Universidade Metodista de Piracicaba
Faculdade de Ciências Exatas e da Natureza
Mestrado em Ciência da Computação

Arquitetura para Integração de Módulos de Reconhecimento de Fala em Plataforma Robótica Móvel

Tarcisio Jorge Bezerra

Orientador: Prof. Dr. Luiz Eduardo G. Martins

Piracicaba, SP
2012

Universidade Metodista de Piracicaba
Faculdade de Ciências Exatas e da Natureza
Mestrado em Ciência da Computação

Arquitetura para Integração de Módulos de Reconhecimento de Fala em Plataforma Robótica Móvel

Tarcisio Jorge Bezerra

Orientador: Prof. Dr. Luiz Eduardo G. Martins

Trabalho de Dissertação apresentado ao Mestrado em Ciência da Computação, da Faculdade de Ciências Exatas e da Natureza, da Universidade Metodista de Piracicaba – UNIMEP, como parte dos requisitos para obtenção do Título de Mestre em Ciência da Computação.

Piracicaba, SP
2012

Bezerra, Tarcisio Jorge

Arquitetura para Integração de Módulos de Reconhecimento de Fala em Plataforma Robótica Móvel
Piracicaba, 2012

140 p.

Orientador: Prof.Dr. Luiz Eduardo Galvão Martins
Dissertação (Mestrado) – Universidade Metodista de Piracicaba, Faculdade de Ciências Exatas e da Natureza, Programa de Pós-Graduação em Ciência da Computação.

1- Reconhecimento de fala, 2- Robótica móvel, 3- *RoboDeck*.

Arquitetura para Integração de Módulos de Reconhecimento de Fala em Plataforma Robótica Móvel

Autor: Tarcisio Jorge Bezerra

Orientador: Prof. Dr. Luiz Eduardo G. Martins

Dissertação de Mestrado apresentada em 10 de agosto 2012, à Banca Examinadora constituída pelos Professores:

Prof. Dr. Luiz Eduardo Galvão Martins – UNIFESP (Orientador)

Prof. Dr. Plínio Roberto Souza Vilela – ESEG

Prof. Dr. Antonio Valério Netto – Xbot

Profa.Dra. Sandra Maria Boscolo Brienza – UNIMEP

DEDICATÓRIA

À minha esposa, Renata, e minha filha, Sofia, pelo tempo que ficaram sozinhas.

AGRADECIMENTOS

A Deus, meu guia, por estar presente em todos os momentos de minha vida.

À minha querida esposa, pela compreensão.

Ao professor Luiz Eduardo Galvão Martins, pela orientação e principalmente incentivo e atenção dispensados no desenvolvimento deste trabalho.

Ao professor Edvaldo Simões da Fonseca Junior (da POLI-USP), por emprestar a plataforma robótica para a realização dos experimentos.

À professora Carmen Faraco.

Ao professor Guilherme Henrique de Souza.

A todos que colaboraram e acreditaram na realização desta dissertação, muito obrigado!

“Tudo tem seu apogeu e seu declínio... É natural que seja assim; todavia, quando tudo parece convergir para o que supomos o nada, eis que a vida ressurge, triunfante e bela!... Novas folhas, novas flores, na indefinida bênção do recomeço!...”

Chico Xavier”, de Carlos A. Baccelli

RESUMO

A área de robótica móvel tem crescido muito nos últimos anos, expandindo a utilização dos robôs móveis, nos afazeres do homem e, consequentemente, elevando a necessidade de uma melhor interface de comunicação entre eles. Em razão disso, o homem vem buscando formas mais práticas de interação com os robôs, como a interface de reconhecimento de fala, uma técnica bastante confortável, principalmente, em comparação com as tradicionais, através de teclados ou botões. Diante dessa realidade, o presente trabalho tem como base o estudo e a contextualização dos pacotes de reconhecimento de fala e das plataformas de robótica móvel, relacionando esses temas, no desenvolvimento de dois módulos de controle por voz: interno e externo ao robô. No módulo interno, integrou-se o programa de reconhecimento de fala *CVoiceControl n*a placa de alto desempenho, da plataforma robótica móvel denominada *RoboDeck*, da empresa Xbot, possibilitando controlar o robô pela fala, cujos comandos envolveram movimentos básicos: para frente, para trás, virar à esquerda, virar à direita e parar. No módulo externo, foi integrado ao SDK chamado “Giga de Testes”, um controle por voz, permitindo, por meio da fala, comandar o robô através de qualquer dispositivo móvel com o *Voice Search* instalado e conectado à Internet, bem como executar aplicativos para realização de atividades pré-definidas.

Palavras-Chave: Reconhecimento de fala, Robótica móvel, *RoboDeck*.

ABSTRACT

The field of mobile robotics has grown considerably in recent years, the use of expanding mobile robot, the tasks of man and thus increasing the need for an improved communication interface between them. As a result, men have been seeking more practical ways of interacting with the robots, such as speech recognition interface, a technique quite comfortable, especially compared with the traditional means of keyboards or buttons. Given this reality, this work is based on the study and the context of speech recognition packages and platforms for mobile robotics, relating these themes in the development of two modules of voice control: internal and external to the robot. In the internal module, has integrated speech recognition program in high-performance card CVoiceControl, mobile robotics platform named RoboDeck, from Xbot company, thus controlling the robot by speech, whose commands involved basic movements: forward, backward, turn left, turn right and stop. In the external module, was integrated into the SDK called 'Giga Test,' a voice control, allowing, through speech, commanding the robot through any mobile device with Voice Search installed and connected to the Internet, as well as running applications to pre-defined activities.

Keywords: *speech recognition, mobile robotics*, *RoboDeck*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE SIGLAS E ABREVIATURAS

API – Application Programming Interface

ASR – Automatic Speech Recognition

GPS – Global Positioning System

HMM – Hidden Markov Models

LPC – Linear Predictive Coding

MAP – Módulo de Alta Performance

MCC – Módulo de Controle de Comunicação

MCS – Módulo de Controle de Sessão

QV – Quantização Vetorial

SDK – *Software Development Kit*

USB – *Universal Serial Bus*

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização

Com o crescente desenvolvimento tecnológico, as interfaces homem-computador estão cada vez mais complexas, exigindo constantemente novas tecnologias de acesso, para que a interação homem - máquina ocorra em um nível de acessibilidade sempre eficiente.

A fala, os gestos, a linguagem de sinais, a linguagem escrita e o tato são formas utilizadas pelo homem para comunicar-se com o seu semelhante. Dentre essas formas de comunicação, a mais usada ainda é a fala, desde que os interlocutores não apresentem nenhum problema auditivo. A língua também pode ser um obstáculo para a comunicação entre duas pessoas de idiomas diferentes. No entanto, tem-se a Língua Inglesa, atualmente considerada universal.

Nos sistemas computacionais tradicionais, o homem se comunica com a máquina através de comandos acionados por entrada de texto ou por opções oferecidas pelos botões.

Nos sistemas computacionais que utilizam interface de reconhecimento e síntese de voz, a comunicação pode ser feita através da fala, da seguinte maneira: os comandos da máquina são sintetizados em voz para serem entendidos pelo homem e a fala do homem é transformada em comandos reconhecidos pela máquina, permitindo um diálogo direto entre eles, ao invés de enviar comando de texto por meio de uma interface baseada em teclado ou botões.

1.2 Objetivo

O objetivo deste trabalho está dividido em duas etapas.

A primeira consiste na integração de um aplicativo de reconhecimento de fala ao MAP (Módulo de Alta Performance) do robô móvel *RoboDeck*, que utiliza o sistema operacional Linux distribuição Debian, para comandar o robô por meio da voz. Os comandos que deverão ser reconhecidos e executados com velocidade contínua são: (1) para frente, (2) parar, (3) virar à esquerda e (4) virar à direita.

A segunda refere-se ao controle do robô pela fala, através de qualquer dispositivo móvel que possua Internet e um navegador da Google, como por

exemplo, um celular com Sistema Operacional *Android* ou um notebook com o navegador *Chorme* e a API *Voice Search*. Nesta etapa, além do robô reconhecer todos os comandos básicos de movimentação através da fala, ele também poderá girar à direita ou à esquerda no seu próprio eixo em 90 graus, desviar de objetos, ir até o usuário e se movimentar em um circuito de formato quadrado.

1.3 Justificativa

Os robôs móveis estão sendo utilizados progressivamente em diversos lugares: fato que torna fundamental o homem ter uma comunicação mais “natural” com eles.

Integrar uma interface de voz ao robô é possibilitar ao homem comandá-lo pela fala, através do próprio robô ou por meio de uma interface móvel, como um celular. Considerando-se que a fala é uma propriedade natural do homem, torna-se suficientemente motivador o incremento desse tipo de comunicação.

Outro fato importante é que os sistemas de reconhecimento de fala, ao proporcionarem melhor acessibilidade às maquinas, ampliam as possibilidades de uso, permitindo que pessoas com deficiência física, por exemplo, comandem uma cadeira de rodas robotizada e se locomovam através de comandos simples de fala (movimentar-se à direita, à esquerda, entre outros).

Pode-se também usar a integração de uma interface de voz para comandar robôs, em diversas outras situações como:

- ✓ Realizar tarefas domésticas em residências, usando um aspirador de pó ou um cortador de grama robotizados.
- ✓ Controlar e patrulhar ambientes, utilizando-se robôs móveis com câmeras.
- ✓ Transportar cargas, através do uso de plataformas robóticas móveis.
- ✓ Resgatar ou explorar ambientes hostis, como no caso das viagens interplanetárias, onde os robôs móveis fazem o conhecimento de lugares, em condições adversas ao ser humano.

1.4 Caracterização do Problema

Com base na plataforma *RoboDeck*, notou-se que integração interna deve ser realizada no MAP (Módulo de Alta Performance) do robô, caracterizando-se os seguintes problemas:

O MAP (Módulo de Alta Performance) do *RoboDeck* permite somente a integração de sistemas computacionais compatíveis com a plataforma Linux, em especial, a versão Debian. Além disso, por se tratar de um Robô fabricado no Brasil, é essencial que o mesmo reconheça comandos em português.

Atualmente, são escassos os pacotes de reconhecimento de fala compatíveis com o Sistema Operacional Linux e que permitam o reconhecimento da Língua Portuguesa. Isso ocorre em virtude do domínio da Microsoft no mercado de Sistemas Operacionais e do Inglês ser uma língua universal, diferentemente do Português.

Para a integração da fala em uma interface móvel, observou-se a necessidade de adotar um sistema de reconhecimento de fala que dispensasse treinar o locutor, antes de sua utilização. Assim, integrou-se ao SDK (Kit de Desenvolvimento de Software), que acompanha a plataforma *RoboDeck*, um software de reconhecimento de fala que entendesse qualquer locutor em português, por meio de um PC ou de um celular, gerando maior flexibilidade e mobilidade ao usuário.

1.5 Metodologia do Trabalho

A metodologia adotada foi dividida em seis etapas:

- ✓ Revisão bibliográfica;
- ✓ Comparação entre os pacotes de reconhecimento e síntese de fala;
- ✓ Comparação entre plataformas de robótica móveis disponíveis no mercado;
- ✓ Desenvolvimento dos Módulos de Controle de Voz do *RoboDeck* (Módulos interno e externo);
- ✓ Realização de experimentos;
- ✓ Análise e Discussão dos Resultados.

Na primeira etapa do trabalho foi realizada uma revisão bibliográfica fundamentada em livros e artigos sobre o estado da arte da área de reconhecimento de fala, com o intuito de adquirir uma base teórica das tecnologias de voz, assim como suas aplicações.

Na segunda etapa, notou-se a necessidade de pesquisar os pacotes de reconhecimento e síntese de fala. A partir dos pacotes encontrados, verificou-se a conveniência de estabelecer uma comparação entre eles, com o objetivo de identificar o sistema mais compatível para se obter uma integração do aplicativo escolhido a uma plataforma robótica móvel. Após essa pesquisa e comparação, observou-se que os dois sistemas de reconhecimento de fala mais compatíveis com o propósito do trabalho seriam o *CVoiceControl*, para o desenvolvimento do módulo de voz interno, e o *Voice Search,* para o módulo externo.

Na terceira etapa, produziu-se um estudo relacionado às plataformas de robótica móvel, fazendo também uma comparação entre os robôs móveis fabricados e comercializados, em especial, os que possuem protótipos ou kits para desenvolvimento, com o intuito de facilitar a integração da interface de voz a eles. Verificou-se que entre as plataformas robóticas, somente o *RoboDeck* era 100% brasileiro, sendo desenvolvido e fabricado no interior de São Paulo, na cidade de São Carlos: fato que motivou a escolha desse robô.

Na quarta etapa, desenvolveram-se dois Módulos de Controle de Voz para o *RoboDeck*, um Módulo de reconhecimento de fala interno, integrado diretamente ao MAP (Módulo de Alta Performance) do robô, e, um módulo externo, integrado ao SDK Giga de Testes, possibilitando controlar o robô pela fala, por meio de qualquer dispositivo móvel com acesso à Internet.

Na quinta etapa, foram realizados seis experimentos: cinco utilizaram o módulo de controle de voz externo e um, o módulo interno, permitindo observar e analisar os resultados do comportamento do *RoboDeck* aos comandos de movimentação, via fala.

