulo de reconhecimento
MGE	Módulo GNU Emacs
SGDB	Sistema de Gerenciamento de Banco de Dados
SRF	Sistemas de Reconhecimento de Fala
TCP/IP	Transmission Control Protocol/Internet Protocol

SUMÁRIO

1 INTRODUÇÃO

Profissões como desenvolvedor de software e similares estão se tornando cada vez mais populares, sendo uma das áreas que mais crescem nos Estados Unidos (EUA, 2017). Contudo, estudos indicam que o uso excessivo de teclado e mouse, principalmente quando não são tomados os devidos cuidados, está diretamente relacionado a Lesões por Esforço Repetitivo (LER) (BLATTER; BONGERS, 2002, tradução nossa).

A tarefa de manipulação de código-fonte, que costuma ser a principal tarefa de um desenvolvedor de software, dificilmente pode ser executada sem teclado e mouse. Existem poucas alternativas para esse tipo de interação, e, em geral, costumam ser pouco produtivas (FONTANEZ; FRANCO, 2014, tradução nossa).

Por isso, é importante pensar em novos métodos para realizar esse tipo de tarefa, buscando diminuir a incidência de LER nos profissionais da área, além de auxiliar aqueles que, por algum tipo de limitação física, possuem dificuldade para utilizar teclado ou mouse.

Uma dessas alternativas pode ser o reconhecimento de fala. Já existem meios de controlar o computador (MICROSOFT, 2016) e celulares (GOOGLE, 2017a) apenas com comandos de voz, mas esses não são focados para a manipulação de código fonte.

Neste trabalho, desenvolveu-se um protótipo que possibilita manipular códigos de programação através de comandos de voz, onde o CMU Sphinx é responsável pela conversão de fala para texto e o GNU Emacs é o editor de texto onde os comandos serão executados e visualizados. Este protótipo foi desenvolvido em GNU/Linux, podendo potencialmente ser portado para outros sistemas operacionais pelo fato de terem sido utilizadas ferramentas multiplataforma durante o desenvolvimento.

1.1 OBJETIVO GERAL

Desenvolver um ambiente de programação controlado por comandos de voz constituindo uma alternativa ao uso de teclado e mouse.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos desta pesquisa compreendem:

a) Compreender as doenças ocasionadas pelo uso constante de teclado e mouse e relação com a atividade de desenvolvimento de software;
b) Estudar a teoria e as tecnologias envolvidas no processo de reconhecimento de fala;
c) Propor um protótipo de ambiente de programação controlado por comandos de voz em uma linguagem de programação.

1.3 JUSTIFICATIVA

O programador é o profissional que trabalha, fundamentalmente, no desenvolvimento de softwares. Tendo em vista que é necessário o uso constante de teclado e mouse para este trabalho, estes profissionais estão propícios a desenvolver LER (TITTIRANONDA; BURASTERO; REMPEL, 1999, tradução nossa).

Até 2019, estima-se que a América Latina precisará de 449 mil profissionais de Tecnologia da Informação (TI) para atender à demanda do mercado. A área que oferece mais empregos é a de sistemas de informação, no desenvolvimento de softwares específicos (PINEDA; GONZALEZ, 2016, tradução nossa).

Um ambiente que não necessite de teclado e mouse para interagir com o computador pode dar oportunidade a pessoas deficientes que queiram trabalhar na área, além de ajudar a suprir a demanda do mercado.

Comandos por voz estão ficando mais e mais comuns em várias áreas da tecnologia, desde smartphones até Smart TVs. No smartphone, por exemplo, é possível fazer ligações, enviar mensagens, definir alarmes, e até mesmo perguntas genéricas (GOOGLE, 2017a). Contudo, mesmo com essa evolução no reconhecimento de fala, nem todas as tarefas executadas nos dispositivos do dia-a-dia são produtivas sem o uso do teclado e mouse, como é o caso da manipulação de código-fonte.

Utilizar o teclado e mouse por muitas horas e com frequência pode desenvolver várias doenças cognitivas (GERR, 2002, tradução nossa). Com o objetivo

de diminuir a incidência desse tipo de doença, especificamente em programadores, e ajudar na inclusão de pessoas com deficiências físicas, este trabalho propõe o desenvolvimento de um sistema de programação por voz, onde o usuário poderá escrever códigos apenas com a fala, sem a necessidade de usar as mãos, focando principalmente na produtividade.

1.4 ESTRUTURA DO TRABALHO

Este trabalho é composto por seis capítulos, onde este primeiro deles trata da introdução, objetivos e justificativa.

O capítulo número dois trata da acessibilidade digital e sua importância para a sociedade por ajudar na inclusão digital de pessoas com necessidades especiais, e, em seguida, comentando sobre a relação entre o uso de computadores e outros dispositivos a distúrbios osteomusculares.

O terceiro capítulo aborda a história do reconhecimento de fala, tratando desde as primeiras implementações até os sistemas presentes. O capítulo também apresenta algumas bibliotecas para implementação disponíveis no mercado e comenta sobre suas diferenças.

Em seguida, o capítulo quatro fala sobre o GNU Emacs, um editor de texto extensivo que também funciona como uma plataforma de desenvolvimento, falando também sobre a possibilidade de se desenvolver um ambiente de programação por fala integrado a este editor.

Os trabalhos correlatos são descritos no capítulo cinco, identificando trabalhos similares que buscam ajudar na inclusão digital de pessoas com necessidades especiais.

O capítulo seis trata do desenvolvimento do projeto, expondo detalhes de sua estrutura e questões técnicas, abordando também a gramática de comandos e o porquê de determinadas escolhas.

Por fim, o sétimo e último capítulo conclui este trabalho e apresenta sugestões para trabalhos futuros.

2 ACESSIBILIDADE DIGITAL

Segundo o censo realizado em 2010 pelo Instituto Brasileiro de Geografia e Estatística (IBGE), 23.9% da população brasileira declarou ter algum tipo de deficiência. As deficiências mais comuns foram visuais e motoras, representando, respectivamente, 18.8% e 7% da população (BRASIL, 2012). Estes dados reforçam a necessidade de utilizar a tecnologia para facilitar a vida dessas pessoas, e também de projetar os dispositivos eletrônicos e seus respectivos *softwares* para que sejam acessíveis a maior parcela possível da população.

Há muitas décadas engenheiros da computação tem trabalhado para colocar a tecnologia a disposição de pessoas com deficiência. O cientista da computação Raymond Kurzweil, em 1976, buscando ajudar pessoas com deficiência visual, desenvolveu uma máquina capaz de reconhecer o conteúdo de documentos impressos e lê-lo. Em 1982, a empresa de Kurzweil desenvolveu um dispositivo capaz de transformar voz em texto, o que permitia que pessoas pudessem escrever textos sem utilizar as mãos (PETRICK, 2015, tradução nossa).

Atualmente, o Windows, sistema operacional da Microsoft, oferece vários recursos de acessibilidade, tais como: leitor de tela, que lê em voz alta o conteúdo exibido; ferramenta de lupa, aumentando o tamanho de uma região específica; comandos de voz; entre outros (MICROSOFT, 2017, tradução nossa). O Android, sistema operacional da Google, possui recursos como o TalkBack, que descreve o conteúdo da tela, alertas e as ações disponíveis (GOOGLE, 2017a, tradução nossa).

Softwares de reconhecimento de fala ainda são incomuns, porém necessários. Poucas opções para escrever códigos de programação utilizando reconhecimento de fala podem ser encontradas atualmente. Uma delas está sendo desenvolvida na Universidade de Porto Rico, e permite controlar *Integrated Development Environments* (IDE) e editores como Netbeans, Sublime, e Gedit com comandos de voz, possibilitando que pessoas com limitações físicas possam interagir com as essas ferramentas (FONTANEZ; FRANCO, 2014, tradução nossa).

2.1 LESÕES POR ESFORÇO REPETITIVO E DISTÚRBIOS OSTEOMUSCULARES RELACIONADAS AO TRABALHO

Lesões por esforço repetitivo e Distúrbios Osteomusculares Relacionadas ao Trabalho (DORT) são termos abrangentes utilizados para se referir a distúrbios que podem afetar diferentes partes do corpo associadas com movimento, como ombros, pulsos, e costas, podendo atingir tendões, músculos, juntas, nervos, e o sistema vascular. Tendinite, tenossinovite, bursite, e síndrome do túnel carpal são algumas das doenças mais comuns resultantes destes distúrbios (SIMONEAU; ST-VINCENT; CHICOINE, 1996, tradução nossa).

Estes distúrbios não são novos; existem menções de séculos atrás sobre problemas causados por movimentos irregulares e violentos e por posturas não naturais para o corpo. Entretanto, os casos de LER e DORT vêm aumentando de forma preocupante nas últimas décadas (YASSI, 1997, tradução nossa).

LER e DORT começaram a se tornar um problema de saúde pública a partir da segunda metade do século XX com os novos postos de trabalho criados graças a evolução industrial e tecnológica. Nos dias de hoje, estes distúrbios atingem várias áreas profissionais (BRASIL, 2012) e, segundo a Pesquisa Nacional de Saúde (2013), milhões de brasileiros possuem algum tipo de LER ou DORT.

2.2 LESÕES POR ESFORÇO REPETITIVO E COMPUTADORES

Esta seção analisa a relação entre LER e computadores e outros dispositivos, discutindo também formas de prevenção e tratamento.

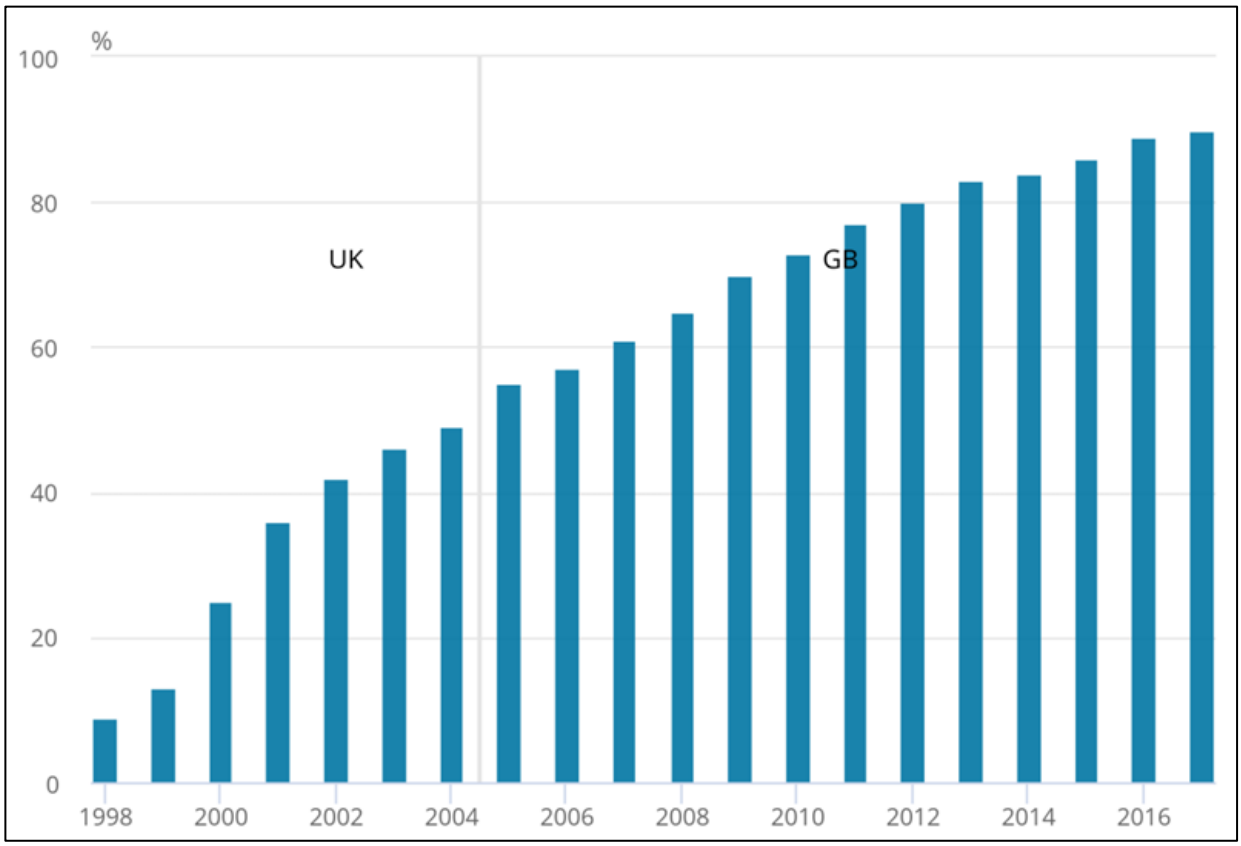
2.2.1 Crescimento do uso de computadores

Nos Estados Unidos, a profissão de desenvolvedor de software está entre as que mais crescem, com um aumento previsto de 30% entre 2016 e 2026, bem acima da média de 7%. Outras profissões que dependem do uso de computador também

aparecem próximo ao topo deste ranking, como analista de segurança da informação, com uma previsão de crescimento de 28% (EUA, 2017, tradução nossa).

O uso de computadores tem crescido bastante nas últimas décadas. Em 1998, no Reino Unido, cerca de 10% da população possuía acesso à internet em casa. Em 2017, a estimativa subiu para 90% da população britânica (REINO UNIDO, 2017, tradução nossa).

Figura 1 – Casas com acesso à internet, de 1998 a 2017, Reino Unido e Grã-Bretanha



Fonte: Reino Unido (2017, tradução nossa)

2.2.2 Relação entre o uso de computador e lesões por esforço repetitivo

A partir dos dados apresentados anteriormente, é pertinente ficar atento aos possíveis males que o uso de periféricos pode trazer. Após ter seu uso popularizado, computadores vêm causando distúrbios de natureza musculoesquelética em muitos de

seus usuários (SIMONEAU; ST-VINCENT; CHICOINE, 1996, tradução nossa), podendo ocorrer em várias regiões no corpo, como mãos, braços, ombros e pescoço (GERR, 2002, tradução nossa).

Um estudo de Blatter e Bongers, 2002, tradução nossa, buscou analisar a relação entre a incidência de LER e o uso prolongado de teclado e mouse. Foi identificado que quanto mais tempo se passa utilizando estes periféricos, maior é a probabilidade do surgimento de alguma lesão. Neste estudo, foi analisada uma população de 516 secretários e digitadores, e aproximadamente 31% deles desenvolveu algum tipo de distúrbio musculoesquelético.

Crianças que passam muito tempo em computadores ou videogames frequentemente sentem algum tipo de desconforto e podem, possivelmente, desenvolver algum tipo de lesão. Para diminuir esse risco, é importante que as crianças adotem posturas adequadas, intervalos frequentes, entre outros hábitos preventivos (RAMOS; JAMES; BEAR-LEHMAN, 2005, tradução nossa).

Estes dados apontam para uma ligação direta entre o uso de computador e outros dispositivos à LER e DORT. E, somando isso a popularização desses dispositivos, torna-se cada vez mais importante pensar em novas formas de interação com dispositivos digitais. Reduzir o uso de teclado e mouse com tecnologias como o reconhecimento de fala, por exemplo, pode ser uma alternativa ao uso do teclado e mouse, evitando possíveis lesões.

2.3 PREVENÇÃO E TRATAMENTO

A partir dessas informações apresentadas, conclui-se que é importante estudar e analisar as causas de LER e DORT e como podem ser prevenidas e tratadas. Existem vários estudos que tratam essas questões de diferentes formas.

Simoneau, Marklin e Berman (2003) concluíram que posicionar o teclado num ângulo negativo pode reduzir a pressão no túnel carpal, reduzindo também o risco de lesões.

O posicionamento do monitores, mesas e cadeiras também são importantes para prevenir o surgimento de lesões nas costas e pescoço (IDOWU; ADEDOYIN; ADAGUNODO, 2005, tradução nossa).

Existem vários exercícios que podem ser realizados com objetivo de reduzir o risco de desenvolvimento de LER e DORT. Os exercícios podem ser feitos para várias partes do corpo, incluindo pescoço, ombros, cotovelos, costas e quadril, e joelhos. Porém, alguns desses exercícios podem não ser recomendados para quem já possui algum tipo de distúrbio musculoesquelético (LEE et al., 1992, tradução nossa).

Para o tratamento destes distúrbios, existem vários métodos. É possível encontrar estudos que tratam LER e DORT com fisioterapia, exercícios, terapia comportamental, abordagem ergonômica, entre outros. A eficácia destes métodos varia; enquanto alguns se mostram ineficazes, outros apresentam melhoras significativas nos pacientes. Contudo, ainda é necessário realizar mais estudos buscando novas formas de tratamento e identificar quais são as mais eficazes, visto que muitos destes estudos são de qualidade questionável, com algumas falhas de metodologia (KONIJNENBERG et al., 2001, tradução nossa).

Além de exercícios, outras formas de interação que não envolvem teclado e mouse também podem ser uma alternativa para auxiliar pessoas com deficiência e também na prevenção de lesões. Um exemplo pode ser encontrado no Windows, onde a Microsoft desenvolveu um sistema para controlar seu sistema operacional apenas com voz. É possível clicar em botões, escrever textos, abrir o menu iniciar, abrir e navegar pelos programas abertos, entre outros recursos, sem utilizar mouse ou teclado. Dessa forma, pode-se prevenir o surgimento de LER e DORT ao evitar o uso do mouse e teclado. Porém, esse recurso é exclusivo para o Windows e de uso genérico, não sendo focado para tarefas mais específicas, como por exemplo, escrever códigos de programação (MICROSOFT, 2016).

3 RECONHECIMENTO DE FALA

A comunicação oral tem sido, desde a pré-história, a principal forma de troca de informações entre as pessoas, e passou a ser explorada pela tecnologia ao serem desenvolvidos dispositivos como telefone, rádio, TV e outros (HUANG; ACERO; HON, 2001, tradução nossa).

O reconhecimento de fala também vem sendo estudado há décadas, e depende de um profundo estudo linguístico e de como a fala humana é produzida. Seu objetivo é fazer com que máquinas sejam capazes de reconhecer a fala humana da forma mais natural possível, beneficiando várias áreas ocupacionais onde os profissionais podem interagir com o computador ou outra máquina enquanto estão com as mãos ocupadas, além de possibilitar que pessoas que, por algum tipo de deficiência tem dificuldade para utilizar mouse ou teclado, interajam com o computador (WAIBEL; LEE, 1990, tradução nossa).

3.1 PRIMEIROS SISTEMAS DE RECONHECIMENTO DE FALA

Desenvolver Sistemas de Reconhecimento de Fala (SRF) é uma tarefa complexa. Atualmente, não existe um SRF universal capaz de reconhecer a fala de qualquer indivíduo em qualquer circunstância, situação ou ambiente com total precisão. Quando o primeiro SRF foi criado, em 1952, pela Bell Laboratories, os computadores ainda eram bastante primitivos, contando com pouco poder de processamento. Este sistema reconhecia apenas dígitos isolados, e o fazia analisando apenas as vogais dos dígitos, porque o reconhecimento de vogais é mais simples de ser realizado. Em 1973, o sistema Hearsey I, desenvolvido *Carnegie Mellon University*, já utilizava informações semânticas do idioma com o objetivo de reduzir significativamente o número de alternativas baseado em informações semânticas, e, consequentemente, melhorando a precisão do processo de reconhecimento (HUANG; ACERO; HON, 2001, tradução nossa).

Além disso, também existe a classificação em relação a dependência da voz do usuário. Alguns SRVs, principalmente os primeiros criados, são desenvolvidos para

funcionar exclusivamente com a voz de um único usuário, diminuindo a complexidade do desenvolvimento e, portanto, o custo. Sistemas dependentes de usuário também costumam ter uma precisão mais alta que um sistema independente de usuário (HUNT, 1993, tradução nossa).

Na década de 1980, houve uma mudança de paradigma nos algoritmos de reconhecimento de fala, saindo de uma abordagem mais direta e simples para um framework baseado em estatísticas, geralmente baseado no *Hidden Markov Model* (HMM), método que ainda é utilizado por SRFs nos dias de hoje (FREITAS; CHEN; JOKINEN, 2010, tradução nossa).

No início dos anos 1990, alguns sistemas foram criados com o objetivo de imitar a comunicação humana. Estes sistemas não apenas reconheciam a voz do interlocutor, como também respondiam, simulando uma conversa. Um desses sistemas foi o Pegasus, desenvolvido pela Massachusetts Institute of Technology, que dá informações sobre voos através de uma linha telefônica (JUANG; LAWRENCE, 2005, tradução nossa).

Em 2006 a Microsoft lançou o Windows Vista com um SRF instalado por padrão, que permitia não apenas ditar textos, mas como também controlar o computador com a voz, possibilitando o acesso a menus e botões, gerenciamento dos programas, entre outras funções (MICROSOFT, 2007).

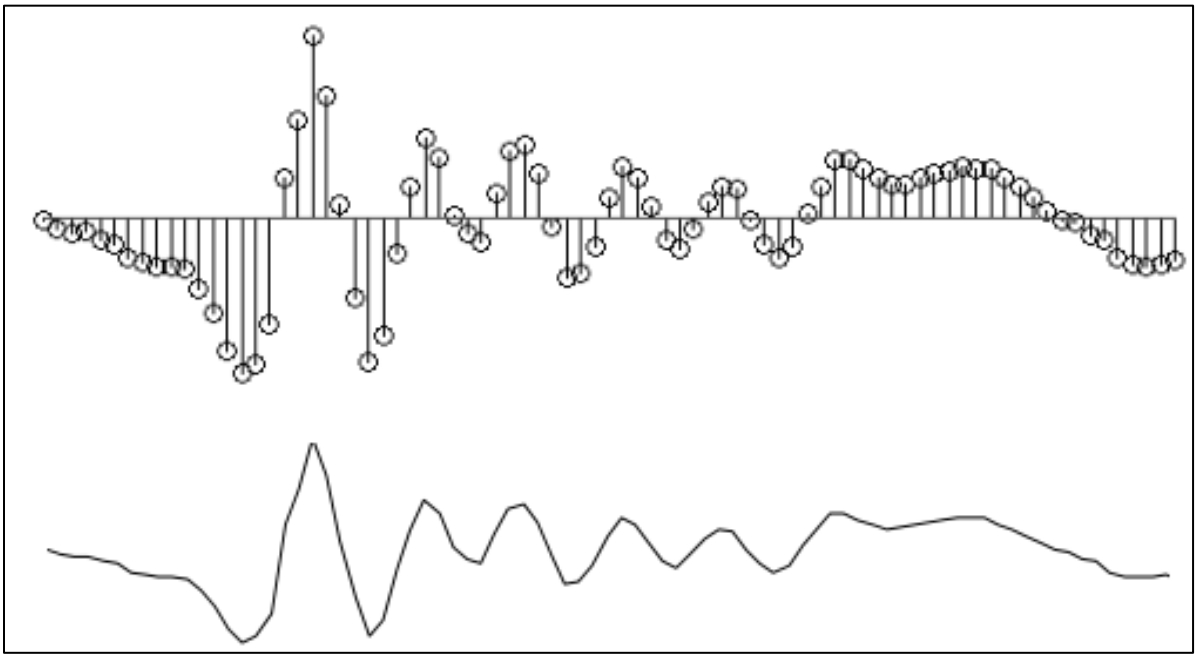
3.2 MÁQUINA DE RECONHECIMENTO DE FALA

Resumidamente, o funcionamento da maioria dos SRVs consiste em isolar os fonemas, que são a menor unidade sonora de um idioma, e, a partir do resultado, tentar organizá-los conforme as possibilidades do idioma sendo trabalhado, buscando, em seguida, a palavra correspondente no banco de dados, levando também em consideração a semântica, caso não seja uma palavra isolada. Portanto, as palavras montadas pelo SRF são na verdade uma sequência de fonemas que correspondem a pronuncia da mesma. Porém, existem vários métodos e algoritmos para executar esse processo (HUANG; ACERO; HON, 2001, tradução nossa).

Os fonemas podem ser classificados em dois tipos: vogais e consoantes, e eles podem variar mesmo dentro do próprio idioma, visto que diferentes dialetos de um mesmo idioma podem ter fonemas diferentes, o que pode tornar ainda mais complexo o reconhecimento. (KENYON; KNOTT, 1949, tradução nossa).

O reconhecimento de padrões é um dos principais pilares do reconhecimento de fala. A fala é composta de padrões de som analógicos que podem ser divididos em fonemas, sílabas e palavras. Por ser um sinal analógico, após a captura da fala, a conversão e processamento do sinal de analógico para digital são fundamentais para o processo de reconhecimento, e através das variações do sinal digital os fonemas são identificados. A Figura 2 ilustra um sinal digital analógico (embaixo) e um sinal digital (em cima) após a conversão (HUANG; ACERO; HON, 2001, tradução nossa).

Figura 2 – Sinal analógico e seu sinal digital correspondente



Fonte: Huang, Acero e Hon (2001).

No SRF Pocketsphinx, por exemplo, o processo de conversão de fala para texto funciona da seguinte forma: após a captura do áudio através de um microfone, o mesmo é segmentado em quadros, e, então, cada quadro é analisado através da Transformada de Fourier -- método que transforma o sinal analógico buscando eliminar

variáveis que não estão relacionadas com a voz, como riso, respiração e outros. Em seguida, utiliza-se o método HMM para analisar os espectros de frequência gerados através da Transformada de Fourier e buscar quais são as possíveis combinações dentro do idioma que está sendo trabalhado, e qual delas é a mais plausível (HUGGINS-DAINES et al., 2006, tradução nossa).

3.3 APIS DE RECONHECIMENTO DE FALA

Existem várias *Application Programming Interfaces* (API) para implementação de reconhecimento de fala, permitindo que desenvolvedores criem programas e soluções utilizando essa tecnologia. Algumas dessas API's são:

a) **CMU Sphinx:** desenvolvido pela Carnegie Mellon University, este projeto busca oferecer uma *engine* de reconhecimento de fala com baixo consumo de recursos de hardware; arquitetura flexível que permite a adição de novos idiomas; funciona *off-line*; gratuito e de código aberto, permitindo uso tanto para objetivo pessoal quanto para comercial (CMU Sphinx, 2009);

b) **Google Cloud Speech API:** API de reconhecimento de fala da Google que oferece suporte a mais de 80 idiomas e variantes. Está disponível apenas para uso através na nuvem e não é gratuito (GOOGLE, 2017b);

c) **Dragon NaturallySpeaking:** ambiente de reconhecimento de fala disponível para Windows e OSX. É oferecido em vários idiomas e possui suporte para desenvolvedores. É necessário comprar uma licença para poder utilizar (NUANCE, 2017);

d) **Watson:** assim como o Google Cloud Speech, o Watson é uma API da IBM na nuvem para reconhecimento de fala. Possui suporte para 8 idiomas, incluindo português brasileiro. Possui uma versão gratuita com limitações (IBM, 2017).

Através destas ferramentas que possuem uma API, é possível desenvolver aplicações que utilizem reconhecimento de fala como entrada de dados. Porém, é importante atentar-se às limitações de cada API. Apenas o Dragon NaturallySpeaking e

o CMU Sphinx funcionam sem conexão com a internet, e, somente este último é gratuito e código aberto.

Além disso, apenas o CMU Sphinx permite que novos idiomas sejam criados pelo usuário, através de modelos acústicos, chamados *acoustic models*. Esses modelos acústicos são um conjunto de dados de áudio que o sistema utiliza para processar o reconhecimento de fala. Dessa forma, é possível criar modelos de idiomas não disponibilizados por padrão, ou até mesmo otimizar os modelos existentes para uma melhor precisão (CMU SPHINX, 2017).

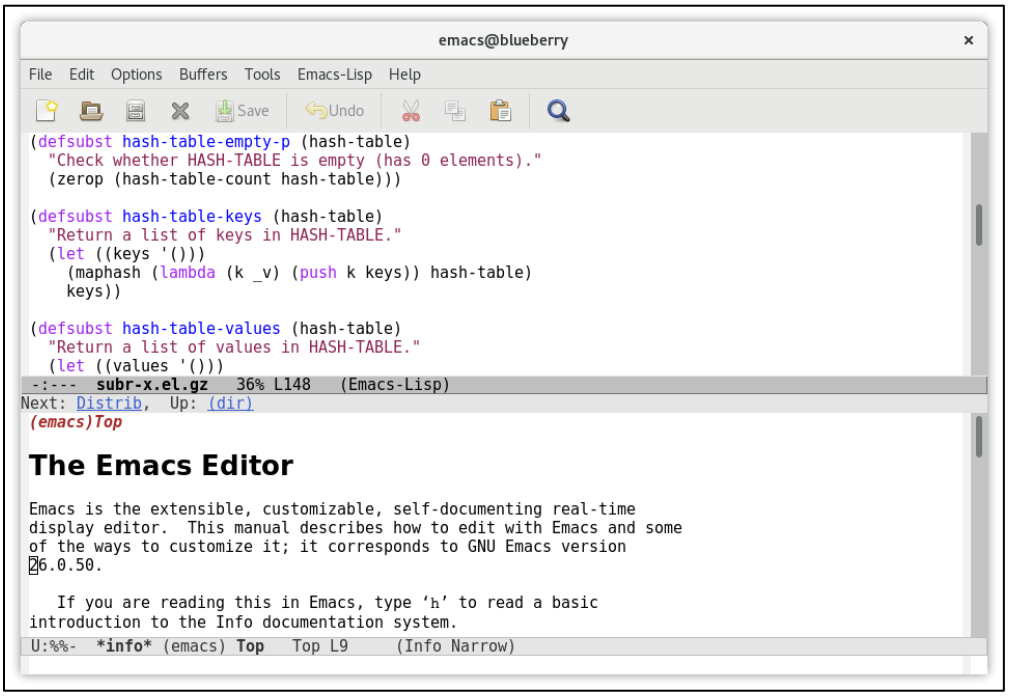
Algumas APIs, como o Google Speech Cloud API, fazem uso do *Machine Learning*, onde o sistema utiliza algoritmos de redes neurais detectar padrões nas falas recebidas e melhorar a precisão (GOOGLE, 2017).

Existe uma biblioteca para Python, chamada SpeechRecognition, que suporta várias API's, como CMU Sphinx, Google Cloud Speech API, Wit.ai, Houndify, entre outras. Dessa forma, pode-se desenvolver aplicações onde é possível escolher a API que será utilizada, sem ser necessário implementar cada API individualmente (ZHANG, 2017, tradução nossa).

4 GNU Emacs

Inicialmente criado em 1974 nos laboratórios de inteligência artificial do *Massachusetts Institute of Technology* (MIT) por Richard Stallman, o GNU Emacs é um editor de texto que tem a extensibilidade como principal característica (STALLMAN, 1982, tradução nossa).

Figura 3 – Editor de texto GNU Emacs



Fonte: GNU Emacs (2017)

4.1 EMACS LISP

O editor possui um núcleo escrito em C que implementa uma linguagem de programação própria baseada em Lisp chamada Emacs Lisp. Algumas das funções básicas de edição também são escritas em C, porém maior parte do código-fonte do editor é escrito em Emacs Lisp (GNU EMACS, 2017).

É possível desenvolver novos recursos, bibliotecas, suporte para linguagens, auxílios de edição, recursos de acessibilidade, entre outros, para o editor utilizando o

Emacs Lisp, que são chamados de pacotes. Como a maior parte do GNU Emacs também e escrita em Emacs Lisp, isso significa essas extensões possuem a mesma linguagem que o editor é desenvolvido, proporcionando um maior poder extensibilidade (CAMERON, 2009, tradução nossa).