1.6 Organização do Trabalho

O presente trabalho se inicia com a contextualização do desenvolvimento do reconhecimento de fala e da robótica móvel, relacionando os dois temas que perfazem o Capítulo1. O Capítulo 2 apresenta o levantamento das referências

bibliográficas das tecnologias de voz, abordando os conceitos, aplicações e estado da arte da área. O Capítulo 3 exibe as características e as funcionalidades dos pacotes de Reconhecimento de Fala, com ênfase no software *CVoiceControl* e *Voice Search*. O Capítulo 4 mostra as plataformas da robótica móvel, com ênfase no *RoboDeck*. O Capítulo 5 expõe o desenvolvimento dos módulos de controle de voz interno e externo. O Capítulo 6 descreve os experimentos realizados e a análise dos resultados obtidos. E o Capítulo 7 relata a conclusão do estudo e indicação de trabalhos futuros.

2. TECNOLOGIAS DE VOZ

Há muito tempo o homem busca formas de interagir com as máquinas. As tecnologias de voz proporcionam uma fácil adaptação do usuário ao sistema, aliada a uma grande capacidade de transmitir informações, dispensando qualquer interação física com o equipamento.

Dessa forma as novas gerações de interfaces homem-máquina serão de fácil aprendizado e de alta acessibilidade, destacando o uso das interfaces de voz, que através do reconhecimento e da síntese de fala proporcionam maior mobilidade, permitindo que os olhos ou as mãos estejam livres para realizarem outras tarefas, além de ser uma forma mais intuitiva de comunicação, pois o ser humano já utiliza a voz para se comunicar (ZELTER et al. 1994).

É importante destacar que com o uso das interfaces de voz na máquina ocorrerá um consumo maior de recursos computacionais (memória, processamento e espaço em disco), em virtude da alocação dos processos de reconhecimento e síntese de voz e da necessidade de incorporá-los ao ambiente. Assim, os processos e recursos simples realizados na máquina, como consultar ou visualizar informações, acabam disputando processos e exigindo um grande poder de processamento.

Em 1952 começaram as primeiras pesquisas com máquinas que podiam reconhecer as pronúncias de determinadas palavras. Já nos anos 60, com a descoberta de algumas propriedades da voz e as novas facilidades que os computadores digitais proporcionavam, ocorreu um grande crescimento nas pesquisas sobre o assunto. A partir da década de 70, o desenvolvimento de reconhecimento de voz associou-se bastante à tecnologia dos computadores e passou a ser desenvolvido com base em sofisticados métodos e algoritmos (FECHINE, 2000).

Para que um sistema de reconhecimento de voz homem-máquina seja bem sucedido, é importante o pesquisador ter conhecimento e experiência em diferentes áreas, como:

- ✓ Processamento de sinal: extração de uma informação relevante, em um sinal de fala de maneira eficiente.
- ✓ Física (Acústica): ciência que estuda as relações entre os mecanismos produtores da fala e da audição, pelos quais a fala é percebida.

- ✓ Reconhecimento de Padrões: conjunto de algoritmos usados para agrupar dados, criando padrões de um conjunto de dados que são comparados mediante as características dos padrões.
- ✓ Teoria da Comunicação e Informação: métodos para detectar a presença de um padrão de fala particular, conjunto de algoritmos modernos de codificação e decodificação, incluindo programação dinâmica e algoritmos estáticos, usados para encontrar o melhor caminho correspondente à melhor seqüência de palavras reconhecidas.
- ✓ Linguística: relação entre sons, sintaxe e semântica, consistente nas informações necessárias para o entendimento na comunicação.
- ✓ Fisiologia: estudo do funcionamento do sistema que produz a fala e a audição nos seres humanos. Muitas técnicas modernas tendem a colocar esse tipo de conhecimento, dentro de redes neurais artificiais.
- ✓ Ciência da Computação: estudo de algoritmos eficientes para a implementação de softwares usados em sistemas de reconhecimento de fala.
- ✓ Segundo Rabiner (1978), a área de processamento de voz possui várias aplicações e pode ser dividida em diferentes áreas individuais, conforme suas aplicações ou tecnologias.

O reconhecimento de palavras utiliza o comando de voz para identificar qual ação o sistema deve tomar; tem como característica processar apenas um pequeno trecho de fala, tornando o processamento simples, pois o sistema já sabe quais comandos estão disponíveis ao usuário. Um bom exemplo encontra-se em algumas centrais de atendimento telefônico, em que o usuário pode usar a voz ao invés das teclas do telefone.

O reconhecimento de fala natural ou contínua envolve uma ou mais frases, ou seja, várias palavras que tenham um sentido semântico. A fala reconhecida é processada e transformada em texto: essa tecnologia é muito usada para pronunciar palavras em qualquer editor de texto ou mesmo em um simples e-mail, possibilitando seu uso por pessoas com alguma deficiência física, que impossibilite a digitação com as mãos.

A síntese de voz transforma um texto no formato digital em ondas sonoras, fazendo o processo inverso do reconhecimento da fala, frequentemente utilizado por deficientes visuais, através de programas que replicam o texto digitado em voz alta.

A autenticação utiliza o princípio de que cada indivíduo possui características únicas de voz, capazes de identificá-lo e de controlar o acesso.

2.1 Fala

A fala é um processo tão natural para os seres humanos, que nem sempre se questiona como ela acontece.

Segundo Moreira (2006), existe um modelo de comunicação conhecido como cadeia de fala, o qual descreve a comunicação em três fases: produção, transmissão e recepção da fala.

A fase de produção é formada basicamente por dois processos, sendo que no primeiro a pessoa que está falando, transforma a informação que pretende transmitir em símbolos de uma estrutura lingüística e no segundo processo consolida-os em unidades acústicas. Isso ocorre através da associação dos músculos necessários à geração do fluxo de ar dos pulmões, quando ocorre uma redução desse fluxo através da geometria das cordas vocais, produzindo uma onda de pressão acústica. É importante destacar que a realimentação do sinal é feita pela própria pessoa que está falando, através de seu aparelho auditivo, possibilitando, então, uma avaliação e controle do processo de produção de sua fala.

Na fase de transmissão, o sinal de fala pode sofrer a intervenção de diversos tipos de ruídos, como a fala de outras pessoas ou até distorções em um canal de transmissão, via telefone, entre outros.

Na fase de recepção da fala, a pessoa que está ouvindo recebe a onda de pressão acústica pelo ouvido, tentando extrair as informações nela contida por um processo inverso ao da produção, ignorando os ruídos obtidos na fase de transmissão.

Já Simões (1999), descreve a fala como uma representação dos conjuntos de sons da voz de um ser humano com o intuito de se comunicar. Dessa forma, a fala tem o objetivo de compreender uma determinada informação, a fim de armazenar o que foi pronunciado, transmitindo o sinal de voz por meio de canais e extraindo os

parâmetros ou informações da fala, para manipular o sinal nos mais diversificados fins.

2.2 Reconhecimento de Fala

Um sistema de reconhecimento de fala basicamente tem a função de reconhecer uma sentença falada, respondendo de forma correta ao que está sendo dito. Para isso é necessário comparar os padrões acústicos de voz associados a um conjunto de símbolos que codificam uma mensagem, além de ser essencial que os sinais de entrada que serão comparados não apresentem muitas variações, facilitando o processo de comparação dos padrões. Para esse processo necessita-se também considerar várias formas de reconhecer o sinal de fala do ponto de vista do usuário, pois, de acordo com a pronúncia das palavras e frases, é possível conseguir uma comparação próxima.

O método de reconhecimento depende de características genéricas do sinal de fala que se pretende reconhecer e podem ser classificadas como (FURUI,1989):

- ✓ Espontâneas: falas reconhecidas em situações reais do cotidiano, em que a fala da pessoa é capturada com ruídos diversos, tosse e até mesmo outras vozes, incluindo também mensagens compostas na fala.
- ✓ Contínuas: falas geralmente utilizadas na leitura de um texto, apresentando um vocabulário normalmente grande. São mais difíceis de serem implementadas porque complexo é localizar o início e o fim de cada palavra, devido à tendência das línguas de unirem o último fonema de cada palavra ao primeiro da palavra seguinte.
- ✓ Ligadas ou ativadas: falas usadas com frequência em sistemas de comando e controle por fala, que possuem um conjunto de sinais obtido por meio de uma entrada de áudio, comparada através de um pequeno vocabulário.
- ✓ Palavras isoladas: falas que empregam um processo com pausas, antes e depois de cada palavra e depende de um vocabulário específico e pequeno.

O grau de dependência do locutor, como o estilo de fala ou mesmo a forma de pronúncia da sentença falada, além do tamanho do vocabulário de

reconhecimento de fala, levam a restrições na escolha do hardware e software a serem empregados.

De acordo com Roe et al. (1994), é possível classificar um sistema de reconhecimento de fala pelas características de dependência do locutor, sendo que tais características ajudarão na forma de implementação do sistema de reconhecimento de fala.

Os sistemas dependentes de locutor são caracterizados por serem treinados para obedecerem às características da voz dos seus usuários. Desse modo, é importante utilizar vários locutores com diferentes características lingüísticas como idade, sexo, nível sociocultural, entre outros fatores, para o treinamento. Porém, se outro locutor tiver um timbre de voz parecido com os padrões acústicos, mesmo que ele não tenha participado do treinamento, o sistema poderá reconhecer algumas palavras ou comandos. Assim, quanto mais locutores forem treinados no sistema, maior será a chance de reconhecer um locutor genérico.

Os sistemas de reconhecimento de fala independentes do locutor podem ser definidos como aqueles que não estão presos às características específicas da voz do usuário. São capazes de reconhecer a fala de qualquer locutor, sem a necessidade de obter padrões de referências de modelos acústicos, através de treinamento.

Segundo Campbell (1997), existem alguns fatores externos a um ambiente ou humanos que contribuem para erros em um sistema de reconhecimento de fala:

- ✓ Erro de locução ou leitura das frases;
- ✓ Estado emocional;
- ✓ Variação da posição do Microfone;
- ✓ Ambiente Acústico inadequado ou inconsistente com ruídos;
- ✓ Erro de sincronia de microfones;
- ✓ Idade do locutor, já que o trato vocal pode sofrer alterações com o tempo.

Mencionados fatores são importantes e dependendo da situação, mesmo que o algoritmo de reconhecimento de fala seja de boa qualidade, o erro humano pode limitar o desempenho.

Portando, um bom projeto de reconhecimento de fala deve procurar diminuir ao máximo possíveis erros externos e utilizar técnicas que possam representar eficientemente as características vocais das sentenças.

2.2.1 Métodos para o Reconhecimento de Fala

Basicamente o reconhecimento de fala funciona a partir da conversão do sinal acústico (voz) em um sinal digital de áudio, através de um hardware (placa de som e microfone).

Com o passar dos anos, os métodos de reconhecimento de fala evoluíram, admitindo que os usuários efetuem ditados de forma contínua ao computador, mas, apesar disso, as taxas de acerto não passam dos 95%.

De acordo com Hugo (1995), existem dois métodos para o reconhecimento de fala, o global e o analítico, que podem ser complementares entre si, a fim de solucionar o problema de reconhecimento.

O método global de reconhecimento de palavras que emprega a técnica de comparação global das palavras, para reconhecer as diversas formas de referência armazenadas, e, o seu tratamento acústico é bem simples, pois as mensagens identificadas são consideradas como uma forma única, não ocorrendo problemas de segmentação, não devendo ser utilizado em grandes vocabulários ou em falas contínuas.

O método analítico segmenta a mensagem, dividindo os elementos, fonemas, sílabas e outros, identificando cada elemento e reconstruindo a frase pronunciada por etapas fonética, léxica e sintática.

Tatham (1995) sugere outro método de reconhecimento de fala, realizado de quatro formas básicas, em que o sistema recebe uma entrada de voz e traduz em ortografia normal de acordo com o idioma e a gramática utilizada. As quatro formas básicas são:

Baseada em Modelos: o sistema recebe uma entrada de voz e compara com os modelos já armazenados, tentando encontrar um que mais se aproxime da entrada de voz.

Baseada em Conhecimento: utiliza técnica de inteligência artificial e raciocínio baseado em casos, para simular o conhecimento humano a fim de reconhecer a voz.

Estocástica: utiliza as propriedades da estatística da ocorrência de sons de voz individuais.

Conexionista: aplica a teoria dos grafos que trabalha com redes de um grande número de nós simples interconectados, representando fonemas e suas conexões, treinados para reconhecer a fala.

Já Pizzolato (2001) cita três métodos distintos para o reconhecimento de fala: o acústico, o reconhecimento de padrões e a inteligência artificial.

O acústico é baseado na teoria da acústica e fonética, utilizando a existência de um conjunto finito e distinto de unidades fonéticas em uma linguagem, através das propriedades presentes na fala.

O reconhecimento de padrões trabalha com duas fases: a primeira consiste no treinamento e a segunda, no reconhecimento. No treinamento é armazenada uma quantidade considerável de padrões, sendo possível obter as variações de cada unidade sonora de um grupo de usuários. No reconhecimento os padrões de entrada são comparados com os padrões armazenados.