Qualquer ação executada no GNU Emacs está ligada a uma rotina de programação (em C ou Emacs Lisp) que pode ser chamada de comando. Quando uma tecla do teclado é pressionada, o editor verifica qual comando que está mapeado a essa tecla e o executa. Todo comando pode ser chamado manualmente ou mapeado a uma tecla ou atalho do teclado. Existe um comando, por exemplo, para mover o cursor para a direita, que, por padrão, está mapeado na seta para a direita do teclado. Ou seja, quando a tecla para direita é pressionada, o GNU Emacs apenas executa o comando mapeado a essa tecla, que, se não foi alterado, move o cursor para a direita. Isso oferece uma flexibilidade que pode ser explorada por pacotes desenvolvidos por terceiros ou pelo próprio usuário, que podem facilmente invocar os comandos nativos do editor ou de outros pacotes instalados, além de configurar as ações do teclado como for conveniente (STALLMAN et al., 2017, tradução nossa).

4.2 CARACTERÍSTICAS

Este editor possui algumas peculiaridades. Apesar de permitir trabalhar com vários arquivos de forma simultânea, o GNU Emacs não possui abas, e sim com um sistema de *buffers*. Cada *buffer* pode ou não estar associado a um arquivo e pode-se ter vários *buffers* abertos ao mesmo tempo. Ao fechar o editor, caso um *buffer* não esteja associado a um arquivo, por padrão o seu conteúdo será perdido (STALLMAN et al., 2017, tradução nossa).

Outra característica sua é a auto-documentação. O GNU Emacs possui uma documentação interna sobre seus próprios recursos e atalhos de teclado. É possível, por exemplo, mostrar o que determinado atalho do teclado faz e a qual comando ele está apontando, ou também exibir uma explicação o que o comando faz. Mesmo os recursos criados por terceiros podem ser documentados dentro do próprio editor (STALLMAN et al., 2017, tradução nossa).

Por si só, o GNU Emacs não possui um foco específico em relação a acessibilidade, porém, sua característica de extensibilidade pode proporcionar uma plataforma onde o objetivo deste trabalho possa ser desenvolvido. Entretanto, no momento da escrita deste, não foi encontrado um trabalho acadêmico com esse mesmo objetivo.

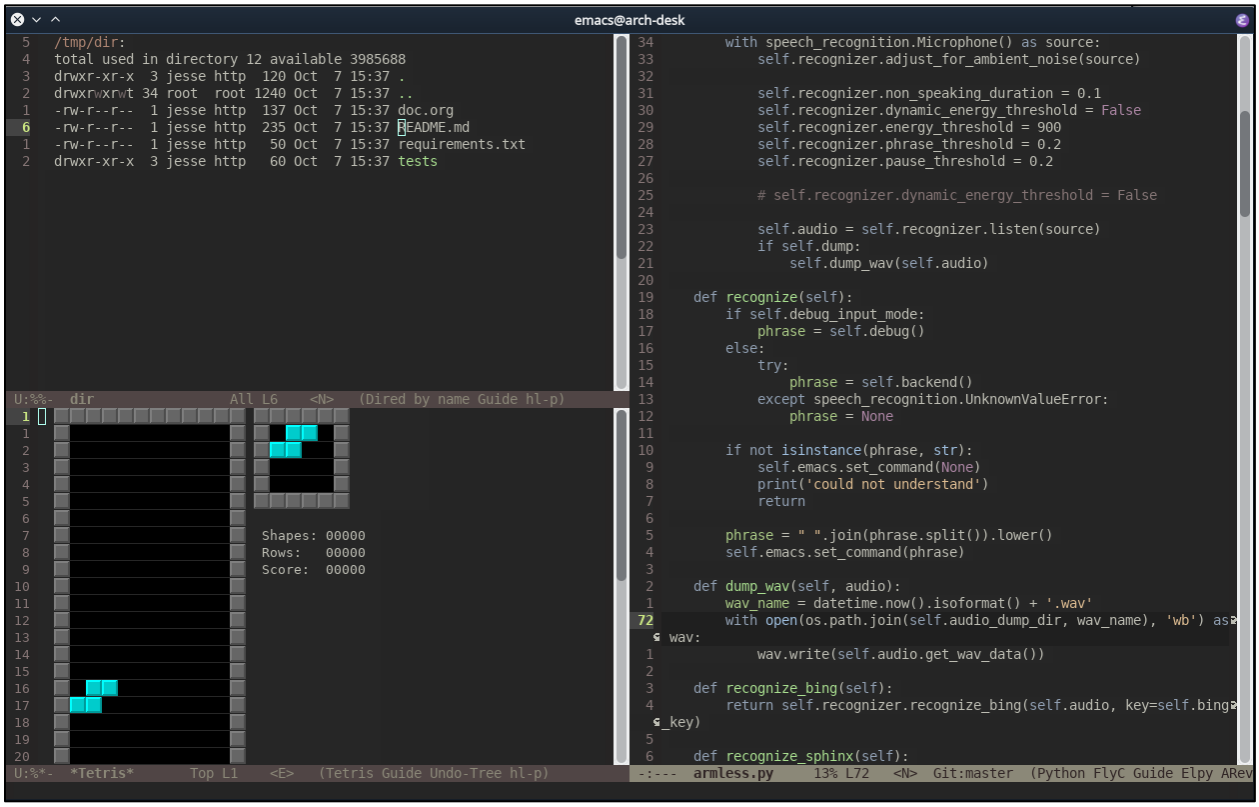
4.2.1 Extensibilidade

Por padrão, o GNU Emacs possui vários recursos que vão além de edição de texto. Suas funções são divididas em pacotes. A versão 25.2, lançada em abril de 2017, possui 81 pacotes, que vão de um terminal de comando embutido até um navegador de diretórios e visualizador de PDF (GNU EMACS, 2017).

Existem repositórios criados por usuários que oferecem pacotes não oficiais, permitindo que qualquer pessoa possa distribuir suas modificações. O repositório MELPA, por exemplo, possui um total de mais de 3 mil pacotes e mais de 50 milhões de downloads (MELPA, 2017).

Por ter sido desenvolvido desde o início com extensibilidade em mente, o GNU Emacs oferece uma plataforma para desenvolvimento que vai além de ferramentas para edição de texto. Pode-se, por exemplo, utilizar o GNU Emacs para ler e enviar e-mails, e inclusive jogar jogos como Tetris e Snake (STALLMAN et al., 2017).

Figura 4 – Recursos do GNU Emacs



Fonte: Do autor.

Os pacotes desenvolvidos com o GNU Emacs, quando não são bibliotecas ou funções manuais, costumam ser classificados em dois tipos (CAMERON, 2009):

a) ***major mode:*** modo que determina o comportamento de edição em relação ao buffer/arquivo aberto. Ao editar um arquivo de uma linguagem de programação, por exemplo, o *major mode* pode oferecer recursos de edição dedicados a linguagem em questão, como coloração da sintaxe, indentação específica para a linguagem, compilação e execução do código, e outras funções específicas. Cada buffer pode ter apenas um *major mode* ativo. Alguns exemplos são: *java-mode*, *c-mode*, *dired-mode*, *fundamental-mode, python-mode*;

b) ***minor mode:*** modo que oferece recursos de edição opcionais ao GNU Emacs complementares ao *major mode*. Podem ser ativados e desativados pelo usuário a qualquer instante, além de poderem também ser

configurados para serem ativados por padrão quando um *major mode* for iniciado. Alguns exemplos de *minor mode* são: *flyspell*, que destaca palavras escritas com erros ortográficos, oferecendo alternativas com possíveis correções; *linum*, que exibe os números das linhas do buffer; *auto save*, que salva o *buffer* periodicamente de forma automática, evitando perder alterações não salvas.

Essa arquitetura voltada para a extensibilidade torna o GNU Emacs não apenas um editor, mas também uma plataforma de desenvolvimento baseada em Lisp. Dessa forma, é possível criar basicamente qualquer tipo de aplicação, desde criar um jogo dentro do editor até integrá-lo com uma API de reconhecimento de fala.

4.3 COMPATIBILIDADE E ACESSIBILIDADE

O GNU Emacs suporta uma grande variedade de caracteres internacionais, incluindo variantes europeias e vietnamita, assim como caracteres arábicos, hindus, gregos, japoneses, chineses, e vários outros (STALLMAN et al., 2017).

Utilizar este editor dispensa o uso do mouse; todos os comandos e funções podem ser executados exclusivamente através do teclado. Isso evita o frequente movimento dos braços do teclado até o mouse e vice-versa, contribuindo para que pessoas que tenham dificuldade para utilizar o mouse se sintam mais confortáveis (CAMERON, 2009).

O GNU Emacs é compatível com uma grande variedade de plataformas, que incluem os principais sistemas operacionais: Windows, GNU/Linux e Mac OS. O editor também possui suporte para várias linguagens de programação e marcação. Sua extensibilidade também permite que, mesmo quando o suporte para uma linguagem não está disponível por padrão, possa-se desenvolver uma extensão que ofereça os recursos necessários para essa nova linguagem. Desta forma, o GNU Emacs torna-se uma ótima opção para quem precisa de um editor híbrido que não se prenda a apenas algumas linguagens (STALLMAN et al., 2017).

5 TRABALHOS CORRELATOS

Este capítulo aborda trabalhos que possuem alguma similaridade a este, seja relacionada ao tema ou ao objetivo.

5.1 APRIMORANDO INTERFACES DE PROGRAMAÇÃO PARA PESSOAS COM MOBILIDADE LIMITADA ATRAVÉS DE RECONHECIMENTO DE FALA

Neste artigo publicado na Universidade de Porto Rico, Fontanez e Franco (2014, tradução nossa) buscam, através do reconhecimento de fala, auxiliar pessoas com algum tipo de limitação de mobilidade possam programar.

No projeto, foi utilizado o programa de reconhecimento de fala de código aberto Simon, que, diferente de sistemas de reconhecimento de fala comerciais, permite criar modelos acústicos manualmente. Simon precisa ser configurado com cenários de uso, e, então, aguarda o usuário falar as palavras e frases que foram informadas e executa a ação correspondente.

No caso deste trabalho, foram criados cenários de uso, com vocabulário limitado, para controlar IDE’s e editores de texto, além de possuir modelos acústicos variados para cada linguagem trabalhada. Entretanto, existem muitos detalhes específicos que precisam ser resolvidos, e os autores concluem que a melhor solução seria criar um ambiente de programação totalmente novo, além de sugerirem a criação de uma linguagem de programação falada que seja acessível para a maioria dos programadores (FONTANEZ; FRANCO, 2014, tradução nossa).

5.2 MODELO DE ACESSIBILIDADE EM TELECENTROS

Este trabalho, de Miranda (2007), publicado na Universidade Federal de Santa Catarina (UFSC), propõe um modelo de acessibilidade para atendimento em TELECENTROS, buscando ajudar pessoas com necessidades especiais a interagir com o computador.

Esse modelo considera deficiências visuais, auditivas, físicas, motoras, mentais. Os monitores dos TELECENTROS avaliam cada usuário individualmente através de dez etapas definidas no modelo, possuindo equipamentos especiais para os diferentes tipos de deficiência, como teclados e mouses adaptados, monitores reestruturados com cores e adesivos, fones de ouvido, softwares específicos, e outros.

Durante o trabalho, percebeu-se um confronto entre acessibilidade e a interação humano-computador, reforçando a importância de se pensar na inclusão digital de pessoas com necessidades especiais, onde milhões de pessoas acabam sendo privadas do acesso à informação.

A autora sugere futuros trabalhos para aprimorar o modelo tanto para TELECENTROS quanto para outros tipos de iniciativas de inclusão digital (MIRANDA, 2007).

5.3 PROGRAMANDO POR VOZ: UMA ABORDAGEM SEM A NECESSIDADE DE MÃOS PARA CRIANÇAS COM PROBLEMAS MOTORES

Wagner et al. (2012) desenvolveram uma ferramenta chamada Myna para controlar o Scratch (programa utilizado para aprender programação) através de comando de voz. O Scratch, que permite organizar blocos de forma lógica, como uma linguagem visual, pode ser controlado apenas por teclado e mouse de forma nativa.

Qualquer interface de comandos de voz possui o desafio de “imitar” as ações do teclado e mouse, evitando aumentar a dificuldade de interação. E esse desafio se torna ainda maior quando a tarefa é controlar programas que foram feitos serem controlados exclusivamente por teclado e mouse.

O Myna, que roda em paralelo ao Scratch, fica aguardando um comando de voz do usuário. Quando o usuário fala um comando, o Myna processa, interpreta de acordo com uma gramática pré-definida, e simula ações de teclado e mouse dentro do Scratch.

Esse modelo de interação permitiu que pessoas com limitações físicas (principalmente crianças, que são o foco do trabalho) pudessem aprender a programar utilizando o Scratch.

Para futuros estudos, os autores planejam coletar dados de usabilidade e performance da aplicação, realizar testes com usuários e verificar se todos os *bugs* estão corrigidos, além de assegurar que os comandos de voz são intuitivos (WAGNER *et al*., 2012, tradução nossa).

6 AMBIENTE DE RECONHECIMENTO DE FALA PARA ESCREVER CÓDIGOS DE PROGRAMAÇÃO INTEGRADO AO GNU EMACS

Por meio do conhecimento adquirido durante o desenvolvimento do referencial teórico, foi desenvolvido um protótipo de um ambiente de programação controlado através da fala integrado ao editor de texto GNU Emacs. Foram utilizadas apenas ferramentas gratuitas e de código aberto, permitindo que qualquer pessoa com um computador possa usufruir. Este protótipo permite escrever e editar códigos de programação utilizando apenas comandos de voz, e pode ser dividido em dois módulos: o Módulo de Reconhecimento (MR) e Módulo do GNU Emacs (MGE).

6.1 METODOLOGIA

O processo metodológico deste trabalho constituiu-se das seguintes etapas: levantamento bibliográfico, definição da arquitetura da gramática e ações dos comandos, escolha de linguagens e bibliotecas a serem utilizadas, otimização das palavras da gramática no CMU Sphinx, desenvolvimento dos MR e MGE, testes e análise do projeto final e dos resultados obtidos.

Inicialmente, focou-se no estudo referente as possíveis lesões causadas pelo uso de periféricos como teclado e mouse e da importância de meios alternativos para interação com o computador. Percebeu-se que existe uma carência de meios de interação alternativos para tarefas mais específicas, como é o caso do desenvolvimento de *software*.

Em seguida, foram estudados os SRVs, passando por sua história, detalhes técnicos de suas implementações, e listando alguns dos mais populares SRVs disponíveis atualmente. Esta pesquisa bibliográfica contribuiu para um conhecimento mais teórico em relação ao reconhecimento de fala, o que ajudou num melhor entendimento dos SRVs e auxiliou nas configurações acústicas do CMU Sphinx e seleção das palavras da gramática.

Numa terceira etapa, estudou-se o editor de texto GNU Emacs, onde foi possível entender melhor sobre sua história e suas funcionalidades, colaborando no

desenvolvimento das funções auxiliares dentro do editor, bem como a implementação do servidor TCP/IP, responsável pelo envio de comandos para o GNU Emacs através do MR.

O CMU Sphinx foi escolhido para este trabalho por ser o código aberto, gratuito, e funcionar sem a necessidade de uma conexão com a internet, além de ser altamente configurável. Este SRF permite realizar adaptações acústicas com o objetivo de melhorar a precisão. Além disso, também é possível limitar o vocabulário para trabalhar apenas com as palavras definidas, consequentemente melhorando a precisão. Ambos processos foram realizados.

Para o desenvolvimento do MR foi escolhida a linguagem de programação Python, junto com a biblioteca SpeechRecognition, que por padrão oferece integração com várias APIs de reconhecimento, incluindo o CMU Sphinx, que, pelos motivos listados anteriormente, foi a que recebeu o foco neste trabalho.

Alguns comandos de voz precisaram de argumentos para executar ações como escrever uma frase. Ao falar “insert hello world”, a palavra *insert* está ligada a uma rotina de pré-processamento que analisa o comando, identificando o argumento (nesse caso, *hello world*) e, só então, envia para o GNU Emacs o comando em Emacs Lisp “(insert “hello world”)”, que insere a palavra ou frase no *buffer*.

Uma preocupação deste trabalho foi utilizar apenas softwares livres e multiplataforma, com o objetivo de tornar o projeto o mais acessível possível, não se fazendo necessário a aquisição de licenças, que muitas vezes são caras e pouco acessíveis. Desta forma, o único custo necessário para a execução deste projeto é apenas do hardware.

Este trabalho foi desenvolvido utilizando a distribuição GNU/Linux Arch Linux e o GNU Emacs como ambiente de desenvolvimento. Suporte para outros sistemas operacionais não foi testado neste tbalho.

6.2 LINGUAGEM DE PROGRAMAÇÃO ESCOLHIDA

Apesar do desenvolvimento do projeto ser em Python, nada impede que a utilização de outras linguagens para serem manipuladas por este projeto. Esta seção trata da escolha dessa linguagem.

Alguns dos desafios da programação por fala são a manipulação de símbolos e a inserção de códigos de maneira eficiente e produtiva. Linguagens muito verbosas podem tornar as tarefas de edição por fala mais demoradas e trabalhosas. Também é importante que a linguagem escolhida seja interpretada ao invés de compilada para facilitar no processo de execução do código. Por isso, neste trabalho, foram analisadas algumas linguagens interpretadas com sintaxe mais simples para realizar o desenvolvimento.

Inicialmente, pretendia-se utilizar Python, pois é uma linguagem de sintaxe simples, objetiva e de tipagem dinâmica, ou seja, não é necessário informar o tipo de variáveis. Diferente de linguagens como C ou Java, por exemplo, não necessita de, entre outras formalizações, função *main* para executar o código. Porém, pelo fato de a indentação fazer parte da sintaxe, seria preciso que o usuário indentasse manualmente os blocos de código, gerando mais uma tarefa extra que não seria necessária em uma outra uma linguagem sem esta característica.

Figura 5 - Hello world em Java

```java
// Hello world em Java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello, World");
    }

}
```

Fonte: do Autor

A Figura 6 ilustra um *hello world* em Java, uma linguagem mais verbosa, em contraste com um *hello world* em Python na Figura 7, uma linguagem mais simples e objetiva.

Figura 6 - Hello world em Python

Fonte: do autor

Após novas análises, decidiu-se utilizar Ruby, que, assim como Python, também é uma linguagem interpretada e de sintaxe simples e objetiva. Inclusive, estas duas linguagens compartilham algumas semelhanças na sintaxe, como na declaração de funções, onde ambas utilizam a palavra *def*. No Ruby, porém, a indentação é tratada apenas como *whitespace* e não faz parte da sintaxe, eliminando a necessidade de intervenção do usuário na indentação do código.

6.3 GRAMÁTICA DE COMANDOS

A gramática de comandos foi desenvolvida utilizando a língua inglesa, que é o idioma que, na maioria dos SRVs, possui uma qualidade de reconhecimento mais avançada que os demais. Além disso, o inglês está presente na maioria das linguagens de programação, que costumam utilizar termos como *if, for, while, function,* entre outros.

Para obter-se uma melhor precisão, foi utilizado um dicionário com um número limitado de palavras. Dessa forma, ao falar um comando, o CMU Sphinx não precisa compará-lo com todas as palavras da língua inglesa, diminuindo a probabilidade de retornar um falso positivo. A partir deste dicionário, foi criado um banco de dados em *sqlite* com, até então, 68 comandos para edição, navegação, inserção e execução de códigos, entre outras tarefas.

Neste banco de dados são armazenadas as seguintes informações:

a) ***id*:** identificação única do registro na tabela;

b) ***phrase*:** comando de voz que deve ser falado para executar a ação correspondente;
c) ***command*:** ação a ser executada, que pode ser enviada direto para o GNU Emacs ou processada antes, para formatar argumentos ou realizar algum outro tratamento. No segundo caso, será chamado a função armazenada neste campo, que está presente no MR, passando como argumento o comando falado;
d) ***callback*:** flag para diferenciar comandos que devem ser executados direto no GNU Emacs ou se devem ser processados pelo MR antes;
e) ***edit*:** flag para identificar comandos de edição.

O Sistema de Gerenciamento de Banco de Dados (SGDB) utilizado foi o *sqlite*. Este SGDB não necessita de processos externos, dependendo apenas de um arquivo contendo o banco de dados. Além disso, pode ser instalado como uma biblioteca do Python, sem precisar baixar ou instalar um programa externo.

6.3.1 Comandos

Esta seção descreve os principais comandos e suas ações. A lista completa encontra-se no apêndice A.

6.3.1.1 Repeat

Para não ser necessário ficar repetindo um mesmo comando várias vezes, foi desenvolvido o comando *repeat*, que repete o último comando. Este comando pode ser seguido de um argumento numérico, indicando quantas vezes deseja-se repetir o último comando.

6.3.1.2 Navegação

O projeto possui vários comandos para navegação. Pode-se subir ou descer a linha (*line up* e *line down),* ir para o início ou final da linha (*line start* e *line end*), ir para

próxima palavra ou palavra anterior (*word next* e *word previous*), ir para o início e final do arquivo (*buffer start* e *buffer end*), entre outros.

6.3.1.3 First

Devido às dificuldades em relação a precisão e ao dicionário reduzido, muitas palavras não poderiam ser escritas diretamente. Pensando nisso, foi criado o comando *first*, que insere apenas a primeira letra da palavra falada. Ao falar *"first dot"*, por exemplo, insere-se a letra *d* (primeira letra da palavra *dot*) no *buffer*. Dessa forma é possível, mesmo que de forma mais lenta, escrever palavras que não estão presentes no dicionário, além de também permitir escrever combinações de letras que não são exatamente palavras.

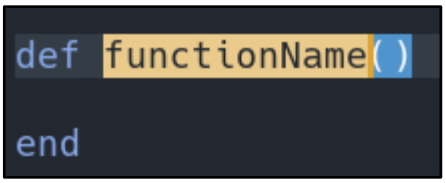
6.3.1.4 Manipulação do texto

Pode-se apagar palavras inteiras ou apenas o caractere à esquerda ou direita (*delete word, delete left, delete right*), apagar do cursor até o final da linha (*kill line*), copiar e colar (*copy e paste*), desfazer e refazer (*undo* e *redo*), etc.

6.3.1.5 Inserção de códigos

Para inserir textos, utiliza-se o comando *insert* seguido das palavras que se deseja escrever. Ao falar *insert enter number one*, será inserido no buffer "enter number one". Existem variações do comando *insert,* como o *quote,* que faz a mesma função, mas colocando as palavras faladas entre aspas. É possível também inserir *snippets* de códigos pré-definidos, como blocos condicionais, blocos de repetição, funções e classes, com os comandos *new if, new while, new function, new class*. Ao criar uma nova função (ilustrado na Figura 7), por exemplo, o cursor será colocado no nome da função que, após informado, pode-se falar *next* para levar o cursor para os argumentos da função, e falar *next* novamente para levar o cursor para o corpo da função.

Figura 7 - *Snippet* de uma função



Fonte: do autor

Também existem comandos para escrever símbolos (*equal* escreve “=” e *plus* escreve “+”, por exemplo) e números (o comando *number* seguido dos números desejados os escreve em forma de símbolos).

6.3.1.6 Seleção de texto

Pode-se iniciar seleção de texto com o comando *selection start*. Com a seleção ativa, pode-se navegar pelo texto normalmente com os comandos de navegação. Quando a seleção de texto estiver ativa, comandos como *delete* e *copy* agirão sobre a região selecionada. Também existe o comando *expand region*, que expande a seleção em níveis semânticos. Ao falar uma vez, seleciona a palavra sob o cursor, falando novamente, seleciona a linha inteira, depois o bloco, e assim por diante.

6.3.1.7 Execução do código

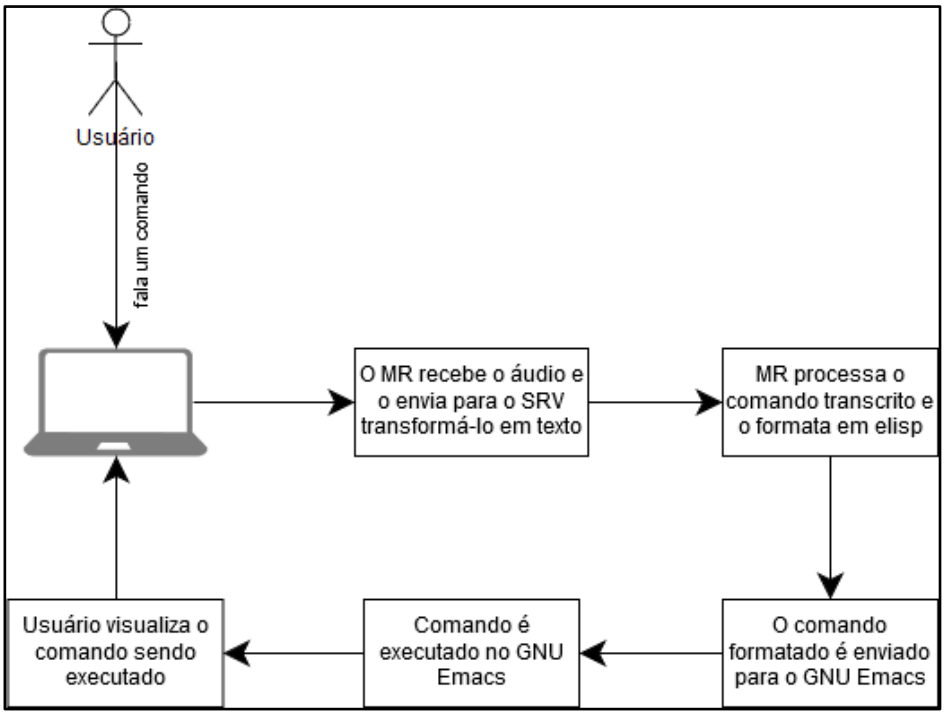
O projeto possui o comando *run* que executa o código e abre um novo *buffer* na mesma janela mostrando o resultado. O comando *switch buffer* alterna para a janela aberta, onde pode-se alimentar o programa com informações do usuário.

6.4 ESTRUTURA DO PROJETO

O projeto consiste em um MR, desenvolvido em Python, que ouve os comandos, os analisa e processa, se necessário, e os envia para o GNU Emacs para ser executado. Também foi desenvolvido um *minor mode* em conjunto com algumas funções auxiliares no GNU Emacs. Internamente, o projeto foi batizado de *armless*.

A Figura 8 apresenta um diagrama exibindo o fluxo dos comandos, mostrando os passos desde de a fala do usuário até a execução dentro do GNU Emacs.

Figura 8 - Diagrama do fluxo de execução



Fonte: do autor

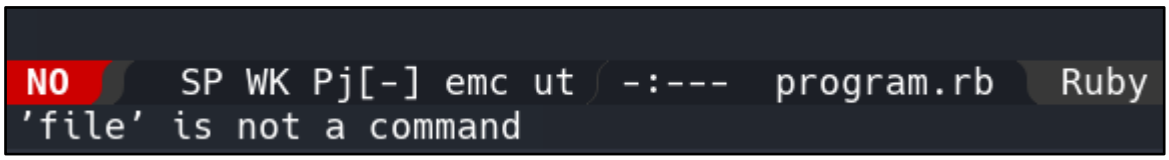
6.4.1 Módulo de reconhecimento

O MR é um programa que fica rodando em *loop,* aguardando comandos do usuário. Quando o usuário fala um comando, o áudio é capturado e enviado para o CMU Sphinx para ser analisado, retornando então a fala convertida em texto.

Após a fala retornada, para identificar o comando o MR analisa as primeiras palavras do comando de voz e procura no banco de dados a ação correspondente. Quando a ação é um *callback*, ou seja, precisa ser processada pelo MR antes de ser enviada para o GNU Emacs, a função armazenada no campo é chamada passando como argumento a fala reconhecida. Os *callbacks* analisam os argumentos e os formatam conforme necessário. Por exemplo, quando é falado o comando *first*, o MR coleta a primeira letra do argumento e formata em um comando *elisp* de inserção, para

só então enviar para o GNU Emacs e ser executado. Quando não é encontrada uma ação correspondente, envia-se uma mensagem dizendo que o comando não existe, como ilustrado na Figura 9.

Figura 9 - Mensagem informando que comando não existe

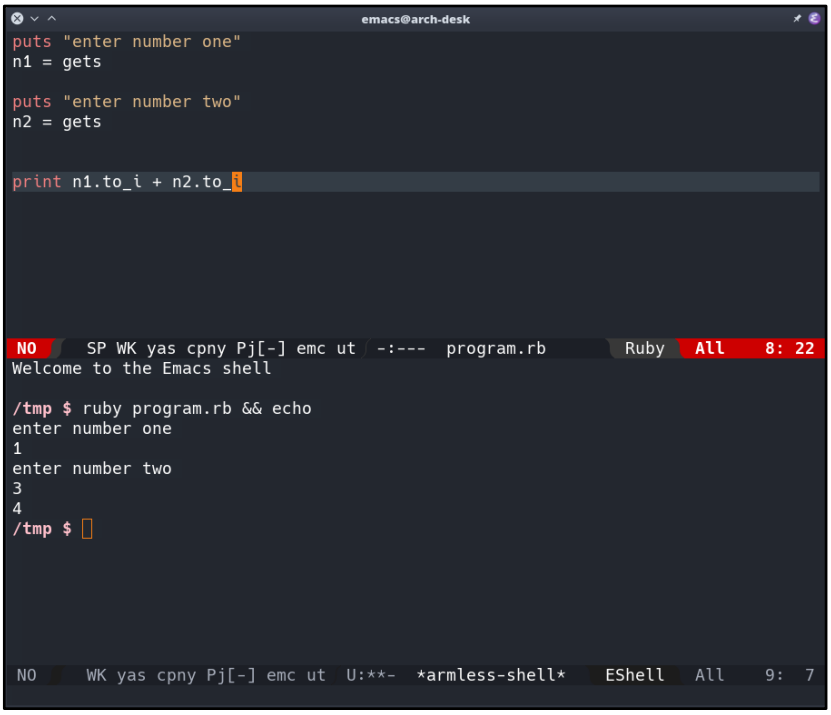


Fonte: do autor

Todo comando, depois de devidamente processado, se transforma em um comando do GNU Emacs. Alguns deles são recursos nativos do editor, outros foram desenvolvidos especialmente para este trabalho.

Além de escrever e editar os códigos, também é possível executá-los. O comando *run* abre um terminal dentro do GNU Emacs exibindo o resultado do código executado. Também é possível alimentar o código com dados do usuário; o comando *switch buffer* foca no terminal aberto onde pode-se interagir, informando dados para o programa. Ao falar o comando *switch buffer* novamente, foca-se de novo no código, como demonstrado na Figura 10.

Figura 10 - Código em Ruby (em cima) e seu resultado (embaixo)



Fonte: do autor

O *buffer* execução de código também funciona como um terminal. Pode-se realizar comandos do sistema operacional e visualizar suas saídas normalmente, como *ls* para listar os arquivos ou *rm* para apagar um arquivo. A interação com o terminal, porém, não foi o foco deste trabalho, que não foi otimizado para esta tarefa.

6.4.2 *Armless mode*

Além do MR, foi também foi desenvolvida uma biblioteca de comandos para o GNU Emacs. Esta biblioteca consistes em funções auxiliares e complementos para executar ações dos comandos que não estão disponíveis de forma nativa no GNU Emacs.