A inteligência artificial reúne os conhecimentos acústicos, léxico, sintático, semântico e pragmático. O conhecimento de acústica utiliza evidências sonoras para detectar unidades fonéticas; o léxico permite combinar evidências acústicas de tal forma a construir palavras de acordo com um dicionário; o sintático possibilita construir sentenças a partir de combinações corretas de palavras; o semântico tem a capacidade de entender o domínio da frase e verificar a consistência da mesma; e, por fim, o pragmático permite resolver ambiguidades.

Devilliers (1996) estabelece um nível de interação entre os sistemas compostos de reconhecimento de voz, assim como a distinção de suas partes, conforme mostra a Figura 1 abaixo.

Figura 1 – Níveis do sistema de compreensão da linguagem falada (DEVILLIERS, 1996)

Observa-se que para compreender uma frase falada por uma pessoa, o software reconhecedor precisa resolver e interpretar as análises pragmática, semântica, sintática e morfológica. Vale observar que o software depende integralmente do módulo intermediário *Middleware,* que faz a analise léxica e de outros serviços de hardware, para conseguir a análise acústica.

Segundo Hugo (1995), a tarefa de reconhecimento de falas deve ser realizada em três fases, sendo que a primeira é a aquisição do sinal de voz, a segunda é o pré-processamento, consistente na extração de parâmetros e o terceiro é o processamento que fará o reconhecimento do padrão. Dessa maneira, cada uma dessas fases realiza transformações nos dados recebidos, conforme ilustra a Figura 2.

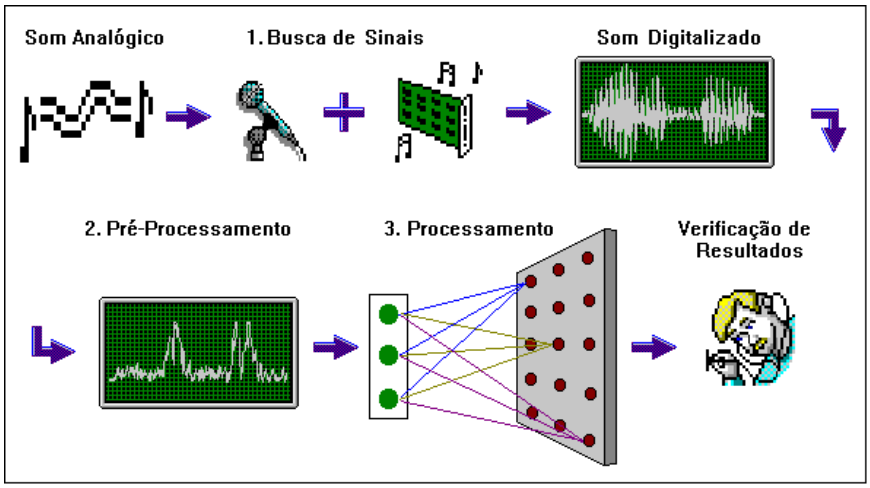


Figura 2 – Fases empregadas para o reconhecimento de fala (HUGO, 1995)

Na primeira fase faz-se a captura do sinal de voz, ou seja, a fala humana por meio de um microfone, o qual está ligado em uma placa de som digitalizadora, que converte em sinal digital, pois para o computador armazenar o som ou manipular os sons na memória, o formato da informação precisa ser alterado.

Na segunda fase é realizado o pré-processamento de sinais, responsável por gerar o vetor característico do padrão, também conhecido como *feature number,* onde é realizada a extração de parâmetros, eliminando os sinais redundantes ou insignificantes, por meio de seleção ou transformação, visando a organizar ou ajustar os sinais capturados, deixando-os adequados para a fase de processamento.

Na terceira e última fase é realizado o processamento, reconhecimento de padrões com a interpretação dos dados, permitindo a extração das características relevantes desses objetos e criando um agrupamento de objetos semelhantes dentro de uma determinada classe ou categoria.

2.2.2 Reconhecimento de Padrões de Fala

Um dos pontos-chave no reconhecimento de fala é a comparação dos padrões, momento em que são confrontados os parâmetros ou características de um sinal de voz capturado com os padrões de referências, já armazenados no sistema. O objetivo é determinar o grau de similaridade entre dois padrões ou a distância entre eles.

Segundo Doddington (1985), quando um locutor pronuncia uma palavra ou uma frase, durante o treinamento, um conjunto de vetores de padrões é estabelecido e os segmentos acústicos são convertidos em características representativas de cada sentença. A partir daí, o vetor de padrões de teste é comparado com todos os padrões de referência armazenados, para escolher o mais parecido, produzindo uma saída correspondente àquela referência. Se no momento da comparação existirem outras referências semelhantes, o sistema pode adiar a decisão e solicitar que o locutor repita seu padrão.

Chengalvarayan (1999) descreve que um sistema de reconhecimento de fala consiste na extração e seleção dos parâmetros vocais, seguido pelo processo de classificação, com o uso de um vetor de características, que deve conter toda informação relevante do sinal de voz, ignorar as informações irrelevantes, e ainda, possuir uma pequena dimensão, objetivando minimizar o tempo computacional na etapa de classificação.

Os sistemas de reconhecimento têm usado diversas características, como formantes, particularmente importantes na determinação da fala, intensidade e coeficientes obtidos a partir dos métodos de extração de parâmetros. De certo modo, a formação das vogais se dá praticamente pela alteração das regiões formânticas e a escolha das características implica a qualidade do reconhecimento.

Uma das mais importantes técnicas para análise de voz, segundo Vieira (1984), é a análise dos Coeficientes por Predição Linear (LPC): usada para estimar os parâmetros básicos da voz, fornecendo estimativas extremamente corretas

desses parâmetros, com uma boa velocidade computacional. Eles são estimados por minimização do erro quadrático entre a amostra corrente e a sua predição. O princípio da predição linear é a de que um valor de uma amostra pode ser aproximado por combinação linear aos valores das amostras anteriores, com base na correlação entre elas.

Muitos sistemas de reconhecimento de fala têm utilizado os parâmetros obtidos na análise LPC, devido às vantagens proporcionadas em termos de generalização da envoltória espectral e sua habilidade para modelar os picos espectrais.

Fechine (2000) aponta que os métodos de reconhecimento de voz, de modo geral, diferenciam-se na forma como os parâmetros extraídos são utilizados na construção dos padrões. Assim, podem ser divididos em dois grupos: os paramétricos e os estatísticos.

Nos métodos paramétricos, a comparação de padrões é baseada em medidas de distância das amostras. A principal técnica utilizada por esse método é a Quantização Vetorial (QV), que é a conversão de um vetor de características em um código relacionado a vetores de mesma dimensão previamente treinados. Essa técnica é muito útil no reconhecimento de fala, pois proporciona uma eficiente taxa de redução de dados, dentro da parametrização de voz, otimizando o espaço de memória usada. A principal vantagem da QV é a produção de um dicionário, que serve para identificar a similaridade entre as elocuções de uma mesma sentença e as diferenças entre os locutores.

Nos métodos estatísticos, a criação dos padrões é realizada através de modelos como os de Markov Escondidos (HMMs). Nesse método não é realizada uma comparação direta de padrões e a decisão é feita através de cálculos de probabilidades associadas aos modelos. Os parâmetros são extraídos com o auxílio da teoria das probabilidades e são representados por modelos estocásticos, obtendo-se uma redução implícita de dados.

Um sistema de reconhecimento automático de fala pode ser implementado, usando um ou até mesmo uma combinação dos dois métodos citados. Fechine (2000) utiliza técnicas paramétricas e estatísticas, para modelagem das características vocais dos locutores: a medida de distorção obtida, através do uso da Quantização Vetorial, seguida da probabilidade obtida pelo HMM, empregado como parâmetro de refinamento para o processo de identificação.

2.2.3 Reconhecimento Automático de Fala

O reconhecimento automático de fala (*Automatic Speech Recognition* – ASR) é um processo de extração automática das informações linguísticas de um sinal de voz, baseado na identificação de padrões de fala, obtidos através do treino do sistema pelo orador. Portanto, é importante tomar certos cuidados na preparação de um sistema para reconhecimento da fala, pois, além da relevância da utilização de um orador conhecido com um vocabulário finito, é possível obter padrões diferentes para ambientes diferentes.

Para Hugo (1995), as informações linguísticas contidas em um sinal de voz devem estar codificadas de forma que, mesmo ocorrendo uma grande variação do sinal, causada pelo ambiente e pelo locutor, praticamente não interfira na percepção da informação pelo homem.

Em testes aplicados por Hugo (1995), notou-se que a acurácia no reconhecimento de fala chega até 90% de certeza, na maioria dos casos em condições triviais de uso.

A acurácia de uma estimativa é uma medida da correlação entre o valor estimado e os valores das fontes de informação, ou seja, mede o quanto a estimativa obtida é relacionada com o "valor real" do parâmetro. Ela informa a proximidade do valor estimado ao valor real e fornece a "confiabilidade" daquela ao valor.

Uma condição trivial de uso pode ser determinada quando um orador realiza o treino, pronunciando as palavras sem interrupção, de modo que a captura do som seja feita em um lugar onde não exista interferência de outros sons. Entretanto, em situações onde o vocabulário é pequeno e o microfone não reconhece os ruídos, o reconhecimento pode ultrapassar os 98% de acurácia.

Na fala contínua, o processo de reconhecimento é mais difícil, pois quanto maior for o vocabulário, maiores serão as semelhanças entre as palavras e seus fonemas, tornando mais complexa a análise das palavras realmente ditadas.

De acordo com Moreira (2006), o reconhecimento automático de fala possui dois módulos fundamentais: um correspondente à análise do sinal e o outro, à sua classificação.

No módulo de análise, é feita uma conversão do sinal de entrada, tornando-o um sinal adequado para o processo de classificação. Os componentes do vetor de

características devem ser formados com pouca variação das características dentro de cada classe, mas que tenha uma grande variação entre as classes diferentes, para que possa ser feita uma discriminação entre elas.

Além disso, esse módulo gera uma representação menor que o sinal original, permitindo o uso de algoritmos mais poderosos, que normalmente ocupam mais espaço, mas proporcionam uma aceleração no processo de classificação.

É importante ficar atento ao intervalo de tempo entre as observações do sinal, evitando perder as informações entre os vetores de características sucessivos.

No módulo de classificação, realiza-se uma conversão do vetor de características numa sessão de símbolos linguísticos, que pertencem a um vocabulário Γ (Gama), relacionado com as classes padrão, (MOREIRA, 2006):

2.3 Aplicações da Tecnologia de Reconhecimento de Fala

Existem inúmeras aplicações que aplicam a tecnologia de reconhecimento de fala, tanto nos computadores quanto nos celulares. Tais como:

- ✓ Tetraplégicos que utilizam o comando de voz para manipular o computador.
- ✓ Motoristas que usam o comando de voz para mudar a música do rádio, ou falar o endereço de seu destino para um GPS sem ter de digitar, permanecendo atento ao trânsito.
- ✓ Alguns escritores que ditam seus livros para um computador, o qual transcreve as palavras com precisão.
- ✓ A Google que tem investido fortemente em comandos de voz para seu sistema operacional móvel, o Android.

Por outro lado, a oferta de sistemas de reconhecimento de fala em Português para celulares e computadores é quase nula. O Android da Google, por exemplo, cuja tecnologia de reconhecimento de voz (*Voice Actions*) só interpreta comandos em Inglês.

Maciel (2007) destaca as aplicações de reconhecimento de fala, em duas amplas áreas: a telecomunicações e a de negócios.

Na área de telecomunicações, é citada a interface de reconhecimento de voz para sistemas de menu de telefones, em que o usuário pode falar a opção desejada,

ao invés de pressionar botões com o número correspondente. Também as requisições de informações de lista telefônica por voz, que proporcionam aos usuários uma forma natural de interação com uma base complexa de informação.

Na área de negócios, destacam-se os sistemas de reconhecimento de fala para preencher formulários, geralmente usando vocabulários de tamanhos pequenos, entre dez a duzentas palavras, o que torna esse tipo de sistema bastante eficaz, e, de reserva ou consulta de passagens aéreas, situação em que o usuário pode consultar a disponibilidade de voos, falando o nome da cidade, dia e horário.

3 BIBLIOTECAS DE INTERFACE DE VOZ

O estado da arte em reconhecimento e síntese de voz tem evoluído bastante nas últimas décadas, tornando viável integrar a voz em diversos tipos de aplicações.

A existência de diversos sistemas de reconhecimento e síntese de voz exige que se estabeleça uma comparação para identificar o sistema mais compatível na integração do aplicativo escolhido a uma plataforma robótica móvel.

Para integrar um aplicativo de reconhecimento de fala ao MAP (Módulo de Alta Performance) do robô móvel *RoboDeck,* era necessária uma biblioteca de interface de voz que possuísse um sistema de reconhecimento de fala compatível com o sistema operacional Linux, que tivesse código aberto para mudanças e que reconhecesse falas em Português. Já para integrar um aplicativo de reconhecimento de fala, ao SDK chamado “Giga de Testes”, era necessária uma biblioteca de interface de voz que pudesse ser acessada através de um celular ou PC, que reconhecesse a fala de qualquer locutor sem a necessidade de um treinamento e que fosse compatível a qualquer plataforma.

Partindo dessas características, foram encontradas as seguintes bibliotecas:

3.1 XVoice

O *XVoice* permite que o usuário faça um ditado, utilizando fala contínua e comando de voz para a maioria dos aplicativos. Para converter a fala dos usuários em texto, ele usa o IBM *ViaVoice*, que é distribuído separado do software.

Em modo de ditado, o *XVoice* envia o texto direto para o aplicativo que o usuário estiver utilizando. No modo de comando, ele reconhece a fala e executa-o conforme o comando pré-definido pelo usuário.

A desvantagem desse sistema é a IBM não disponibilizar mais os pacotes que permitiam a sua instalação em Sistemas Operacionais Linux. Portanto, sem o pacote *ViaVoice* SDK, que era distribuído separado do XVoice, o programa não tem utilidade, pois fica sem o módulo para reconhecimento de fala (CREEMER D et al. 1999).

A Figura 3 mostra uma sessão típica do *XVoice*, em que a fala reconhecida é visualizada no painel direito. Os vocabulários atualmente ativos são listados à

esquerda. O aplicativo para onde os comandos estão sendo enviados e aparecem listados na parte superior (CREEMER D et al. 1999).

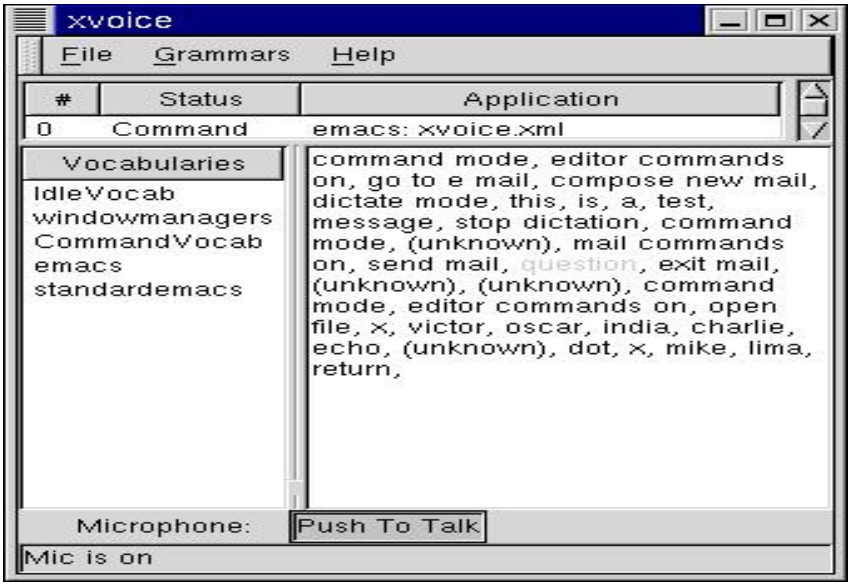


Figura 3 – Tela do XVoice
Fonte: <http://xvoice.sourceforge.net/>

3.2 Festival Speech

O *Framework Festival* é um projeto desenvolvido pela *University of Edinburgh*, consistente na conversão de frase no formato de texto em voz. É um software capaz de fazer um discurso artificial no lugar de um ser humano real, além de ser o mais completo sistema de síntese de uso geral disponível, com licença gratuita, usado por sites de pesquisas e outros projetos em todo o mundo.

O sistema é todo desenvolvido em C++ sob licença do X11[1], que permite uso comercial e não-comercial de forma irrestrita.

Por outro lado, existem dois fatores negativos: o primeiro é a língua utilizada, pois no site oficial estão presentes pacotes de dois narradores, sendo um para o Inglês americano/britânico e um para o Espanhol, não possuindo nenhum narrador para o Português; o segundo é o projeto Festival não possuir sistema de reconhecimento de voz incluso, limitando-se à síntese de voz (CALEY R et al, 1999).

[1] X11 é uma licença utiliza em software livre, que permite a reutilização de software licenciado em programas livres ou proprietários.

A Figura 4 representa uma demonstração do Festival, em que o usuário pode sintetizar suas próprias sentenças *on-line* através do Site, que se destina a permitir um controle mais rigoroso dos resultados de diferentes métodos de síntese. As vozes são faladas no presente, com uma indicação da quantidade de dados de fala usados para construir a voz (CALEY R et al, 1999).

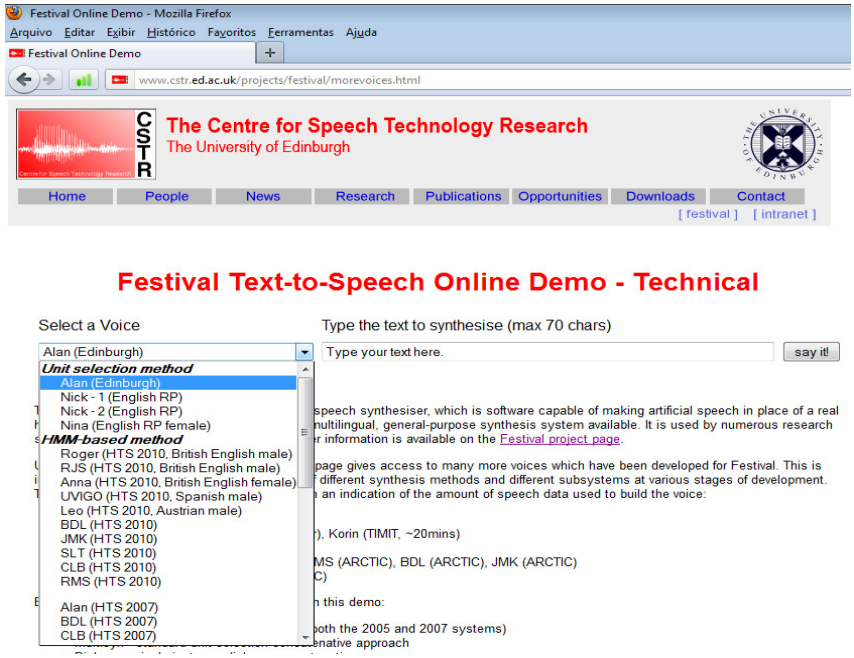


Figura 4 – Tela de demonstração interativa do sintetizador de voz "*Festival on-line*"
Fonte: <http://www.cstr.ed.ac.uk/projects/festival/onlinedemo.html>

3.3 Dragon Naturally Speaking

O *Dragon Naturally Speaking* é um dos primeiros sistemas de reconhecimento de voz natural, desenvolvido pela Nuance Communications Inc., detentora de um monopólio no ramo de reconhecimento de voz, já que alguns concorrentes como a Philips e a Lernout & Hauspie deixaram o mercado, além de possuir também a propriedade do IBMViaVoice.

Através do *Dragon* o usuário consegue criar documentos, relatórios ou mensagens apenas falando. Possui atalhos por voz, o que permite criar e-mail, compromissos, agenda e pesquisar na *Web* usando comandos de voz simples.

O programa tem suporte para a língua inglesa ditado em ritmo de fala normal, cujo processo de treinamento demora em média 4 minutos. Assim, o usuário estará apto a ditar o texto para a máquina com se estivesse numa conversa normal, tendo as palavras ditadas 99% de precisão (NUANCE, 2011).

A Figura 5 mostra a tela do *DragonPad*, um editor de texto do *Dragon Naturally Speaking Preium Editon* 11.

Figura 5 – Tela do *DragonPad do Dragon*
Fonte: <http://voice-recognition-software-review.toptenreviews.com/3060-screenshots.htm>

O *Dragon* também permite criar várias contas de usuário, que podem ser adaptadas a estilos específicos de falar. A Figura 6 mostra o *Manage Users,* um programa que gerencia os usuários de *Dragon Naturally Speaking Premium Edition 11.*

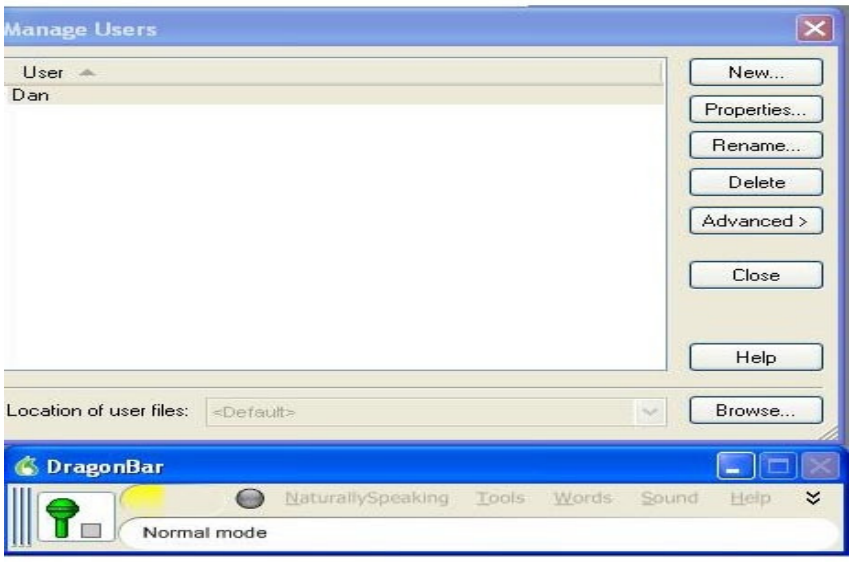


Figura 6 – Tela *do Manage Users do Dragon*
Fonte: http://voice-recognition-software-review.toptenreviews.com/3060-screenshots.htm

As desvantagens do Dragon Naturally Speaking são o não reconhecimento da Língua Portuguesa, a incompatibilidade com o Sistema Operacional Linux, a não gratuidade da licença para seu uso e a inexistência de código aberto (NUANCE, 2011).

3.4 Praat

O *Praat* é um programa para análise de fonemas com recursos para avaliação de áudio. Utilizado a partir de um arquivo de voz gravado, cria uma imagem de espectrograma, baseada nas variações de timbre da gravação do arquivo de áudio.

Possui versões para Windows, Linux, Macintosh e demais sistemas UNIX. Além disso, a licença é gratuita e o código-fonte está disponível para modificações. O programa possui documentação consistente, com explicações das principais análises sobre áudio: a espectral (espectrogramas) e a formal, entre outras. Para o estudo de padrões fonéticos, o sistema oferece técnicas de inteligência artificial como redes neurais.

Seu único ponto negativo é não haver suporte ao reconhecimento de fala (BOERSMA, 2001).

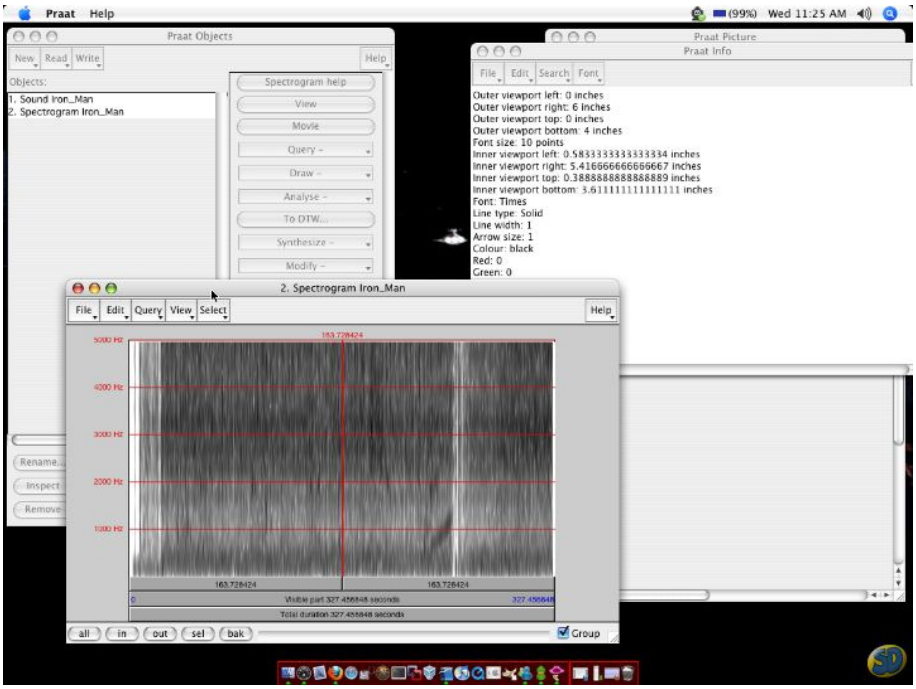


Figura 7 – Tela do *Praat* 5.2.44 para MAC OSX
Fonte: http://www.superdownloads.com.br/imagens/telas/praat-mac-117673,2.jpg

3.5 Transcriber

O *Transcriber* é uma ferramenta específica para a transcrição e segmentação de gravações de voz de longa duração. Realiza uma transcrição dos sinais de áudio em formato de texto, utilizando dados em formato estruturado no padrão XML com suporte à transcrição em múltiplas línguas.