Neste módulo estão presentes ações para copiar e colar, abrir o arquivo para edição, converter letras em maiúsculo ou minúsculo, correções de comportamentos indesejados por estar sendo controlado externamente, inserção de texto, navegar entre os buffers abertos, entre outros. Também possui alguns tratamentos de baseados no

major-mode. Ao estar editando um arquivo, por exemplo, após inserção de texto, é chamado o comando do GNU Emacs *indent-for-tab-command*, que mantém o código indentado. Porém, essa ação não é desejável quando estiver utilizando o terminal. O MGE também é responsável por esse tipo de tratamento.

6.4.3 Comunicação

Após o MR finalizar de processar e formatar um comando, ele deve ser executado no GNU Emacs. Para isso, foi preciso analisar um modelo de comunicação entre o MR e o GNU Emacs pelo qual os comandos possam ser enviados, onde o GNU Emacs atue como um servidor, aguardando os comandos servem enviados, e o MR atue como um *client,* enviando os comandos para o GNU Emacs. Ao falar o comando “line up”, por exemplo, o MR o transformará em um código de Emacs Lisp para então enviar através deste meio de comunicação para o GNU Emacs, onde este comando será executado e visualizado pelo usuário. Dois métodos foram desenvolvidos, que são descritos a seguir.

No primeiro, utiliza-se o comando *server-start*, que é um recurso do GNU Emacss que permite utilizar a mesma instância do editor em um terminal ou em uma nova janela (ou frame, na terminologia do GNU Emacs). Quando o servidor foi iniciado, é possível enviar comandos através do programa *emacsclient* pela linha de comando. Dessa forma, através da linha de comando, o MR envia as ações para serem executadas no GNU Emacs.

O segundo método inicia um servidor Transmission Control Protocol/Internet Protocol (TCP/IP) dentro do próprio GNU Emacs, e foi implementado buscando uma melhor alternativa de comunicação que não envolvesse enviar os comandos através da linha de comando. O servidor é iniciado automaticamente quando o *armless-mode* é ativado, e parado quando este modo é desativado. Quanto servidor está ativo, ele fica aguardando os pacotes contendo os comandos serem enviados para executá-los, permitindo que os comandos sejam enviados a partir de qualquer dispositivo. Isso pode possibilitar utilizar o GNU Emacs em um dispositivo e o MR em outro, se devidamente configurados.

Neste trabalho não foram analisadas as possíveis vulnerabilidades de segurança causadas pelo servidor TCP/IP. É importante que trabalhos futuros façam essa análise, implementando recursos como bloqueio de IPs no servidor do GNU Emacs para permitir a execução de apenas de dispositivos autorizados.

6.5 INSTALAÇÃO

A instalação dos componentes necessários para executar este trabalho foi feita de forma manual. Existem vários componentes que precisam ser instalados e configurados para seu funcionamento. Em resumo, os seguintes passos precisam ser realizados (os comandos descritos devem ser executados na raiz do projeto):

a) Instalação do GNU Emacs;
b) Instalação das configurações e pacotes do GNU Emacs, incluindo o *armless-mode;*
c) Instalação do Python 3.6 ou superior, incluindo *pip*, seu gerenciador de pacotes;
d) Instalação das dependências do projeto com o comando “pip3 install -r requeriments.txt”;
e) Instalação do CMU Sphinx;
f) Executar os comandos “python3 db/db.py --create-schema --populate” e “python3 install_lang.py --install --update-dict” para criar e popular o banco de dados da gramática de comandos e instalar o dicionário de dados do CMU Sphinx;

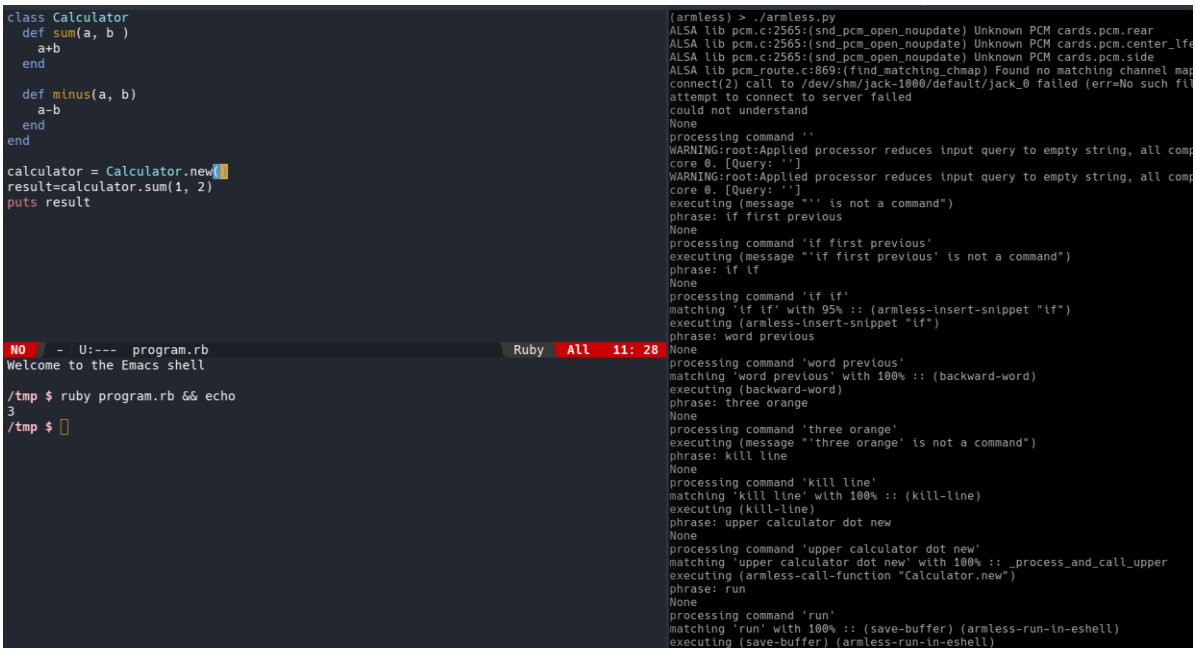
6.6 EXECUÇÃO

Para iniciar este trabalho, primeiro deve-se abrir o GNU Emacs, e, em seguida, abrir um terminal e executar o programa em Python chamado *armless.py*, na raíz do projeto. Este programa executa o que este projeto chama de MR. Por padrão, a conexão entre o MR e GNU Emacs é feita por TCP/IP. Porém, pode-se passar a flag “--daemon-mode” para utilizar o servidor interno do GNU Emacs.

Após executar o programa *armless.py*, este ficará rodando em segundo plano, aguardando o usuário falar os comandos de voz, que serão executados e visualizados no GNU Emacs. No terminal, são exibidas algumas informações de *debug*, como o comando sendo executado e a transcrição do áudio realizada pelo CMU Sphinx.

A Figura 11 mostra o protótipo desenvolvido em execução. Na parte superior da janela esquerda está o código escrito, e, na parte inferior, a saída do código executado. Na janela direita está o programa sendo executado e exibindo suas informações de *debug*.

Figura 11 - Protótipo em execução



Fonte: do autor

6.7 PRECISÃO DE RECONHECIMENTO

Durante os testes de reconhecimento de fala, houve uma constante dificuldade do reconhecimento preciso de algumas palavras. Ao tentar falar palavras como "undo", frequentemente o CMU Sphinx reconheceu como "one two". Em alguns casos, houve dificuldade para pronunciar a palavra no início da frase. Ao tentar falar “indent” isoladamente ou no início da frase, por exemplo, poucas vezes obteve-se o

resultado esperado. Porém, falando “indent” precedido de alguma outra palavra, os resultados foram mais satisfatórios.

Existem várias variáveis que influenciam na precisão do reconhecimento, como qualidade do microfone, ruídos do ambiente, algoritmo de reconhecimento e seu modelo acústico, sotaque, entre outros. Tentou-se isolá-las o máximo possível para obter uma precisão maior, mas, não obtendo um resultado satisfatório, e devido aos recursos humanos e financeiros limitados, decidiu-se utilizar palavras com fonéticas mais distintas.

Os microfones utilizados foram um *headset* da marca Multilaser e um smartphone Android da Motorola, onde o segundo obteve melhor precisão nos resultados. Existem modelos de microfones voltados para precisão na captura do áudio, ideais para este tipo de tarefa, que podem custar até mil dólares (PHILIPS, 2018), prometendo maior precisão até mesmo em ambientes com barulho.

6.7.1 Adaptação do modelo acústico do CMU Sphinx

Com o objetivo de melhorar a precisão do reconhecimento de fala do CMU Sphinx, foi realizado uma adaptação do modelo acústico. Esta adaptação consiste em definir frases e criar arquivos contendo suas transcrições apontando para um arquivo de áudio (como mostrado na imagem abaixo). Em seguida, grava-se um áudio para cada uma das frases transcritas, e, depois, executa-se alguns comandos para gerar um novo modelo acústico do CMU Sphinx.

Figura 12 - Arquivo com transcrição das frases para a adaptação do modelo acústico

```
<s> author of the danger trail philip steels etc </s> (voice_arctic_0001)
<s> not at this particular case tom apologized whittemore </s> (voice_arctic_02)
<s> for the twentieth time that evening the two men shook hands </s> (voice_arctic_03)
<s> lord but i'm glad to see you again phil </s> (voice_arctic_04)
<s> will we ever forget it </s> (voice_arctic_05)
<s> god bless 'em i hope i'll go on seeing them forever </s> (voice_arctic_06)
<s> and you always want to see it in the superlative degree </s> (voice_arctic_07)
<s> gad your letter came just in time </s> (voice_arctic_08)
<s> he turned sharply and faced gregson across the table </s> (voice_arctic_09)
<s> i'm playing a single hand in what looks like a losing game </s> (voice_arctic_10)
<s> if i ever needed a fighter in my life i need one now </s> (voice_arctic_11)
<s> gregson shoved back his chair and rose to his feet </s> (voice_arctic_12)
<s> he was a head shorter than his companion of almost delicate physique </s>
(voice_arctic_13)
<s> now you're coming down to business phil he exclaimed </s> (voice_arctic_14)
<s> it's the aurora borealis </s> (voice_arctic_15)
<s> there's fort churchill a rifleshot beyond the ridge asleep </s> (voice_arctic_16)
<s> from that moment his friendship for belize turns to hatred and jealousy </s>
(voice_arctic_17)
<s> there was a change now </s> (voice_arctic_18)
<s> i followed the line of the proposed railroad looking for chances </s> (voice_arctic_19)
<s> clubs and balls and cities grew to be only memories </s> (voice_arctic_20)
```

Fonte: Do autor.

Notou-se uma melhor precisão após realizada a adaptação acústica, porém apenas até certo ponto. Segundo o site do CMU Sphinx, pode-se esperar uma melhora de em torno de 10% na precisão (CMU SPHINX, 2018). Pode ser interessante estudar melhor o funcionamento da adaptação acústica e outros meios similares visando obter uma melhora maior na precisão.

6.8 RESULTADOS OBTIDOS

O levantamento bibliográfico realizado neste trabalho permitiu entender melhor os riscos de possíveis lesões causas pelo uso incorreto ou em excesso de teclado e mouse, reafirmando a importância do desenvolvimento de métodos alternativos para programar (como através de comandos de voz, como este trabalho propõe), buscando tanto auxiliar deficientes físicos quanto prevenir possíveis lesões, principalmente pela escassez de softwares com este propósito. Além disso, o levantamento bibliográfico também permitiu entender melhor a respeito de detalhes técnicos dos SRVs e seus algoritmos.

Por meio do ambiente desenvolvido neste trabalho, foi possível inserir, editar, navegar e manipular códigos de programação, além de executá-los e visualizar o

resultado. Constituindo, assim, uma alternativa para que pessoas com limitações físicas nos membros superiores tenham a oportunidade de programar, além de também servir de complemento ao teclado e mouse. A precisão de reconhecimento, porém, não foi tão satisfatória, apesar da adaptação acústica realizada no CMU Sphinx. Por isso, para obter uma precisão maior, limitou-se o dicionário de palavras analisado pelo SVR, estando disponíveis apenas aquelas necessárias para realizar as tarefas.

Durante o desenvolvimento, os testes foram realizados manualmente, já que não se usou *Test-Driven Development* (TDD). Por este motivo, os testes tornaram-se trabalhosos e exaustivos, consumindo boa parte do tempo de desenvolvimento, visto que uma nova alteração frequentemente afetava o funcionamento de outras partes, e isso, muitas vezes, só era percebido em um outro momento.

O protótipo desenvolvido permite a inserção palavras, frases, símbolos e nomes de métodos onde os espaços são substituídos por *underscore*. Para agilizar a escrita, foram desenvolvidos *snippets*, permitindo que blocos de código recorrentes, como *if, while,* classes e funções sejam inseridos com apenas um comando. Também foram desenvolvidos comandos para, por exemplo, inserir uma frase ou palavra entre aspas.

Foram criados vários comandos de navegação, podendo-se mover o cursor em linhas, palavras, início e fim do arquivo, entre outros. Estes comandos podem ser repetidos automaticamente, para evitar ficar falando o mesmo comando várias vezes. Em conjunto com a navegação, o protótipo permite selecionar um bloco de código para copiar, colar ou apagar.

A qualquer momento, é possível executar o código escrito e visualizar sua saída em um emulador de terminal dentro do próprio GNU Emacs, podendo inclusive interagir informando dados para o programa, caso o programa solicite entrada de dados por parte do usuário. Neste emulador de terminal também é possível executar comandos nativos do sistema operacional, como *ls* para listar os arquivos do diretório ou *cp* para copiar um arquivo. A interação com o terminal, porém, não foi otimizada para os comandos de voz.

O módulo principal foi desenvolvido em Python, o que facilitou a integração com o CMU Sphinx, pelo fato de existir a API SpeechRecognition, que abstrai a

comunicação entre a linguagem de programação e o SRF. Dessa forma, não foi necessário realizar implementações de baixo nível para captura de áudio e comunicação com o SRF.

A linguagem de programação escolhida para programar com os comandos de voz foi o Ruby, por ser uma linguagem de sintaxe simples e objetiva, além de ser interpretada, o que facilita sua execução. Estas características tornam a interação por comandos de voz menos complexa e mais rápida. Linguagens como C ofereceriam mais obstáculos por serem muito verbosas, além de ser necessário compilar o código antes de executar.

Durante os testes, percebeu-se que a viabilidade da utilização de voz para programar é promissora não apenas para pessoas com deficiências físicas ou motoras, mas para qualquer programador, pois algumas tarefas podem ser executadas de forma mais ágil com comandos de voz. Sendo assim, utilizar comandos de voz em conjunto com o tradicional teclado e mouse pode, potencialmente, oferecer uma melhor produtividade do que apenas teclado e mouse.

A escolha do GNU Emacs para executar os comandos de voz se mostrou propícia. Por se tratar de um editor extensível, flexível e altamente configurável, não houve grandes dificuldades para realizar a integração com os comandos de voz. A possibilidade de iniciar um servidor TCP/IP dentro do próprio editor e enviar os comandos através de pacotes pela rede foi simples e eficaz para atingir o objetivo desejado. Funções de inserção de *snippets*, inserção de texto e navegação foram desenvolvidas dentro do próprio editor. Estas tarefas provavelmente seriam mais problemáticas em editores ou IDE’s mais tradicionais.

Percebeu-se que, apesar das dificuldades encontradas, o resultado final foi satisfatório. Utilizar a voz para programar pode ser uma alternativa viável, tendo o potencial de ser tanto um complemento para o teclado e mouse, principalmente se os problemas encontrados forem trabalhados e o projeto otimizado para uma melhor produtividade.