O programa possui código aberto para mudanças e permite carregar arquivos de áudio de diferentes formatos como WAV, MP3, OGG, entre outros. Para a sua análise, permite selecionar determinada parte do áudio, distinguindo música de voz, admitindo ainda que o usuário modifique aspectos do texto depois de terminado o processo, como tipo de letra, cor e formato (GEOFFROIS et al. 2011).

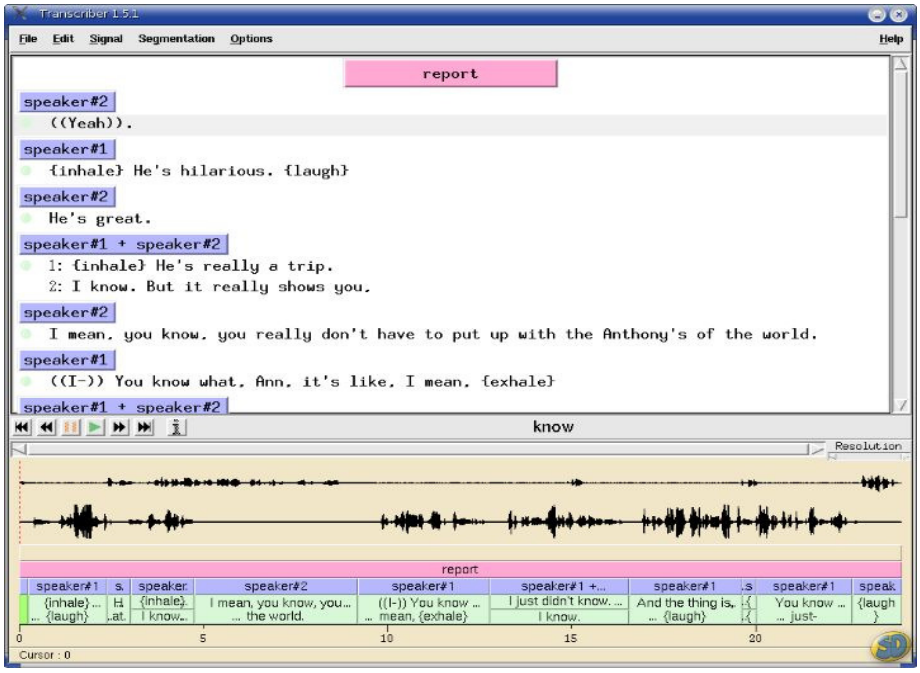


Figura 8 – Tela do editor de anotação do *Transcriber*
Fonte: <http://www.superdownloads.com.br/imagens/telas/transcriber-126087,2.jpg>

3.6 IBM ViaVoice 9

O IBM *ViaVoice 9* é um software de reconhecimento e sintetização de voz da IBM com suporte a diversas línguas, como o Português do Brasil, para sistemas Operacionais da Microsoft. Ele permite que o usuário fale ao invés de digitar e escute ao invés de ler um texto. Assim, usando a fala, o usuário pode ditar textos nos aplicativos da Microsoft como o Outlook, Excel e Word.

O *ViaVoice 9* executa também comandos de voz, permitindo que o usuário controle o Windows, por meio de instruções dadas ao microfone, como por exemplo, abrir, fechar, executar programas etc.

Uma característica muito interessante desse software está na sua capacidade de aperfeiçoamento, conforme o seu uso contínuo, sendo que seu vocabulário básico para Língua Portuguesa tem aproximadamente 60 mil palavras, ou seja, à medida que o usuário dita uma nova palavra, ela se armazena no banco de dados, aumentando o número de palavras de seu vocabulário.

O tempo de treinamento leva em média 15 minutos, para a leitura de um texto padrão. Em seguida, pode ser feito o ditado, citando as palavras e a pontuação.

A versão IBM *ViaVoice 9* não é compatível com o sistema Operacional Linux e o código não é aberto (IBM, 2008). Por algum tempo, a IBM disponibilizou a versão do *ViaVoice* para o Linux, mas, interrompeu a disponibilização e o suporte, tornando inviável a sua utilização em Sistemas Operacionais que não fossem os da Microsoft.

A *Oralux Association* lançou o *Voxin*, que foi uma forma de vender licenças individuais do IBM *ViaVoice* para o Linux, entretanto, com um custo maior ao que era vendido pela IBM. O software era parecido com o da IBM, porém a *Oralux Association* acabou abandonando o projeto de expandir o programa para novos idiomas, por motivos financeiros.

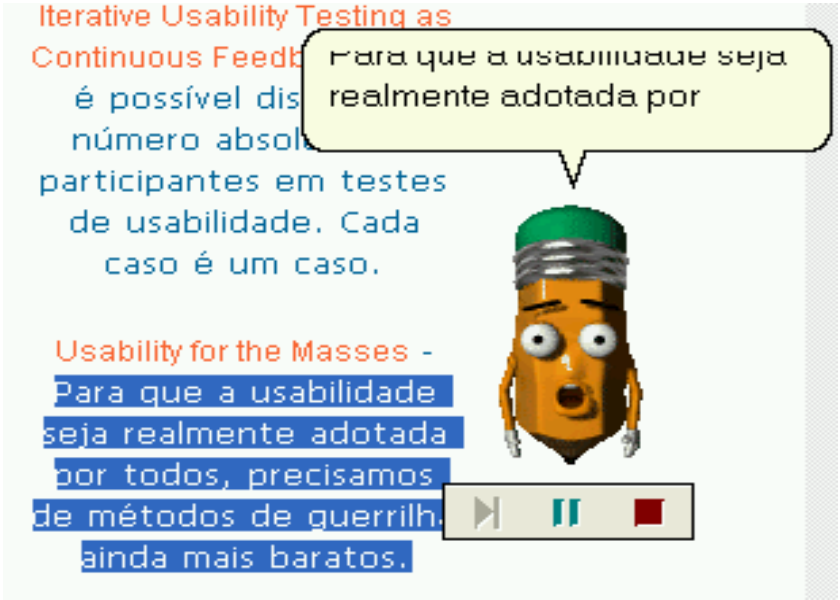


Figura 9 – Tela do assistente do IBM ViaVoice 9,0 para Windows
Fonte: <http://usabilidoido.com.br/assistentes_do_ibm_viavoice.html>

3.7 CVoiceControl

O *CVoiceControl* é um programa de reconhecimento de fala, compatível com o sistema Operacional Linux, permitindo, dessa maneira, o usuário mapear comando de voz para comandos LINUX. O programa automaticamente detecta a entrada da fala no microfone e processa o sinal: se o reconhecimento tiver sucesso, o comando correspondente é executado. É importante destacar que o *CVoiceControl* é uma atualização do *KVoiceControl "KDE VoiceControl*" (KIECZA 2002).

Ele não possui um dicionário com comandos preestabelecidos, diferentemente dos outros sistemas de reconhecimento, nos quais existe na maioria dos casos, a limitação de só reconhecer comandos em Inglês. Nele o próprio usuário grava uma sentença e faz sua associação a um comando em LINUX, independente da língua usada na sentença.

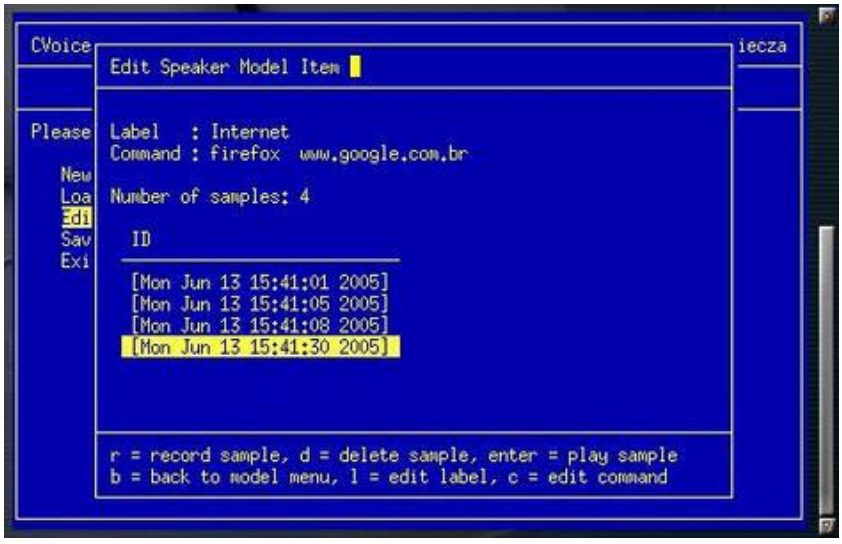


Figura 10 – Tela do *CVoiceControl* exibindo o comando associado à sentença

Além disso, o *CVoiceControl* foi desenvolvido em linguagem C e possui código aberto à mudanças.

Seu único problema é não conter uma boa documentação, pois o projeto foi descontinuado pelo seu criador Daniel Kiecza, assim como as pesquisas e o aperfeiçoamento, (o programa não tem nenhuma atualização desde 11 de novembro de 2002).

O *CVoiceControl* utiliza três módulos:

- ✓ *microphone_config;*
- ✓ *model_editor;*
- ✓ *cvoicecontrol.*

Ao iniciar o *CVoiceControl* pela primeira vez, o usuário deve calibrar o microfone para o reconhecimento de voz, usando o módulo *microphone_config.* Depois que a placa de som estiver preparada, o usuário deve criar os modelos com as sentenças faladas e associá-las aos comandos que deseja realizar no Linux, através do *model_editor:* esses são os principais objetos necessários para o processo de reconhecimento de fala (KIECZA, 2002).

3.8 Voice Search

O *Voice Search* ou Buscar por Voz é um sistema de reconhecimento de fala desenvolvido pela *Google*, podendo ser utilizado em qualquer microcomputador que tenha o navegador *Chrome* instalado ou em celulares que possuam o Sistema Operacional Android. Recentemente a Google lançou a versão do *Voice Search* 2.0.0, também compatível com *iPhone, iPod touch* e *iPad*, com Sistema Operacional iOS 4.2 ou posterior.

Ele não precisa treinar um locutor para o seu uso, isto é, pode ser utilizado por vários locutores, com uma precisão dos resultados bastante eficiente, reconhecendo palavras complicadas e também termos compostos por mais de uma palavra.

É importante destacar que esta biblioteca requer conexão com a Internet para funcionar, pois ela reconhece a fala por meio de um serviço fornecido pela *Google*.

Atualmente o *Voice Search* consegue reconhecer mais de 28 Línguas diferentes, entre elas variações do Inglês (britânico, sul-africano e estadunidense), do Espanhol (europeu, mexicano, argentino e do restante da América Latina) e o Português do Brasil.

Figura 11 – Tela do *Voice Search* sendo executado, pelo ícone do microfone

Para acionar seu recurso de reconhecimento de fala, basta iniciar o navegador e clicar no ícone do microfone, o qual exibirá, conforme a Figura 11, um alerta “Fale agora”. Depois da fala da sentença pelo usuário, ela será reconhecida e os resultados imediatamente apresentados. Caso o usuário deseje falar uma nova sentença, ele deverá tocar novamente no ícone de microfone (POZZEBON, 2012).

3.9 Comparações entre as bibliotecas de interface de voz

A tabela 1 mostra as bibliotecas de interface de voz e faz uma comparação entre sistemas operacionais compatíveis, bibliotecas que possuem código aberto para mudanças, idiomas de reconhecimento ou síntese de fala, tecnologia de voz utilizada, e as últimas atualizações disponíveis. Os dados contidos nesta tabela foram estabelecidos de acordo com a documentação fornecida pelos fabricantes.

Tabela 1 – Bibliotecas de interface de voz

Bibliotecas de interface de Voz	Sistemas Operacionais	Código Aberto	Idiomas	Tecnologias de Voz	Últimas Atualizações
XVoice	Linux	Sim	Inglês	Reconhecimento de fala	XVoice0,9,6-2 ano de 2007
Festival Speech	Linux, OSX, Windows.	Sim	Inglês e Espanhol	Síntese de fala	Festival Speech versão 2.1 Novembro de 2010
Dragon Naturally Speahing	Microsoft Windows	Não	Inglês	Reconhecimento de fala	Dragon NaturallySpeaking 11,5 agosto 2011
Praat	Windows, Linux, Macintosh	Sim	Inglês	Análise de fonemas	*Praat 5.2.44* 23 setembro de 2011
Transcriber	Windows XP, Mac OS X e Linux	Sim	Francês, Inglês Russo, Árabe e Chinês	Transcrição e segmentação de gravações de voz	TranscriberAG julho de 2011
IBM Via Voice	Windows e Mac OS X	Não	Inglês Português	Reconhecimento e Síntese de fala	IBM Via Voice 11 Português Millennium Edition - Standar Ano de 2010
CvoiceContrl ol	Linux	Sim	Qualquer	Reconhecimento de fala	CVoiceControl-0.9Alpha novembro de 2002
Voice Search	Windows, iOS 4.2 e Android	Não	Qualquer	Reconhecimento de fala	Voice Search 2.0.0 23 maio de 2012

Analisando a tabela, observa-se que as bilbiotecas *XVoice, Dragon Naturally Speaking, IBM ViaVoice, CVoiceControl e Voice Search* são as únicas que possuem a tecnologia de reconhecimento de fala.

Entre elas, apenas as bibliotecas IBM *ViaVoice*, *CVoiceControl* e *Voice Search* reconhecem comandos de fala em Português, mas somente o *CVoiceControl* é compatível com o Sistema Operacional Linux, configurando-se, portanto, a biblioteca mais adequada para a integração interna com a plataforma robótica móvel.

Para a integração externa ao SDK da plataforma robótica, a biblioteca com melhor compatibilidade foi o *Voice Search*, uma vez que pode ser usado por vários locutores sem a necessidade de treinamento. Pelo fato de ser compatível com os sistemas operacionais Windows e *Android*, apresenta-se como a biblioteca ideal para controlar o robô pela fala, através de um dispositivo móvel como os celulares.

4. PLATAFORMAS DE ROBÓTICA MÓVEL

A robótica é uma área em grande expansão, que movimentando bilhões de dólares, gera novos mercados de trabalho. Entre as suas áreas, pode-se destacar a robótica móvel, atualmente utilizada na criação de robôs com várias finalidades como na indústria, nas operações militares, na vigilância, no entretenimento e até mesmo em trabalhos de limpeza.

Um robô móvel é um dispositivo mecânico montado sobre uma base móvel, que pode ser usado em ambientes terrestres, aéreos ou aquáticos. Ele se move através de comandos de um sistema computacional, equipado com sensores e atuadores que permitem interagir com o ambiente (BEKET, 2005).

O controle do robô móvel, geralmente é realizado através de comandos por meio de uma interface baseada em teclado ou botões. Dessa forma, ao integrar uma interface de reconhecimento de fala em um robô móvel, o meio de comunicação entre o homem e o robô será mais prático e cômodo.

A existência de diversas plataformas de robótica móvel propicia uma comparação entre os robôs móveis fabricados e comercializados no mundo, em especial, os que possuem protótipos ou kits para desenvolvimento, porque facilita integrar-lhes uma interface de voz.

4.1 IRobot Create

O Robô *Create* foi criado pela empresa IRobot, com base no Robô *Roomba*. Trata-se de um kit completo para o desenvolvimento robótico, contendo uma plataforma de programação robótica, onde desenvolvedores e estudantes podem acessar vários sensores e atuadores do robô, por meio de uma interface aberta. Essa plataforma possibilita controlar o robô por meio de aplicativos robóticos feitos pelos usuários, bem como instalar novos componentes, como braços, usando as interfaces de conexões do Create (IROBOT, 2006).

É possível controlar o Robô *Create*, através de pequenas rotinas já prontas ou programar um novo comportamento, aplicando as linguagens de programação C ou C++, mediante uma interface serial do computador.

Esse robô também possui:

- ✓ 32 sensores integrados;

- ✓ 2 rodas motorizadas;
- ✓ 10 comportamentos programados;
- ✓ 1 porta de entrada/saída expansível para sensores e atuadores personalizados;
- ✓ 1 zona de carga com pontos de montagem.

Figura 12 – Robô móvel *Create* da empresa IRobot
Fonte:< http://uncrate.com/stuff/irobot-create>

4.2 LEGO *Mindstorms* NXT

O robô *MINDSTORMS* NXT é um produto da LEGO, que possui um bloco de controle computacional inteligente, ideal para os desenvolvedores ou estudantes personalizarem o robô com novos modelos de programação.

O Robô combina a versatilidade do sistema de construção LEGO com as novas tecnologias robóticas, pois vem com 612 peças, permitindo que o usuário crie diversas configurações robóticas.

Seu bloco inteligente NXT LEGO é formado por um microcontrolador ARM 32 bits, um grande display de matriz, 4 portas de entrada e 3 de saída, Bluetooth, além de USB.

Através do *Toolkit* do LabVIEW, o usuário pode desenvolver programas e descarregar diretamente no bloco NXT. Com conversores analógico/digital (ADCs) de 10 bits como entradas, o bloco pode se comportar como uma unidade de aquisição de dados, capaz de controlar mais de 3 motores e contadores de rotação.

Encontram-se também no robô, 3 servomotores e 4 sensores, sendo 1 sensor de ultrassom, 2 de toque e 1 de Cor, o qual contém 3 funcionalidades: distinguir cores; configurar luz e funcionar como lâmpada (LEGO, 2009).

Figura 13 – Robô móbel *Mindstorms* NTX 2,0 da empresa LEGO
Fonte:< http://www.tuvie.com/lego-mindstorms-nxt-2-0-review/>

4.3 *MobileRobots* Inc Pionner 3

O robô Pionner 3 é uma plataforma robótica móvel da empresa *MobileRobots*, o qual possui um kit para desenvolvedores denominado *Pioneer SDK* (*Pioneer Software Development Kit)*, formado por um conjunto completo de aplicativos robóticos e bibliotecas que aceleram o desenvolvimento de projetos de robótica.

Ele contém 4 rodas, pode atingir velocidades de 0,8 metros por segundo, transportar uma carga útil de até 12 kg e pode ser utilizado em projetos ao ar livre ou em pisos irregulares.

Sua arquitetura é do tipo cliente-servidor, ou seja, composto por um microcomputador “cliente”, que funciona com o Sistema Operacional Linux e um microcontrolador “servidor”, que possui um sistema próprio, chamado ARCOS, cuja comunicação realiza-se através de uma interface serial.

Dessa maneira, os aplicativos robóticos feitos pelos usuários são carregados no cliente (microcomputador que possui a plataforma Linux), e, por meio da interface serial, o cliente envia e recebe dados do servidor (microcontrolador com sistema ARCOS) (ADEPT, 2011).

O microcontrolador é responsável por gerenciar os controles de baixo nível do robô, tais como:

- ✓ Controle dos diversos sensores e atuadores;
- ✓ Gerenciamento da velocidade dos motores;
- ✓ Ativação dos sonares;
- ✓ Leitura de dados de sensores;
- ✓ *Encoder* das rodas.

Figura 14 – Robô móvel Pioneer 3 da empresa MobileRobots
Fonte:< http://www.mobilerobots.com/researchrobots/researchrobots/p3at.aspx>

4.4 Xbot *RoboDeck*

A Xbot é uma empresa brasileira localizada na cidade de São Carlos, estado de São Paulo, que fabrica e comercializa robôs móveis para as áreas da educação e do entretenimento. Ela surgiu de um projeto de pesquisa e desenvolvimento na área de robótica, tendo apoio financeiro do programa de fomento para pequenas empresas da FAPESP.

Seu objetivo é disponibilizar ferramentas para melhorar o ensino e o aprendizado dos estudantes, de praticamente todas as faixas etárias, desde o Ensino Fundamental até a Pós-Graduação.

O *RoboDeck* é um robô móvel, desenvolvido pela empresa C. Associados Desenvolvimento Tecnológico Ltda e comercializado pela empresa Xbot.

Com finalidade educacional, apresenta-se como uma plataforma robótica, com ambiente de programação baseado na linguagem C/C++ e possui um kit de desenvolvimento de software (SDK) para teste, fornecendo toda liberdade para os usuários desenvolverem aplicações, utilizando as bibliotecas existentes para comandar o robô. Trata-se de uma plataforma robótica móvel universal de código aberto.

O *RoboDeck* contém uma *Giga* de testes, para verificar se os comandos estão sendo executados com sucesso. Além disso, disponibiliza vários recursos, como:

- ✓ Câmera para fotografar e transmitir imagens em tempo real;
- ✓ *GPS* para informar dados como: altitude, latitude, longitude, velocidade, sentido, data e hora;
- ✓ *Wi-fi* que possibilita controlar o robô a uma distância de até 100 metros;
- ✓ *ZigBee* que permite a comunicação com o robô a uma distância de até 1000 metros, oferecendo boa resistência aos ruídos, por se tratar de uma rede mais robusta.

A *Giga* também é capaz de:

- ✓ Informar leituras de distância dos sensores ultrassônicos e infravermelhos;
- ✓ Medir temperatura e umidade;
- ✓ Informar o norte através de uma bússola.

Um sensor ultrassônico possibilita que o robô localize obstáculos de até 6 metros de distância, dentro de um raio de alcance de aproximadamente 45º, auxiliando o robô a não se colidir com possíveis objetos.

Os sensores infravermelhos localizam-se nas laterais do robô e conseguem detectar a presença de objetos, em até 70 cm de distância (XBOT, 2011).

Figura 15 – Robô móvel *RoboDeck* da empresa Xbot
Fonte:< http://www.xbot.com.br/educacional/*robodeck*/>

4.4.1 Estrutura interna do *RoboDeck*

O *RoboDeck* permite adicionar periféricos robóticos como garras, braços, sensores entre outros. Ele é formado por dois hardwares: essencial e opcional.

O hardware essencial é composto por dois microcontroladores: o primeiro responsável pelo gerenciamento dos sensores e atuadores possui um software chamado de MCR (Módulo de Controle Robótico), que faz o robô trabalhar como escravo, executando sequencialmente os comandos. O segundo é responsável pelo gerenciamento da comunicação de controle do robô e também por manter a sua identidade.

A implementação da comunicação de controle é realizada pelo MCC (Módulo de Controle de Comunicação), através do protocolo Zigbee. A implementação da identidade ocorre pelo MCS (Módulo de Controle de Sessão), que implementa o conceito de sessão aos comandos provenientes dos controladores do robô, impedindo-lhe responder a dois controladores simultaneamente, além de ser responsável pela identificação e autenticação dos controladores.

Os softwares que compõem esse hardware são escritos em linguagem C e não são baseados em Sistema Operacional (MUÑOZ, 2011).

Com o hardware opcional é possível carregar aplicativos robóticos feitos pelos usuários, por meio de um software chamado MAP (Módulo de Alta Performance),

permitindo controlar o robô e fornecer-lhe autonomia. Os aplicativos mencionados possibilitam controlar o hardware essencial do robô, através de comandos enviados diretamente ao MCS, podendo ser utilizados para expandir os comandos robóticos, ao implementar comandos acessíveis pelos controladores do robô.

O MAP admite comunicação com os controladores através de adaptadores USB para Bluetooth ou Wi-Fi e funciona em qualquer placa microprocessada, desde que seja compatível com o sistema operacional Linux e possua interfaces serial e USB: essa característica viabiliza ao MAP utilizar novas tecnologias de circuitos integrados, permitindo que o robô sempre esteja atualizado (MUÑOZ, 2011).

A Figura 16 oferece uma visão geral da arquitetura de software do robô, mostrando sua relação com os hardwares essencial e opcional. As setas indicam a dependência entre os módulos:

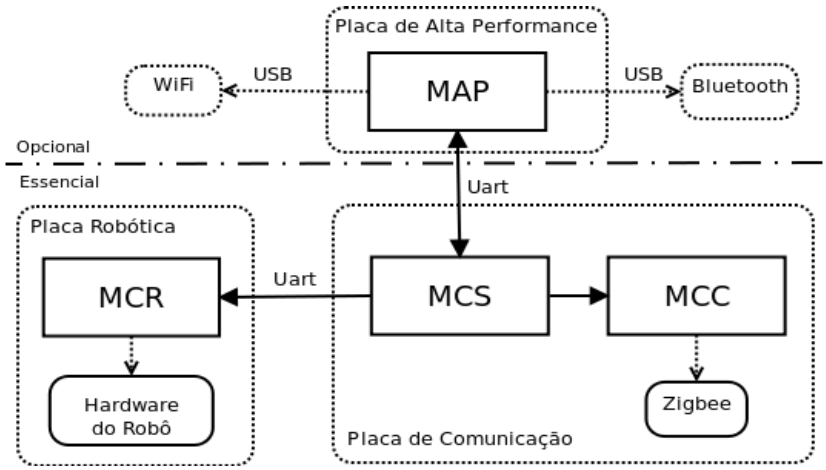


Figura 16 – Principais divisões de hardware e software do *RoboDeck* (MUÑOZ, 2011)

4.5 Comparações entre as plataformas de robótica móvel

A tabela 2 mostra quatro plataformas de robótica móvel que são utilizadas para pesquisa e desenvolvimento e faz uma comparação entre suas principais características. Os dados mostrados nesta tabela foram estabelecidos de acordo com a documentação fornecida pelos fabricantes.

Tabela 2 – Comparação entre as Plataformas de Robótica Móvel Analisadas

Plataformas de Robótica Móvel	iRobot Create Plus Command Module	LEGO MINDSTORMS NXT Base Set	MobileRobots Inc Pioneer 3	Xbot *RoboDeck*
Velocidade do Processador	18 MHz	48 MHz	50 MHz	250MHZ e um de 500MHZ opcional da placa de alta performance
Interface RS232	1 porta	Nenhuma	3 portas	1 porta
Entradas e Saídas	25 DIO, AI e AO	4 portas de entrada e 3 portas de saída	Barramento externo de 8 bits para E/S até 16 dispositivos + PC104	Placa de alta performance opcional (1 Entrada COM_1 3- Entradas USB 1 - Entrada de Áudio. 1 - Entrada de Microfone. 1- Entrada de Rede. 1- Entrada de Vídeo VGA.)
Armazenamento não volátil	144 KB	256 KB	1 Mb	Opcional da placa de alta performance (HD Flash Disk 8GB)
Ambientes Compatíveis	Ambientes internos, piso liso, 2.27 kg de carga	Ambientes internos, piso liso,3 kg de carga	Ambientes Internos, obstáculos de até 2.5 cm de altura, rampas compatíveis com cadeira de rodas, com cargas de 14 a 23kg	Ambientes internos, chão de habilitação: 70,2mm, 10kg de carga
Sensores integrados	Sim	Sim	Sim	Sim

Observa-se que tais plataformas de robótica móvel possuem características gerais muito próximas. No entanto, o *RoboDeck* se destaca, em virtude do seu processador interno com maior velocidade e da opção de incluir uma placa de alta performance.