7 CONCLUSÃO

Existe uma carência de meios de interação alternativos a teclado e mouse para programar. O desenvolvimento de novos meios para realizar esse tipo de tarefa pode ajudar na inclusão de pessoas que, por algum tipo de deficiência física, tem dificuldade de usar teclado e mouse, além de auxiliar na prevenção de lesões causadas pelo uso em excesso destes dispositivos. Utilizar a fala para programar pode ser uma alternativa. Apesar de já existirem softwares de reconhecimento de fala, eles não são otimizados para tarefas mais específicas como programar.

Este trabalho, utilizando a *engine* de reconhecimento de fala CMU Sphinx e o editor de texto GNU Emacs, teve como objetivo desenvolver um protótipo de um ambiente de reconhecimento de fala para programar. Através desse ambiente, foi possível escrever, editar, manipular e executar códigos de programação. Apesar de, dependendo do digitador e do tipo de tarefa, não se mostrar tão ágil quanto um teclado, avalia-se que o reconhecimento de fala pode ser sim uma alternativa viável, podendo-se, inclusive, utilizar de forma híbrida junto com o teclado.

Os códigos escritos utilizando o protótipo desenvolvido foram em Ruby, por ser uma linguagem interpretada e de sintaxe simples e objetiva. Os *snippets* e algumas funções mais específicas, incluindo, por exemplo, tarefas como a execução do programa, foram desenvolvidos e otimizados exclusivamente para o Ruby. Um desafio para futuros trabalhos seria otimizar este ambiente para se programar em outras linguagens, como C ou Java, tornando este projeto mais acessível.

Os testes foram realizados apenas no sistema operacional GNU/Linux. Entretanto, pelo fato de todas as tecnologias usadas neste trabalho serem livres e multiplataforma, é provável que também funcione em outros sistemas como Windows e Mac OS, com algumas eventuais adaptações. Futuros trabalhos podem realizar os testes em outros sistemas operacionais e fazer as alterações necessárias.

Um dos principais desafios foi a precisão de reconhecimento. Frequentemente a palavra ou frase falada era reconhecida de forma incorreta. Este é um item importante para a produtividade e viabilidade do projeto. Para ter-se uma precisão maior, foi utilizado um dicionário limitado, apenas com as palavras essenciais para a utilização. Seria

interessante que trabalhos futuros avaliem formas de aprimorar a precisão e aumentar o dicionário de palavras disponíveis.

Pelo fato de ter sido utilizado apenas softwares livres neste trabalho, não foi analisada a implementação de outros SRVs além do CMU Sphinx. SRVs comerciais, como o Dragon NaturallySpeaking, podem, possivelmente, oferecer melhor precisão de reconhecimento, além suportar mais idiomas. Contudo, SRVs comerciais costumam ser caros e, muitas vezes, são exclusivos para alguma plataforma específica.

A comunicação entre o programa de reconhecimento de fala e o GNU Emacs foi feita através de um servidor TCP/IP dentro do GNU Emacs. Este servidor é iniciado dentro do próprio GNU Emacs, e o programa de reconhecimento de fala atua como *client,* enviando os comandos formatados em *elisp* (linguagem interna do GNU Emacs) através de pacotes pela rede, onde o GNU Emacs, ao receber, os executa. Contudo, essa comunicação pode estar sujeita a vulnerabilidades de segurança, já que isso não foi avaliado neste trabalho.

Além do servidor TCP/IP, também foram desenvolvidas funções auxiliares para o GNU Emacs, com o objetivo de executar comandos enviados que, por padrão, o editor não possui ou não se comporta de forma esperada para o funcionamento correto deste trabalho. Essas funções incluem ações como deletar palavra sob cursor e executar o código em um *buffer* auxiliar.

Por se tratar de um protótipo, este projeto ainda carece de funções e recursos para melhorar a experiência do usuário. Questões como a instalação e execução são manuais. É necessário instalar as configurações do GNU Emacs, instalar o CMU Sphinx e as dependências do MR, entre outros procedimentos. E, para executar, o usuário precisa iniciar tanto o GNU Emacs quanto o MR. Desenvolver uma instalação automatizada e simplificar a execução através de uma interface mais amigável pode ajudar a tornar este projeto mais acessível.

Outra sugestão de trabalhos futuros seria a implementação de Test-drive Development (TDD). Com o aumento da complexidade do trabalho, muitos erros inesperados foram se mostrando, e os testes manuais consumiram bastante tempo. A implementação de TDD pode agilizar o desenvolvimento deste projeto e reduzir a probabilidade de *bugs*.

REFERÊNCIAS

BLATTER, B. M.; BONGERS, P. M. Duration of computer use and mouse use in relation to musculoskeletal disorders of neck or upper limb. **International Journal of Industrial Ergonomics**, v. 30, n. 4–5, p. 295–306, 2002.

BRASIL. **Dor relacionada ao trabalho**. 2012. Disponível em: <http://bvsms.saude.gov.br/bvs/publicacoes/dor_relacionada_trabalho_ler_dort.pdf>.

CAMERON, Debra *et al*. **Learning GNU Emacs**. 3. ed. O'Reilly Media, Inc., 2009.

CMU SPHINX. **CMU Sphinx**. 2009. Disponível em: <http://cmusphinx.sourceforge.net/>. Acesso em: 20 ago. 2017.

CMU SPHINX. **Training an acoustic model for CMUSphinx**. 2017. Disponível em: <https://cmusphinx.github.io/wiki/tutorialam/>. Acesso em: 14 nov. 2017.

CMU SPHINX. **Adapting the default acoustic model**. 2018. Disponível em: < https://cmusphinx.github.io/wiki/tutorialadapt/ >. Acesso em: 20 abr. 2018.

EUA. **Projections of occupational employment,** 2017. Disponível em: <https://www.bls.gov/careeroutlook/2017/article/occupational-projections-charts.htm>.

FREITAS, Diamantino; CHEN, Fang; JOKINEN, Kristiina. **Speech Technology**. 2010. Disponível em: <http://www.springerlink.com/content/nq851h7548004076/>.

FONTÁNEZ, Xiomara Figueroa; FRANCO, Patricia Ordóñez. **Improving Programming Interfaces For People With Voice Recognition**. *In*: Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14. 2014, p. 331–332. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84911458336&partnerID=tZOtx3y1>.

GERR F *et al*. A Prospective Study of Computer Users: I.Study Design and Incidence of Musculoskeletal Symptoms and Disorders. **American Journal of Industrial Medicine**, v. 41, n. May 2002, p. 221–35, 2002.

GNU EMACS. **Emacs Website**. Disponível em: <https://www.gnu.org/software/emacs/>. Acesso em: 14 nov. 2017.

GNU EMACS. **Emacs - Further Information**. Disponível em: <https://www.gnu.org/software/emacs/further-information.html>. Acesso em: 28 ago. 2017.

GNU EMACS. **GNU Emacs repository**. Disponível em: <https://github.com/emacs-mirror/emacs>. Acesso em: 20 set. 2017.

GOOGLE. **Android Accessibility Help**. 2017a. Disponível em: <https://support.google.com/accessibility/android/answer/6007100?hl=en>. Acesso em: 28 ago. 2017.

GOOGLE. **Google Speech API**. Google Cloud Platform. 2017b. Disponível em: <https://cloud.google.com/speech>. Acesso em: 28 ago. 2017.

HUANG, Xuedong; ACERO, Alex; HON, Hsiao-Wuen. **Spoken language processing: A guide to theory, algorithm, and system development**. Prentice Hall PTR, 2001.

HUGGINS-DAINES, David *et al*. **Pocketsphinx: a free, real-time continuous speech recognition system for hand-held devices**. Icassp 2006, p. 185–188, 2006.

HUNT, Andrew. **What is speech recognition?** 1993. Disponível em: <http://www.speech.cs.cmu.edu/comp.speech/Section6/Q6.1.html>. Acesso em: 26 ago. 2017.

IBM. **IBM Watson**. Ibm. 2017. Disponível em: <https://www.ibm.com/watson/services/speech-to-text/>.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **Pesquisa Nacional de Saúde 2013: percepção de estado de saúde, estilo de vida e doenças crônicas**. 2014. Disponível em: <ftp://ftp.ibge.gov.br/PNS/2013/pns2013.pdf>.

IDOWU, PA; ADEDOYIN, RA; ADAGUNODO, RE. **Computer Related Repetitive Strain Injuries**. Journal of the Nigeria Society of Physiotherapy, 2005. Disponível em: <http://www.biomedsearch.com/article/Computer-related-repetitive-strain-injuries/167696978.html>.

JUANG, B. H.; LAWRENCE, R Rabiner. **Automatic Speech Recognition**. 2005. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=03CEB9F119B54EC3CFBEA76C5BBD3A21?doi=10.1.1.90.5614&rep=rep1&type=pdf>. Acesso em: 12 abr. 2018

KENYON, J. S.; KNOTT, T. A. **A Pronouncing Dictionary of American English**. 1949.

KONIJNENBERG, Hester S *et al*. **Conservative treatment for repetitive strain injury**. Scandinavian Journal of Work, Environment & Health, v. 27, n. 5, p. 299–310, 2001.

LEE, K. *et al*. **A review of physical exercises recommended for VDT operators**. Applied Ergonomics, v. 23, n. 6, p. 387–408, 1992.

MELPA. **Melpa**. Disponível em: <http://melpa.org/>. Acesso em: 28 ago. 2017.

MICROSOFT. **Accessibility Tools for Windows**. 2017. Disponível em: <https://www.microsoft.com/en-us/accessibility/windows>. Acesso em: 26 ago. 2017.

MICROSOFT. **Windows Speech Recognition Commands**. 2016. Disponível em: <https://support.microsoft.com/en-us/help/12427/windows-speech-recognition-commands>. Acesso em: 26 ago. 2017.

MICROSOFT. **Windows Vista Speech Recognition Step-by-Step Guide**. 2007. Disponível em: <https://msdn.microsoft.com/en-us/library/bb530325.aspx>. Acesso em: 26 ago. 2017.

MIRANDA, Andréa da Silva. **Modelo de Acessibilidade em Telecentros**. 2007. Disponível em: <https://repositorio.ufsc.br/xmlui/handle/123456789/89574>.

NUANCE. **Nuance Dragon**. Disponível em: <https://www.nuance.com/dragon/dragon-for-pc.html>.

PETRICK, Elizabeth. **Making Computers Accessible**. 2015.

PHILIPS. **SpeechMike Premium Air**. Disponível em: <https://www.dictation.philips.com/tw/products/speech-recognition/speechmike-premium-air-dictation-and-speech-recognition-set-pse4000/>. Acesso em: 26 abr. 2018.

PINEDA, Evelyn; GONZALEZ, Carlos. **Networking Skills in Latin America.** Tirar: Tirar, 2016. 32 p. Disponível em: <http://www.cisco.com/assets/csr/pdf/IDC_Skills_Gap_-_LatAm.pdf>. Acesso em: 28 out. 2016.

RAMOS, Erlynn Mae Ang; JAMES, Christine A.; BEAR-LEHMAN, Jane. **Children's computer usage: Are they at risk of developing repetitive strain injury?** Work: A Journal of Prevention, Assessment and Rehabilitation, v. 25, n. 2, p. 143–154, 2005. Disponível em: <http://iospress.metapress.com/content/2la2bt1qhue7lc4f/>.

REINO UNIDO. **Internet access – households and individuals: 2017**. 2017. Disponível em: <https://www.ons.gov.uk/peoplepopulationandcommunity/householdcharacteristics/homeinternetandsocialmediausage/bulletins/internetaccesshouseholdsandindividuals/2017/pdf>.

SIMONEAU, Serge; ST-VINCENT, Marie; CHICOINE, Denise. **Work-Related Musculoskeletal Disorders (WMSDs): A Better Understanding for More Effective Prevention**. Ergonomic improvement of work: concrete cases, 1996. Disponível em: <https://www.irsst.qc.ca/media/documents/PubIRSST/RG-126-ang.pdf>.

SIMONEAU, Guy G; MARKLIN, Richard W; BERMAN, Joseph E. **Effect of computer keyboard slope on wrist position and forearm electromyography of typists without musculoskeletal disorders.** Physical therapy, v. 83, n. 9, p. 816–830, 2003.

STALLMAN, Richard M. **EMACS the extensible, customizable self-documenting display editor**. 1981. Disponível em: <https://www.gnu.org/software/emacs/emacs-paper.html>.

STALLMAN, Richard. **GNU Emacs Manual**. 2017. Disponível em: <https://www.gnu.org/software/emacs/manual/pdf/emacs.pdf>.

TITTIRANONDA, Pat; BURASTERO, Stephen; REMPEL, David. Risk factors for musculoskeletal disorders among computer users. **Occupational Medicine,** Livermore, CA, v. 14, n. 1, jan. 1999.

WAGNER, Amber *et al*. **Programming by Voice: A Hands-free Approach for Motorically Challenged Children**. *In*: CHI ’12 Extended Abstracts on Human Factors in Computing Systems. New York, NY, USA: ACM, 2012, p. 2087–2092. (CHI EA ’12). Disponível em: <http://doi.acm.org/10.1145/2212776.2223757>.

WAIBEL, Alex; LEE, Kai-fu. **Readings in Seech Recognition**. 1990.

YASSI, Annalee. **Repetitive strain injuries**. Lancet, v. 349, n. 9056, p. 943–947, 1997.

ZHANG, Anthony. **Speech Recognition (version 3.7)**. Disponível em: <https://github.com/Uberi/speech_recognition>. Acesso em: 22 abr. 2018.

APÊNDICE A – Tabela de comandos

(continua)

Frase	Comando (Elisp)	Ação	Callback
line up	(previous-line)	Sobe uma linha	Não
line down	(next-line)	Desce uma linha	Não
word previous	(backward-word)	Vai para palavra anterior	Não
word back	(backward-word)	Alias para o comando anterior	Não
word next	(forward-word)	Vai para próxima linha	Não
word transpose	(transpose-words)	Inverte a palavra da esquerda com a da direita	Não
line start	(move-beginning-of-line 1)	Vai para o início da linha	Não
line end	(move-end-of-line 1)	Vai para o final da linha	Não
move right	(forward-char)	Avança um caractere para a direita	Não
move left	(backward-char)	Avança um caractere para a esquerda	Não
new line	(funcall (lambda () (move-end-of-line 1) (newline)))	Insere uma nova linha	Não
delete word	(delete-word-under-cursor)	Apaga a palavra sob o cursor	Não
delete	(backward-delete-char 1)	Apaga um caractere a esquerda	Não

Frase	**Comando (Elisp)**	**Ação**	**Callback**
delete right	(right-char) (right-char) (backward-delete-char 1)	Apaga um caractere a direita	Não
selection start	(armless-selection-start)	Inicia modo de seleção	Não
selection end	(armless-selection-end)	Finaliza modo de seleção	Não
delete region	(delete-region (region-beginning) (region-end))	Apaga região selecionada	Não
expand region	(er/expand-region 1)	Expande região em níveis	Não
expand	(er/expand-region 1)	Alias para o comando anterior	Não
kill line	(kill-line)	Apaga do cursor até o final da linha	Não
undo	(undo)	Desfaz última ação	Não
goddamnit	(undo)	Alias para comando anterior	Não
redo	(undo-tree-redo)	Refaz ação desfeita	Não
space	(armless-insert " ")	Insere um espaço	Não
copy	(armless-copy)	Copia linha ou seleção se houver	Não
paste	(armless-paste)	Cola conteúdo copiado	Não
next	(armless-next)	Avança para próximo nível do snippet	Não