Outro ponto de destaque do *RoboDeck* é que utilizando a placa de alta performance, o mesmo terá um número maior de interfaces de entrada e saída, além de um incremento na capacidade de armazenamento não volátil.

Com base nessa análise, o *RoboDeck,* usando a placa de alta performance, torna-se a melhor alternativa entre as plataformas de robótica móvel comparadas, para receber a interface de reconhecimento de fala.

5. DESENVOLVIMENTO DOS MÓDULOS DE CONTROLE DE VOZ DO *ROBODECK*

Este capítulo apresenta como foram desenvolvidos os dois módulos de controle de voz para o *Robodeck*, o interno e o externo, destacando suas vantagens e desvantagens.

No módulo interno integrou-se uma biblioteca de reconhecimento de fala ao MAP (Módulo de Alta Performance) do *RoboDeck*, dispensando qualquer dispositivo externo para controlar o robô por meio da fala.

No módulo externo foi integrada uma biblioteca de reconhecimento de fala, ao SDK fornecido pelo fabricante do robô, que utiliza um computador para acessar e controlar o *RoboDeck*: essa integração possibilitou controlar o robô pela fala, através de um dispositivo móvel como um celular.

5.1 Módulo de Controle Interno

Para o desenvolvimento do módulo de controle interno de voz, foi escolhida a biblioteca de reconhecimento de fala *CVoiceControl* por ser compatível com o Sistema Operacional Linux, em especial a versão *Debian,* fato incomum, uma vez que a maioria dos sistemas de reconhecimento de fala são compatíveis, apenas, com os Sistemas Operacionais da Microsoft.

Outro fator importante na escolha desse software é a inexistência de um dicionário com comandos preestabelecidos, diferente de outros sistemas de reconhecimento em que ocorre, na maioria dos casos, a limitação de só reconhecer comandos em Inglês: no *CVoiceControl* o usuário grava uma sentença e faz sua associação a um comando em *Linux*, independente da língua empregada.

Além disso, o *CVoiceControl* foi desenvolvido em linguagem C, a mesma usada no SDK (*Software Development Kit)* do *RoboDeck*, para criação e inclusão de aplicações dentro do próprio robô.

Outra justificativa é não ter a necessidade de instalar ou configurar o *CVoiceControl*, em um ambiente com interface gráfica: ideal para a implementação no *RoboDeck*, que trabalha com o Sistema Operacional *Debian* e não possui essa interface, mas, somente linhas de comando CLI (Interpretador de Linha de Comandos).

A única limitação do *CVoiceControl* é a ausência de uma boa documentação, pois o projeto foi descontinuado pelo seu criador Daniel Kiecza, assim como as pesquisas e o desenvolvimento do mesmo, (o programa não possui nenhuma atualização desde 11 de novembro de 2002). Por outro lado, importa salientar que o código-fonte está disponível, possibilitando novas atualizações e evolução do pacote.

Diante do exposto, instalou-se e configurou-se o *CVoiceControl* no sistema operacional Linux, versão *Debian,* do *RoboDeck*, como uma biblioteca de reconhecimento de fala. Realizando um estudo mais aprofundado da arquitetura interna do *RoboDeck*, iniciou-se o estudo pelo Módulo de Controle Robótico, passando pelo Módulo de Controle de Comunicação e terminou no MAP (Módulo de Alta Performance) do robô.

Constatou-se a necessidade de criar controladores responsáveis pelas movimentações do robô e que cada um deles fosse acionado a uma referência de fala, por meio do *CvoiceControl*. A Figura 17 demonstra a interação entre os comandos de fala reconhecidos, os controladores desenvolvidos e o MAP (Módulo de Alta Performance) do robô.

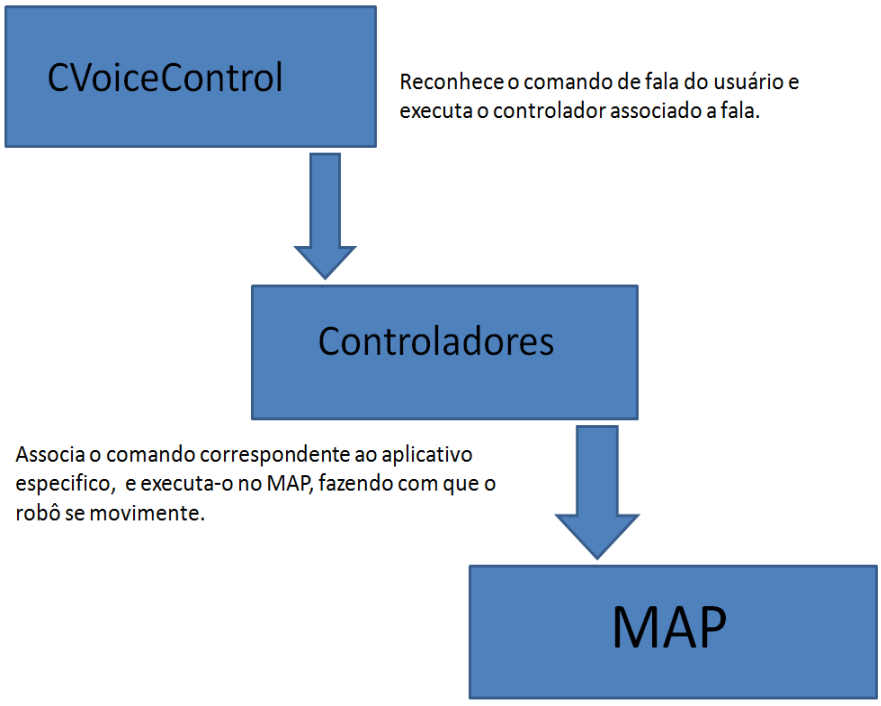


Figura 17 - Integração entre o CVoiceControl, os Aplicativos e o MAP

Os quatro controladores desenvolvidos, cada um associado a uma referência de fala no *CvoiceControl,* são:

- ✓ Controlador **"Ande":** responsável pela movimentação do robô para frente, associou-se à referência de fala **"Mova".**
- ✓ Controlador **"Parar":** responsável por parar a movimentação do robô, associou-se à referência de fala **"Pare".**
- ✓ Controlador **"Esquerda":** responsável por virar o robô à esquerda, associou-se à referência de fala **"Esquerda".**
- ✓ Controlador **"Direita":** responsável por virar o robô à direita, associou-se à referência de fala **"Direita".**

Esses controladores foram criados mediante utilização de uma biblioteca ou API (Interface de Programação de Aplicativos), desenvolvida por Orlandini (2012), com a função de acionar os controles físicos do robô, como os motores. Tal acionamento é feito via MAP, através de uma conexão com ele, iniciando uma sessão. Após, o robô executa o controlador responsável pela movimentação desejada.

A Figura 18 mostra o código-fonte do controlador **"Ande",** cuja função é mover o robô para frente.

```
#include <iostream>
#include <stdio.h>
#include "RoboDeckMoving.hh"
#include "RoboDeckInfo.hh"
using namespace std;

RoboDeckMoving roboDeckMoving;
RoboDeckInfo roboDeckInfo;
RoboDeck robo;

int main(int argc, char *argv[])
{
  roboDeckInfo.openConnection("localhost","2000", robo);
  roboDeckInfo.openSession(robo);

      bool moveuRobo = roboDeckMoving.moveRobot(32000,0,robo);

      cout <<"Robot Comunication Protocol Version: "
<<roboDeckInfo.getRobotProtocolVersion(robo) << endl;
           roboDeckInfo.closeSession(robo);
           return 0;
}
```

Figura 18 – Código-fonte do controlador "Ande"

Observa-se que o código-fonte ficou reduzido, pois inclui duas Bibliotecas desenvolvidas por Orlandini (2012): a primeira, "***RoboDeckInfo.hh***", tem a função de

estabelecer a conexão e iniciar uma sessão com o MAP (Modulo de Alta Performance) do *RoboDeck*, para envio de comandos. A segunda, ***"RoboDeckMoving.hh",*** tem a finalidade de enviar os comandos responsáveis pela movimentação do robô, ao MAP.

Verificam-se também as utilizações dos comandos "***roboDeckInfo.openConnection",*** para estabelecer a conexão com MAP, ***"roboDeckInfo.openSession",*** para iniciar a sessão com o mesmo, e ***"roboDeckMoving.moveRobot",*** para mover o robô para frente.

5.1.1 Funcionamento do *CVoiceControl*

O *CVoiceControl* utiliza três módulos:

- ✓ *microphone_config;*
- ✓ *model_editor;*
- ✓ *cvoicecontrol.*

Ao iniciar o *CVoiceControl* pela primeira vez, o usuário deve calibrar o microfone para usá-lo no reconhecimento de voz, através do módulo *"microphone_config".* Depois que a placa de som estiver preparada, o usuário precisa criar os modelos com as sentenças faladas e associá-las aos comandos que deseja realizar no Linux, através do módulo *"model_editor":* esses são os principais objetos necessários para o processo de reconhecimento de voz (KIECZA, 2002).

5.1.1.1 Módulo de calibragem do microfone

O *Microphone_config,* primeiro módulo a ser configurado, é responsável por calibrar o microfone que receberá o comando de voz do usuário. Seu processo de calibragem é realizado em cinco etapas:

- ✓ *Select Mixer Device;*
- ✓ *Select Audio Device;*
- ✓ *Adjust Mixer Levels;*
- ✓ *Calculate Recording Thresholds;*
- ✓ *Estimate Characteristics of Recording Channel.*

As duas primeiras etapas, *Select Mixer Device* e *Select Audio Device,* servem para selecionar a placa de som ou o dispositivo de áudio que o software usará, no

caso dos equipamentos com duas ou mais placas. Caso o equipamento possua apenas uma placa de som, é provável que o sistema identifique-a automaticamente, concluindo-se essas duas etapas (KIECZA, 2002).

A terceira etapa, *Adjust Mixer Levels,* ajusta o nível de intensidade do microfone, ou seja, determina automaticamente o seu volume no ponto máximo. Ao iniciar esse processo, o usuário deve falar e dar risadas em voz alta diante do microfone: cada vez que o processo é realizado, reduz-se o nível do microfone, sendo necessário repeti-lo várias vezes, até que seu nível de entrada fique na faixa de 60 a 95. (KIECZA, 2002).

A quarta etapa, *Calculate Recording Thresholds,* encontra o nível do sinal sonoro mínimo para iniciar a captura de áudio. No primeiro momento, é necessário que o usuário permaneça em silêncio para serem capturados todos os sons externos proporcionados pelo ambiente. Já no segundo momento, o usuário deve sustentar uma conversa até que o ciclo seja concluído com sucesso. Assim, calculam-se os níveis de intensidade ideal para que a gravação possa ser iniciada e parada (KIECZA, 2002).

A quinta etapa, *Estimate Characteristics of Recording Channel,* visa a novamente estabelecer o nível mínimo do volume do microfone, através da análise dos ruídos do ambiente. Para efetuar essa verificação, o usuário deve ficar em silêncio para capturar todos os sons externos, como o ruído de fundo, entre outros (KIECZA 2002).

Realizadas essas etapas, o item *Write Configuration* ficará disponível para que o usuário possa salvar a calibragem do microfone. Ao salvá-lo, gera-se um arquivo chamado *Config,* que possui as configurações da calibragem, o qual será gravado no diretório do *CVoiceControl*.

5.1.1.2 Módulo de edição de modelo

O *Model_Editor*, segundo módulo a ser configurado, realiza o aprendizado de comandos e o controle através do reconhecimento de fala, podendo o usuário associar referências sonoras a um determinado comando Linux: cada comando deve ter no mínimo quatro referências de voz (KIECZA, 2002).

Este módulo é composto por um número variado de elementos de referência, cada item corresponde a um comando que pode ser reconhecido. Um item de

referência consiste num rótulo, uma transcrição do que é dito em um comando Linux, executado no momento do reconhecimento desse item.

Dessa forma, para reconhecer um comando de voz, esse componente compara a fala do usuário com todos os itens de referência de voz dos modelos ativos: se uma das referências de voz for semelhante à expressão de entrada, escolhe-se tal item como resultado do reconhecimento e executa-se o comando associado a ele.

O *Model_editor* é formando por quatro funções:

- ✓ *New Speaker Model;*
- ✓ *Load Speaker Model;*
- ✓ *Edit Speaker Model;*
- ✓ *Save Speaker Model.*

Através dessas funções o usuário pode redefinir o modelo atual *New Speaker Model*, carregar um modelo existente *Load Speaker Model,* editar o modelo atual *Edit Speaker Model* e salvá-lo com o *Save Modelo Speaker* (KIECZA, 2002).