Frase	**Comando (Elisp)**	**Ação**	**Callback**
previous	(armless-previous)	Retrocede para nível anterior do snippet	Não
print	(armless-insert-snippet "print")	Escreve 'print'	Não
new for	(armless-insert-snippet "for")	Insere o snippet do bloco for	Não
new while	(armless-insert-snippet "while")	Insere o snippet do bloco while	Não
new class	(armless-insert-snippet "class")	Insere o snippet de classe	Não
new if	(armless-insert-snippet "if")	Insere o snippet do bloco if	Não
new function	(armless-insert-snippet "def")	Insere o snippet de uma função	Não
convert integer	(armless-insert ".to_i")	Insere comando de converção para inteiro (Ruby)	Não
integer	(armless-insert ".to_i")	Alias para comando anterior	Não
string	(armless-insert ".to_s")	Insere comando de converção para string (Ruby)	Não
convert string	(armless-insert ".to_s")	Alias para comando anterior	Não
variable	(armless-insert "variable")	Escreve 'variable'	Não
counter	(armless-insert "counter")	Escreve 'counter'	Não
run	(save-buffer) (armless-run-in-eshell)	Salva e executa o programa	Não

Frase	**Comando (Elisp)**	**Ação**	**Callback**
switch-buffer	(other-window 1)	Alterna entre as 'janelas' do Emacs abertas	Não
save	(save-buffer)	Salva	Não
enter	(setq unread-command-events (listify-key-sequence (kbd "RET")))	Simula a tecla Enter	Não
only	(delete-other-windows)	Esconde demais janelas do Emacs	Não
repeat	_repeat	Repete último comando	Sim
cancel	(armless-quit)	Cancela ação ou seleção	Não
number	_insert_number	Escreve números em forma de símbolo	Sim
insert	_process_and_insert	Escreve o que foi falado	Sim
quote	_process_and_insert_quote	Escreve o que foi falado entre aspas	Sim
call	_process_and_call	Escreve a chamada de uma função	Sim
upper	_process_and_call_upper	Converte caractere ou seleção para letra maiúscula	Sim
lower	(armless-uncapitalize)	Converte caractere ou seleção para letra minúscula	Não
first	_process_first	Escreve primeiro caractere do argumento falado	Sim
new file	(armless-open-buffer)	Abre um arquivo para edição	Não
buffer end	(end-of-buffer)	Vai para o fim do buffer	Não

Frase	**Comando (Elisp)**	**Ação**	**Callback**
buffer start	(beginning-of-buffer)	Vai para o início do buffer	Não
equal	(armless-insert "=")	Escreve o caractere =	Não
comma	(armless-insert ", ")	Escreve o caractere ,	Não
times	(armless-insert "*")	Escreve o caractere *	Não
plus	(armless-insert "+")	Escreve o caractere +	Não
minus	(armless-insert "-")	Escreve o caractere -	Não
greater	(armless-insert ">")	Escreve o caractere >	Não
lesser	(armless-insert "<")	Escreve o caractere <	Não
dot	(armless-insert ".")	Escreve o caractere .	Não
At sign	(armless-insert "@")	Escreve o caractere @	Não

APÊNDICE B – ARTIGO CIENTÍFICO

Ambiente de Reconhecimento de Fala para Escrever Códigos de Programação Integrado ao GNU Emacs

Jesse N. Medeiros[1], Gilberto V. Silva[2]

[1] Unidade Acadêmica de Ciências, Engenharias e Tecnologias, Curso de Ciência da Computação – Universidade do Extremo Sul Catarinense (UNESC)
Criciúma – SC – Brazil

jessenzr@gmail.com, gilbertovieirasilva@hotmail.com

Abstract. *There is a growing demand of jobs for the Information Technology field in the entire world. However, many studies concluded that excessive computer usage, along with a bad posture, may cause a variety of injuries. This paper's goal is to use voice commands as an alternative interaction method to program, aiming to reduce or replace the keyboard and mouse usage to write computer code. With that in mind, a prototype to write computer code was developed. This prototype uses the speech recognition engine CMU Sphinx as the backend to perform the speech-to-text conversion and executes the corresponding actions to the spoken commands in the GNU Emacs text editor. Through this prototype it was possible to write, edit, manipulate and run computer code. This environment was written in Python and optimized to work with the Ruby programming language, while still being possible to be optimized for other languages.*

Resumo. *Existe uma crescente demanda por profissionais da área de Tecnologia da Informação no mundo inteiro. Entretanto, várias pesquisas concluem que o uso em excesso de computador, aliado a má postura, podem causar vários tipos de lesões. Este trabalho propõe utilizar comandos de voz como um meio de interação alternativo com o computador para programar. Para isso, foi desenvolvido um protótipo para escrever códigos de programação através de comandos de voz. Este protótipo utiliza a engine de reconhecimento de fala CMU Sphinx para realizar a conversão de áudio para texto e executa as ações correspondentes aos comandos falados no editor de texto GNU Emacs. Através deste protótipo foi possível escrever, editar, manipular e executar códigos de programação. Este ambiente desenvolvido foi escrito em Python e otimizado para trabalhar com a linguagem de programação Ruby, podendo também ser otimizado para outras linguagens.*

1. Introdução

Profissões como desenvolvedor de software e similares estão se tornando cada vez mais populares, sendo uma das áreas que mais crescem nos Estados Unidos (EUA, 2017). Contudo, estudos indicam que o uso excessivo de teclado e mouse, principalmente quando não são tomados os devidos cuidados, está diretamente relacionado a Lesões por Esforço Repetitivo (LER) (BLATTER; BONGERS, 2002, tradução nossa).

A tarefa de manipulação de código-fonte, que costuma ser a principal tarefa de um desenvolvedor de software, dificilmente pode ser executada sem teclado e mouse. Existem poucas alternativas para esse tipo de interação, e, em geral, costumam ser pouco produtivas (FONTANEZ; FRANCO, 2014, tradução nossa).

Por isso, é importante pensar em novos métodos para realizar esse tipo de tarefa, buscando diminuir a incidência de LER nos profissionais da área, além de auxiliar aqueles que, por algum tipo de limitação física, possuem dificuldade para utilizar teclado ou mouse.

Uma dessas alternativas pode ser o reconhecimento de fala. Já existem meios de controlar o computador (MICROSOFT, 2016) e celulares (GOOGLE, 2017) apenas com comandos de voz, mas esses não são focados para a manipulação de código fonte.

Neste trabalho, desenvolveu-se um protótipo que possibilita manipular códigos de programação através de comandos de voz, onde o CMU Sphinx é responsável pela conversão de fala para texto e o GNU Emacs é o editor de texto onde os comandos serão executados e visualizados. Este protótipo foi desenvolvido em GNU/Linux, podendo potencialmente ser portado para outros sistemas operacionais pelo fato de terem sido utilizadas ferramentas multiplataforma durante o desenvolvimento.

2 Reconhecimento de fala, CMU Sphinx, e GNU Emacs

O reconhecimento de fala vem sendo estudado há décadas, e depende de um profundo estudo linguístico e de como a fala humana é produzida. Seu objetivo é fazer com que máquinas sejam capazes de reconhecer a fala humana da forma mais natural possível, beneficiando várias áreas ocupacionais onde os profissionais podem interagir com o computador ou outra máquina em estão com as mãos ocupadas, além de possibilitar que pessoas que, por algum tipo de deficiência tem dificuldade para utilizar mouse ou teclado, interajam com o computador (WAIBEL; LEE, 1990, tradução nossa).

Resumidamente, o funcionamento da maioria dos Sistemas de Reconhecimento de Fala (SRF) consiste em isolar os fonemas, que são a menor unidade sonora de um idioma, e, a partir do resultado, tentar organizá-los conforme as possibilidades do idioma sendo trabalhado, buscando, em seguida, a palavra correspondente no banco de dados, levando também em consideração a semântica, caso não seja uma palavra isolada. Portanto, as palavras montadas pelo SRV são na verdade uma sequência de fonemas que correspondem a pronuncia da mesma. Porém, existem vários métodos e algoritmos para executar esse processo (HUANG; ACERO; HON, 2001, tradução nossa).

O GNU Emacs é um editor de texto que implementa um dialeto de lisp próprio chamado de Emacs Lisp, também conhecida por elisp, e através dessa linguagem pode-se desenvolver vários recursos que vão além de edição de texto. Por ter sido desenvolvido desde o início com extensibilidade em mente, o este editor oferece uma plataforma para desenvolvimento que vai além de ferramentas para edição de texto. Essas características permitem o desenvolvimento de aplicações dentro do próprio GNU Emacs, como calculadora, jogos, ou, como é o caso deste trabalho, integração com comandos de voz (STALLMAN, 1982, tradução nossa).

2.1 Trabalhos correlatos

A seguir são apresentados trabalhos que possuem alguma similaridade a este, seja relacionada ao tema ou ao objetivo.

Fontanez e Franco (2014, tradução nossa) buscam, através do reconhecimento de fala, auxiliar pessoas com algum tipo de limitação de mobilidade possam programar. No caso deste trabalho, foram criados cenários de uso, com vocabulário limitado, para controlar IDE's e editores de texto, além de possuir modelos acústicos variados para cada linguagem trabalhada. Entretanto, existem muitos detalhes específicos que precisam ser resolvidos, e os autores concluem que a melhor solução seria criar um ambiente de programação totalmente novo, além de sugerirem a criação de uma linguagem de programação falada que seja acessível para a maioria dos programadores.

Miranda (2007) propõe um modelo de acessibilidade para atendimento em TELECENTROS, buscando ajudar pessoas com necessidades especiais a interagir com o computador. Esse modelo considera deficiências visuais, auditivas, físicas, motoras, mentais. Os monitores dos TELECENTROS avaliam cada usuário individualmente através de dez etapas definidas no modelo, possuindo equipamentos especiais para os diferentes tipos de deficiência, como teclados e mouses adaptados, monitores reestruturados com cores e adesivos, fones de ouvido, softwares específicos, e outros.

Wagner et al. (2012) desenvolveram uma ferramenta chamada Myna para controlar o Scratch (programa utilizado para aprender programação) através de comandos de voz. O Scratch, que permite organizar blocos de forma lógica, como uma linguagem visual, pode ser controlado apenas por teclado e mouse de forma nativa. Esse modelo de interação permitiu que pessoas com limitações físicas (principalmente crianças, que são o foco do trabalho) pudessem aprender a programar utilizando o Scratch.

Os trabalhos citados reforçam a importância da acessibilidade digital, além de mostrar que é possível utilizar o reconhecimento de fala para programar, auxiliando pessoas com limitações físicas e ajudando a prevenir lesões.

3. Protótipo desenvolvido

O protótipo desenvolvido consiste em um programa chamado de Módulo de Reconhecimento (MR), desenvolvido em Python, que ouve os comandos, os analisa e processa, se necessário, e os envia para o GNU Emacs para serem executados. Também foi desenvolvido um *minor mode* em conjunto com algumas funções auxiliares no GNU Emacs. Internamente, o projeto foi batizado de *armless*. A Figura 1 apresenta um diagrama exibindo o fluxo dos comandos, mostrando os passos desde de a fala do usuário até a execução dentro do GNU Emacs.

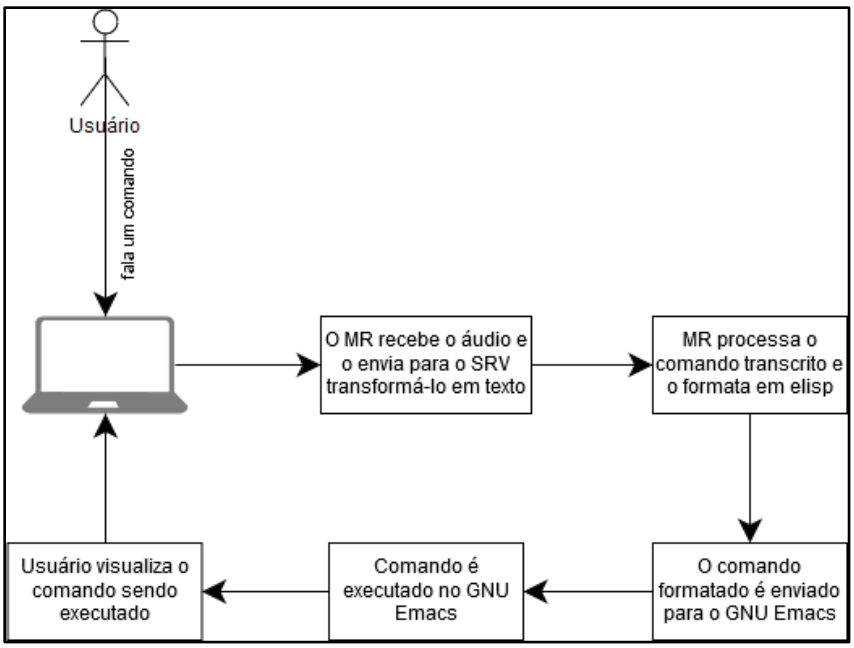


Figura 13. Diagrama do fluxo de execução

3.1 Linguagem de programação escolhida

Apesar do desenvolvimento do projeto ser em Python, nada impede a utilização de outras linguagens para serem manipuladas por este projeto. Esta seção trata da escolha dessa linguagem.

Alguns dos desafios da programação por fala são a manipulação de símbolos e a inserção de códigos de maneira eficiente e produtiva. Linguagens muito verbosas podem tornar as tarefas de edição por fala mais demoradas e trabalhosas. Também é importante que a linguagem escolhida seja interpretada ao invés de compilada para facilitar no processo de execução do código. Por isso, neste trabalho, foram analisadas algumas linguagens interpretadas com sintaxe mais simples para realizar o desenvolvimento.

Inicialmente, pretendia-se utilizar Python, pois é uma linguagem de sintaxe simples, objetiva e de tipagem dinâmica, ou seja, não é necessário informar o tipo de variáveis. Diferente de linguagens como C ou Java, por exemplo, não necessita de, entre outras formalizações, função *main* para executar o código. A Figura 2 ilustra um exemplo de *Hello* World em Python. Entretanto, pelo fato de a indentação fazer parte da sintaxe, seria preciso que o usuário indentasse manualmente os blocos de código, gerando mais uma tarefa extra que não seria necessária em uma outra uma linguagem sem esta característica.

```
# Hello world em Python
print('Hello world')
```

Figura 14. Exemplo de Hello World em Python

Após novas análises, decidiu-se utilizar Ruby, que, assim como Python, também é uma linguagem interpretada e de sintaxe simples e objetiva. Inclusive, estas duas linguagens compartilham algumas semelhanças na sintaxe, como na declaração de funções, onde ambas

utilizam a palavra *def*. No Ruby, porém, a indentação é tratada apenas como *whitespace* e não faz parte da sintaxe, eliminando a necessidade de intervenção do usuário na indentação do código.

3.2 Módulo de reconhecimento

O MR é um programa que fica rodando em *loop,* aguardando comandos do usuário. Quando o usuário fala um comando, o áudio é capturado e enviado para o CMU Sphinx para ser analisado, retornando então a fala convertida em texto.

Após a fala retornada, para identificar o comando o MR analisa as primeiras palavras do comando de voz e procura no banco de dados a ação correspondente. Quando a ação é um *callback*, ou seja, precisa ser processada pelo MR antes de ser enviada para o GNU Emacs, a função armazenada no campo é chamada passando como argumento a fala reconhecida. Os *callbacks* analisam os argumentos e os formatam conforme necessário. Por exemplo, quando é falado o comando *first*, o MR coleta a primeira letra do argumento e formata em um comando *elisp* de inserção, para só então enviar para o GNU Emacs e ser executado. Quando não é encontrada uma ação correspondente, envia-se uma mensagem dizendo que o comando não existe.

Todo comando, depois de devidamente processado, se transforma em um comando do GNU Emacs. Alguns deles são recursos nativos do editor, outros foram desenvolvidos especialmente para este trabalho.

Além de escrever e editar os códigos, também é possível executá-los. O comando *run* abre um terminal dentro do GNU Emacs exibindo o resultado do código executado. Também é possível alimentar o código com dados do usuário; o comando *switch buffer* foca no terminal aberto onde pode-se interagir, informando dados para o programa. Ao falar o comando *switch buffer* novamente, foca-se de novo no código, como demonstrado na Figura 3.

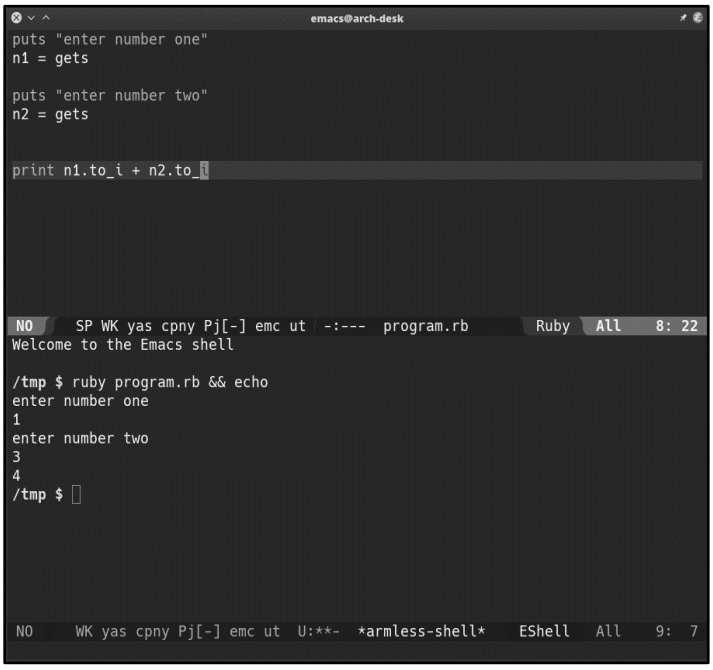


Figura 15. Código em Ruby (em cima) e seu resultado (embaixo)

O *buffer* execução de código também funciona como um terminal. Pode-se realizar comandos do sistema operacional e visualizar suas saídas normalmente, como *ls* para listar os

arquivos ou *rm* para apagar um arquivo. A interação com o terminal, porém, não foi o foco deste trabalho, que não foi otimizado para esta tarefa.

3.3 Armless mode

Além do MR, foi também foi desenvolvida uma biblioteca de comandos para o GNU Emacs. Esta biblioteca consistes em funções auxiliares e complementos para executar ações dos comandos que não estão disponíveis de forma nativa no GNU Emacs.

Neste módulo estão presentes ações para copiar e colar, abrir o arquivo para edição, converter letras em maiúsculo ou minúsculo, correções de comportamentos indesejados por estar sendo controlado externamente, inserção de texto, navegar entre os buffers abertos, entre outros. Também possui alguns tratamentos baseados no *major-mode*. Ao estar editando um arquivo, por exemplo, após inserção de texto, é chamado o comando do GNU Emacs *indent-for-tab-command*, que mantém o código indentado. Porém, essa ação não é desejável quando estiver utilizando o terminal. Este módulo também é responsável por esse tipo de tratamento.

3.4 Comunicação

Após o MR finalizar de processar e formatar um comando, ele deve ser executado no GNU Emacs. Para isso, foi preciso analisar um modelo de comunicação entre o MR e o GNU Emacs pelo qual os comandos possam ser enviados, onde o GNU Emacs atue como um servidor, aguardando os comandos serem enviados, e o MR atue como um *client,* enviando os comandos para o GNU Emacs. Ao falar o comando “line up”, por exemplo, o MR o transformará em um código de Emacs Lisp para então enviar através deste meio de comunicação para o GNU Emacs, onde este comando será executado e visualizado pelo usuário.

Para realizar esta comunicação, foi desenvolvido um servidor Transmission Control Protocol/Internet Protocol (TCP/IP) dentro do próprio GNU Emacs, e foi implementado buscando uma melhor alternativa de comunicação que não envolvesse enviar os comandos através da linha de comando. O servidor é iniciado automaticamente quando o *armless-mode* é ativado, e parado quando este modo é desativado. Quando o servidor está ativo, ele fica aguardando os pacotes contendo os comandos serem enviados para executá-los, permitindo que os comandos sejam enviados a partir de qualquer dispositivo. Isso pode possibilitar utilizar o GNU Emacs em um dispositivo e o MR em outro, se devidamente configurados.

Neste trabalho não foram analisadas as possíveis vulnerabilidades de segurança causadas pelo servidor TCP/IP. É importante que trabalhos futuros façam essa análise, implementando recursos como bloqueio de IPs no servidor do GNU Emacs para permitir a execução de apenas de dispositivos autorizados.

3.5 Gramática de comandos

A gramática de comandos foi desenvolvida utilizando a língua inglesa, que é o idioma que, na maioria dos SRVs, possui uma qualidade de reconhecimento mais avançada que os demais. Além disso, o inglês está presente na maioria das linguagens de programação, que costumam utilizar termos como *if, for, while, function,* entre outros.

Para obter-se uma melhor precisão, foi utilizado um dicionário com um número limitado de palavras. Dessa forma, ao falar um comando, o CMU Sphinx não precisa compará-lo com todas as palavras da língua inglesa, diminuindo a probabilidade de retornar um falso positivo. A partir

deste dicionário, foi criado um banco de dados em *sqlite* com, até então, 68 comandos para edição, navegação, inserção e execução de códigos, entre outras tarefas.

Neste banco de dados são armazenadas as seguintes informações:

a) ***id*:** identificação única do registro na tabela;

b) ***phrase*:** comando de voz que deve ser falado para executar a ação correspondente;

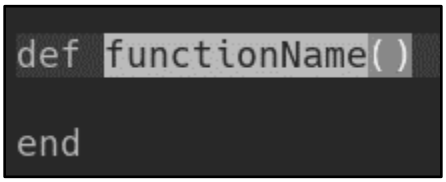
c) ***command*:** ação a ser executada, que pode ser enviada direto para o GNU Emacs ou processada antes, para formatar argumentos ou realizar algum outro tratamento. No segundo caso, será chamado a função armazenada neste campo, que está presente no MR, passando como argumento o comando falado;

d) ***callback*:** flag para diferenciar comandos que devem ser executados direto no GNU Emacs ou se devem ser processados pelo MR antes;

e) ***edit*:** flag para identificar comandos de edição.

Um exemplo entre os comandos disponíveis é o *insert,* que deve ser seguido das palavras que se deseja escrever. Ao falar *insert enter number one*, será inserido no buffer “enter number one”. Existem variações do comando *insert,* como o *quote,* que faz a mesma função, mas colocando as palavras faladas entre aspas. É possível também inserir *snippets* de códigos pré-definidos, como blocos condicionais, blocos de repetição, funções e classes, com os comandos *new if, new while, new function, new class*. Ao criar uma nova função (ilustrado na Figura 7), por exemplo, o cursor será colocado no nome da função que, após informado, pode-se falar *next* para levar o cursor para os argumentos da função, e falar *next* novamente para levar o cursor para o corpo da função.

Figura 16 - *Snippet* de uma função



Fonte: do autor

3.6 Precisão de reconhecimento

Durante os testes de reconhecimento de fala, houve uma constante dificuldade do reconhecimento preciso de algumas palavras. Ao tentar falar palavras como "undo", frequentemente o CMU Sphinx reconheceu como "one two". Em alguns casos, houve dificuldade para pronunciar a palavra no início da frase. Ao tentar falar “indent” isoladamente ou no início da frase, por exemplo, poucas vezes obteve-se o resultado esperado. Porém, falando “indent” precedido de alguma outra palavra, os resultados foram mais satisfatórios.

Existem várias variáveis que influenciam na precisão do reconhecimento, como qualidade do microfone, ruídos do ambiente, algoritmo de reconhecimento e seu modelo acústico, sotaque, entre outros. Tentou-se isolá-las o máximo possível para obter uma precisão maior, mas, não obtendo um resultado satisfatório, e devido aos recursos humanos e financeiros limitados, decidiu-se utilizar palavras com fonéticas mais distintas.

Com o objetivo de melhorar a precisão do reconhecimento de fala do CMU Sphinx, foi realizado uma adaptação do modelo acústico. Esta adaptação consiste em definir frases e criar arquivos contendo suas transcrições apontando para um arquivo de áudio (como mostrado na imagem abaixo). Em seguida, grava-se um áudio para cada uma das frases transcritas, e, depois, executa-se alguns comandos para gerar um novo modelo acústico do CMU Sphinx.

Notou-se uma melhor precisão após realizada a adaptação acústica, porém apenas até certo ponto. Segundo o site do CMU Sphinx, pode-se esperar uma melhora de em torno de 10% na precisão (CMU SPHINX, 2018). Pode ser interessante estudar melhor o funcionamento da adaptação acústica e outros meios similares visando obter uma melhora maior na precisão.

4 Resultados obtidos

Por meio do ambiente desenvolvido neste trabalho, foi possível inserir, editar, navegar e manipular códigos de programação, além de executá-los e visualizar o resultado. Constituindo, assim, uma alternativa para que pessoas com limitações físicas nos membros superiores tenham a oportunidade de programar, além de também servir de complemento ao teclado e mouse. A precisão de reconhecimento, porém, não foi tão satisfatória, apesar da adaptação acústica realizada no CMU Sphinx. Por isso, para obter uma precisão maior, limitou-se o dicionário de palavras analisado pelo SVR, estando disponíveis apenas aquelas necessárias para realizar as tarefas.

O protótipo desenvolvido permite a inserção palavras, frases, símbolos e nomes de métodos onde os espaços são substituídos por *underscore*. Para agilizar a escrita, foram desenvolvidos *snippets*, permitindo que blocos de código recorrentes, como *if, while,* classes e funções sejam inseridos com apenas um comando. Também foram desenvolvidos comandos para, por exemplo, inserir uma frase ou palavra entre aspas.

Foram criados vários comandos de navegação, podendo-se mover o cursor em linhas, palavras, início e fim do arquivo, entre outros. Estes comandos podem ser repetidos automaticamente, para evitar ficar falando o mesmo comando várias vezes. Em conjunto com a navegação, o protótipo permite selecionar um bloco de código para copiar, colar ou apagar.

A linguagem de programação escolhida para programar com os comandos de voz foi o Ruby, por ser uma linguagem de sintaxe simples e objetiva, além de ser interpretada, o que facilita sua execução. Estas características tornam a interação por comandos de voz menos complexa e mais rápida. Linguagens como C ofereceriam mais obstáculos por serem muito verbosas, além de ser necessário compilar o código antes de executar.

Durante os testes, percebeu-se que a viabilidade da utilização de voz para programar é promissora não apenas para pessoas com deficiências físicas ou motoras, mas para qualquer programador, pois algumas tarefas podem ser executadas de forma mais ágil com comandos de voz. Sendo assim, utilizar comandos de voz em conjunto com o tradicional teclado e mouse pode, potencialmente, oferecer uma melhor produtividade do que apenas teclado e mouse.

Percebeu-se também que, apesar das dificuldades encontradas, o resultado final foi satisfatório. Utilizar a voz para programar pode ser uma alternativa viável, tendo o potencial de ser tanto um complemento para o teclado e mouse, principalmente se os problemas encontrados forem trabalhados e o projeto otimizado para uma melhor produtividade.

5 Conclusão

Existe uma carência de meios de interação alternativos a teclado e mouse para programar. O desenvolvimento de novos meios para realizar esse tipo de tarefa pode ajudar na inclusão de pessoas que, por algum tipo de deficiência física, tem dificuldade de usar teclado e mouse, além de auxiliar na prevenção de lesões causadas pelo uso em excesso destes dispositivos. Utilizar a fala para programar pode ser uma alternativa. Apesar de já existirem softwares de reconhecimento de fala, eles não são otimizados para tarefas mais específicas como programar.

Este trabalho, utilizando a *engine* de reconhecimento de fala CMU Sphinx e o editor de texto GNU Emacs, teve como objetivo desenvolver um protótipo de um ambiente de reconhecimento de fala para programar. Através desse ambiente, foi possível escrever, editar, manipular e executar códigos de programação. Apesar de, dependendo do digitador e do tipo de tarefa, não se mostrar tão ágil quanto um teclado, avalia-se que o reconhecimento de fala pode ser sim uma alternativa viável, podendo-se, inclusive, utilizar de forma híbrida junto com o teclado.

Os testes foram realizados apenas no sistema operacional GNU/Linux. Entretanto, pelo fato de todas as tecnologias usadas neste trabalho serem livres e multiplataforma, é provável que também funcione em outros sistemas como Windows e Mac OS, com algumas eventuais adaptações. Futuros trabalhos podem realizar os testes em outros sistemas operacionais e fazer as alterações necessárias.

Um dos principais desafios foi a precisão de reconhecimento. Frequentemente a palavra ou frase falada era reconhecida de forma incorreta. Este é um item importante para a produtividade e viabilidade do projeto. Para ter-se uma precisão maior, foi utilizado um dicionário limitado, apenas com as palavras essenciais para a utilização. Seria interessante que trabalhos futuros avaliem formas de aprimorar a precisão e aumentar o dicionário de palavras disponíveis.

A comunicação entre o programa de reconhecimento de fala e o GNU Emacs foi feita através de um servidor TCP/IP dentro do GNU Emacs. Este servidor é iniciado dentro do próprio GNU Emacs, e o programa de reconhecimento de fala atua como *client,* enviando os comandos formatados em *elisp* (linguagem interna do GNU Emacs) através de pacotes pela rede, onde o GNU Emacs, ao receber, os executa. Contudo, essa comunicação pode estar sujeita a vulnerabilidades de segurança, já que isso não foi avaliado neste trabalho.

Por se tratar de um protótipo, este projeto ainda carece de funções e recursos para melhorar a experiência do usuário. Questões como a instalação e execução são manuais. É necessário instalar as configurações do GNU Emacs, instalar o CMU Sphinx e as dependências do MR, entre outros procedimentos. E, para executar, o usuário precisa iniciar tanto o GNU Emacs quanto o MR. Desenvolver uma instalação automatizada e simplificar a execução através de uma interface mais amigável pode ajudar a tornar este projeto mais acessível.

6. Referências

BLATTER, B. M.; BONGERS, P. M. Duration of computer use and mouse use in relation to musculoskeletal disorders of neck or upper limb. **International Journal of Industrial Ergonomics**, v. 30, n. 4–5, p. 295–306, 2002.

CMU SPHINX. **Adapting the default acoustic model**. 2018. Disponível em: < https://cmusphinx.github.io/wiki/tutorialadapt/ >. Acesso em: 20 abr. 2018.

FONTÁNEZ, Xiomara Figueroa; FRANCO, Patricia Ordóñez. **Improving Programming Interfaces For People With Voice Recognition**. *In*: Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility - ASSETS '14. 2014, p. 331–332. Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84911458336&partnerID=tZOtx3y1>.

GOOGLE. **Android Accessibility Help**. 2017. Disponível em: <https://support.google.com/accessibility/android/answer/6007100?hl=en>. Acesso em: 28 ago. 2017.

HUANG, Xuedong; ACERO, Alex; HON, Hsiao-Wuen. **Spoken language processing: A guide to theory, algorithm, and system development**. Prentice Hall PTR, 2001.

MICROSOFT. **Windows Speech Recognition Commands**. 2016. Disponível em: <https://support.microsoft.com/en-us/help/12427/windows-speech-recognition-commands>. Acesso em: 26 ago. 2017.

MIRANDA, Andréa da Silva. **Modelo de Acessibilidade em Telecentros**. 2007. Disponível em: <https://repositorio.ufsc.br/xmlui/handle/123456789/89574>.

STALLMAN, Richard. **GNU Emacs Manual**. 2017. Disponível em: <https://www.gnu.org/software/emacs/manual/pdf/emacs.pdf>.

WAGNER, Amber *et al*. **Programming by Voice: A Hands-free Approach for Motorically Challenged Children**. *In*: CHI '12 Extended Abstracts on Human Factors in Computing Systems. New York, NY, USA: ACM, 2012, p. 2087–2092. (CHI EA '12). Disponível em: <http://doi.acm.org/10.1145/2212776.2223757>.

WAIBEL, Alex; LEE, Kai-fu. **Readings in Seech Recognition**. 1990.