O editor de modelo *Edit Speaker Model* mostra os itens de referência atual em forma de tabela, exibindo-os em linhas. Nele, o usuário pode adicionar um item com uma nova referência, editar um item destacado ou excluí-lo: no momento em que se adicionar um novo modelo de comando, o sistema exibirá uma tela onde o usuário terá de informar seu título e editá-lo para ser executado. Após definição do comando, o usuário deverá gravar uma referência de voz para ele. Visando a garantir uma qualidade no reconhecimento da fala, torna-se necessário repetir essa operação no mínimo quatro vezes. Em seguida, o sistema apresentará uma lista com as referências de voz, informando a data e a hora em que foi capturada a fala pelo microfone (KIECZA, 2002).

A Figura 19 mostra um exemplo, cujo título do comando é **mova,** com a função de executar o controlador "**ande"** que faz o *RoboDeck* Mover para frente**.**

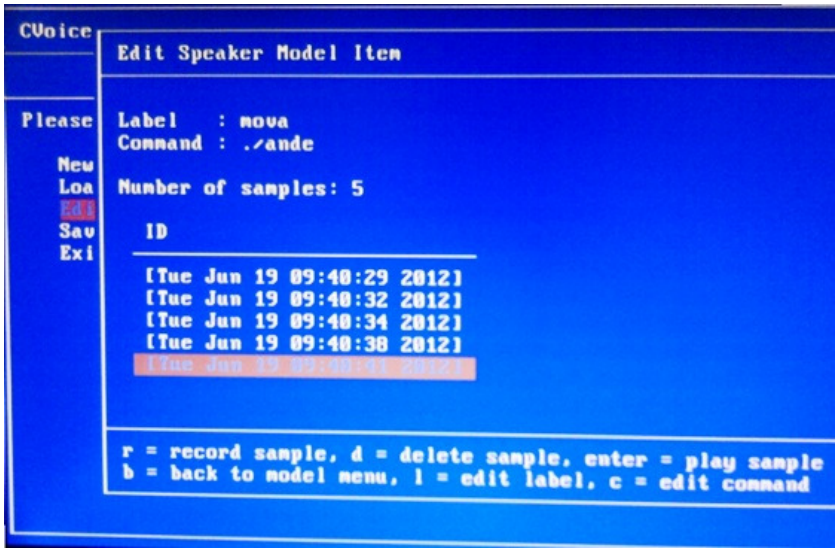


Figura 19 – Tela de configuração do comando mova do *CVoiceControl*

Ao iniciar a gravação das referências de voz, é necessário o usuário esperar um segundo ou mais antes de começar a falar, porque o *buffer* de áudio precisa ser preenchido antes que a fala do usuário seja capturada e gravada.

Uma grande vantagem do *CVoiceControl* é permitir que o usuário crie vários modelos, expondo uma lista com os nomes das referências de voz e os comandos associados a eles.

A Figura 20 mostra uma lista de quatro modelos de referências de voz associados aos controladores responsáveis pela movimentação do robô: ande, pare, esquerda e direita.

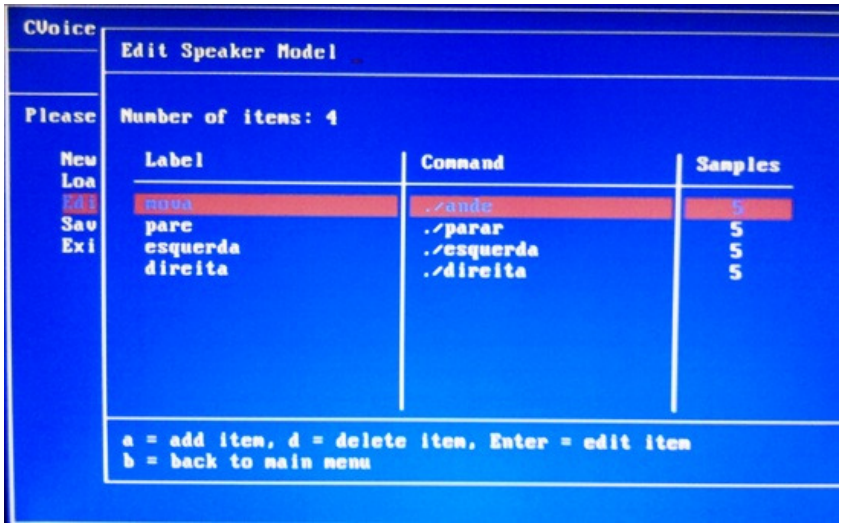


Figura 20 – Tela do *CVoiceControl* exibido a lista de modelos criados

Depois disso, o usuário deverá salvá-los, através da opção *"Save Speaker Model",* atribuindo um nome ao arquivo com extensão ***.cvc*** adotada pelo sistema (KIECZA, 2002).

É importante destacar que o reconhecimento da fala é executado em primeiro plano, ou seja, se algum usuário quiser utilizar a saída da placa de som, o sistema de reconhecimento de fala será interrompido até que ela seja liberada: comportamento necessário porque muitas placas de som não permitem gravar e reproduzir ao mesmo tempo (KIECZA, 2002).

Outro fator relevante é que se um modelo de referência de voz for reconhecido, o comando associado a ele será executado, pois o *CVoiceControl* não verifica se a execução daquele comando irá prejudicar o Sistema Operacional, tornando-se responsabilidade de cada usuário definir comandos que não gerem conflitos ou prejudiquem o seu Sistema Operacional.

5.1.1.3 Módulo de ativação do *CVoiceControl*

Após conclusão dos dois módulos, o usuário pode iniciar o processo de reconhecimento de fala do *CVoiceControl*, fazendo o Linux obedecer aos comandos de voz: para tanto é necessário digitar o comando *cvoicecontrol* "nome-arquivo-do-modelo", isto é, o nome do modelo que se deseja usar, no console do Linux, como no exemplo abaixo:

$ *cvoicecontrol* teste.cvc

O reconhecedor de fala entra em modo de gravação automática e o usuário deve certificar-se de que nenhum aplicativo precisa de acesso ao dispositivo de som, pois neste momento, como na maioria dos dispositivos de som, só se permite acesso exclusivo (KIECZA 2002).

Para encerrar o programa, o usuário deve pressionar a tecla *Ctrl-C* no console e digitar o comando *killall cvoicecontrol.*

5.2 Módulo de Controle Externo

Para o desenvolvimento do módulo de controle externo de voz, foi escolhida a biblioteca de reconhecimento de fala *Voice Search* da empresa Google, pelo fato de

ser compatível com qualquer celular que tenha o Sistema Operacional *Android* e com qualquer microcomputador que possua o navegador *Chrome* instalado.

A escolha dessa biblioteca também aconteceu porque ela reconhece palavras em Português, o que não ocorre na maioria dos sistemas de reconhecimento, limitada à compreensão de comandos em Inglês.

A grande vantagem do *Voice Search* em relação ao *CVoiceControl*, utilizado no módulo de controle de voz interno, é dispensar o pré-treinamento dos locutores. Além disso, o *Voice Search* não precisa ser instalado no sistema operacional *Android*, porque todos os celulares com esse sistema operacional são aptos ao seu uso.

Recentemente a Google lançou uma versão do *Voice Search* compatível com *iPhone, iPod touch e iPad Requires iOS 4.2* ou posterior.

A única limitação do *Voice Search* é a necessidade de uma conexão com a Internet para funcionar, pois ele só reconhece a fala por meio de um serviço fornecido pela *Google*: fato pouco relevante nos dias de hoje, diante da quase totalidade dos proprietários desse tipo de celular possuir acesso à internet.

Ao realizar um estudo mais aprofundado do *Voice Search*, notou-se que ele é acionado através de qualquer página da web, que possua um ***TextBox*** ou caixa de texto.

Dessa forma, o primeiro passo foi gerar uma página de web em ***html,*** com um ***TextBox,*** para acionar o *Voice Search*. Para automatizar a execução dos comandos de fala reconhecidos, criou-se nessa página um *JavaScript*, objetivando verificar a cada 1000ms, novas entradas no *TextBox*.

A Figura 21 mostra o código-fonte da página de web Reconhecimento de Fala com o *JavaScript,* utilizado para automatizar o envio dos comandos de fala reconhecidos pelo *Voice* Search (vide apêndice para maiores detalhes).

```
<HTML>
<HEAD>
<TITLE>Reconhecimento de Falta</TITLE>
    <SCRIPT>        //inicio do script

        //função javascript para verificação de texto no campo
        function verificaCampos(form)
        {
            //variavel que recebe a referencia do campo no formulario
            var campo = form.campo;
            //verifica se o campo esta vazio
            if (campo.value == "")
            {
                //envia o foco (cursor) de volta ao campo
                campo.focus();
            }
            else
            {
                //submete o formulario para a pagina "processa.aspx
                form.submit();
            }
            //funçao recursiva que chama novamente a funcao verificaCampos a cada 1000ms
            setTimeout("verificaCampos(form)",1000);
        }
    </SCRIPT>
</HEAD>
<BODY onLoad="verificaCampos(form);">
    <FORM id="form" action="processa.aspx">
        <h1>Reconhecimento de Fala</h1>
        Fale o comando desejado<BR><INPUT name=campo>  
    </FORM>
</BODY>
</HTML>
```

Figura 21 – Código-fonte da Página de *web* Reconhecimento de Fala com o *JavaScript*

A Figura 22 apresenta a página de web com o ***TextBox,*** recebendo o comando de fala via *Voice Search*.

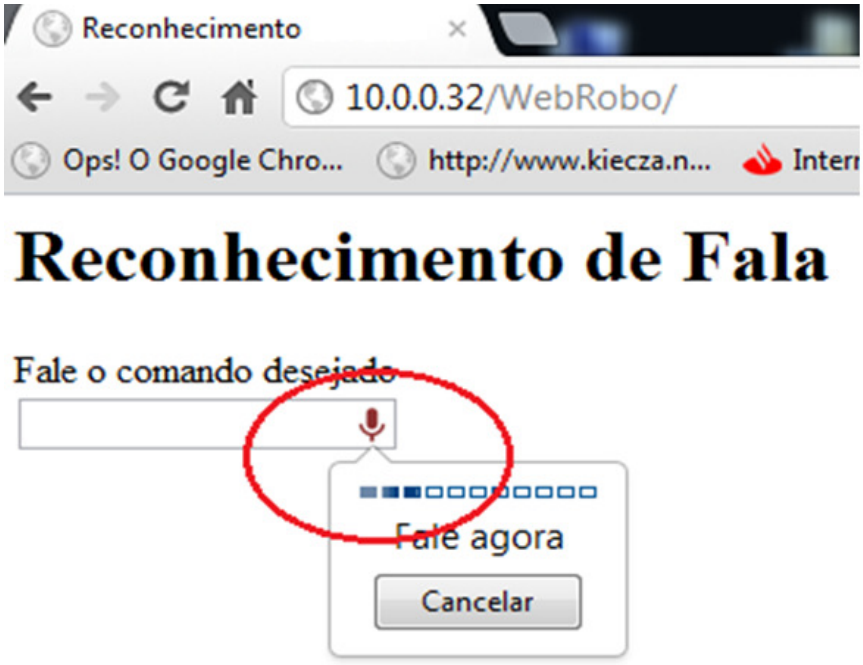


Figura 22 – Página de *web* Reconhecimento de Fala

No próximo passo produziu-se um banco de dados em *MySQL* chamado "**falaBD",** que possui uma tabela denominada **"comandos",** com os campos **Codcomando, Comando, DataHora e Status,** cuja função é receber, armazenar os comandos da página de web e enviá-los para o SDK.

Na sequência, foi estabelecida uma conexão entre a página e o banco de dados, empregada para enviar e armazenar, no banco de dados **"falaBD",** todo comando de fala reconhecido pelo *Voice Search*, dentro do campo **"Comando"**; o campo **"DataHora"** registrou a data e a hora em que o comando foi gravado no banco de dados, e o campo ***"Status"*** manteve uma mensagem, dizendo se o comando foi armazenado com sucesso.

A Figura 23 mostra a descrição da tabela "comandos" do banco de dados **"falaBD":**

Name	Type	Null	Default	Extra	Comment
CodComando	int(11)	No		auto_incre...	
Comando	varchar(255)	Yes			
DataHora	datetime	Yes			
Status	varchar(20)	Yes			

Figura 23 – Descrição da tabela "comandos" do banco de dados "FalaBD"

Passou-se à utilização do SDK (Kit de desenvolvimento de Software), fornecido pela empresa Xbot, consistente num sistema chamado “Giga de testes”, que usa a linguagem C#. O SDK possibilita conectar-se ao *RoboDeck*, através de um computador, e iniciar uma sessão no MAP (Módulo de Alta Performance), para enviar comandos destinados a movimentar o robô ou ler seus sensores.

Dentro do SDK criou-se uma conexão com o banco de dados “falaBD”, cujo objetivo era receber do banco de dados os comandos de fala reconhecidos no *Voice Search,* através da página de web (vide apêndice para maiores detalhes).

A Figura 24 demonstra a interação entre o dispositivo móvel, a página de web, o *Voice Search*, o banco de dados, o SDK “Giga de Testes” e o *RoboDeck*.

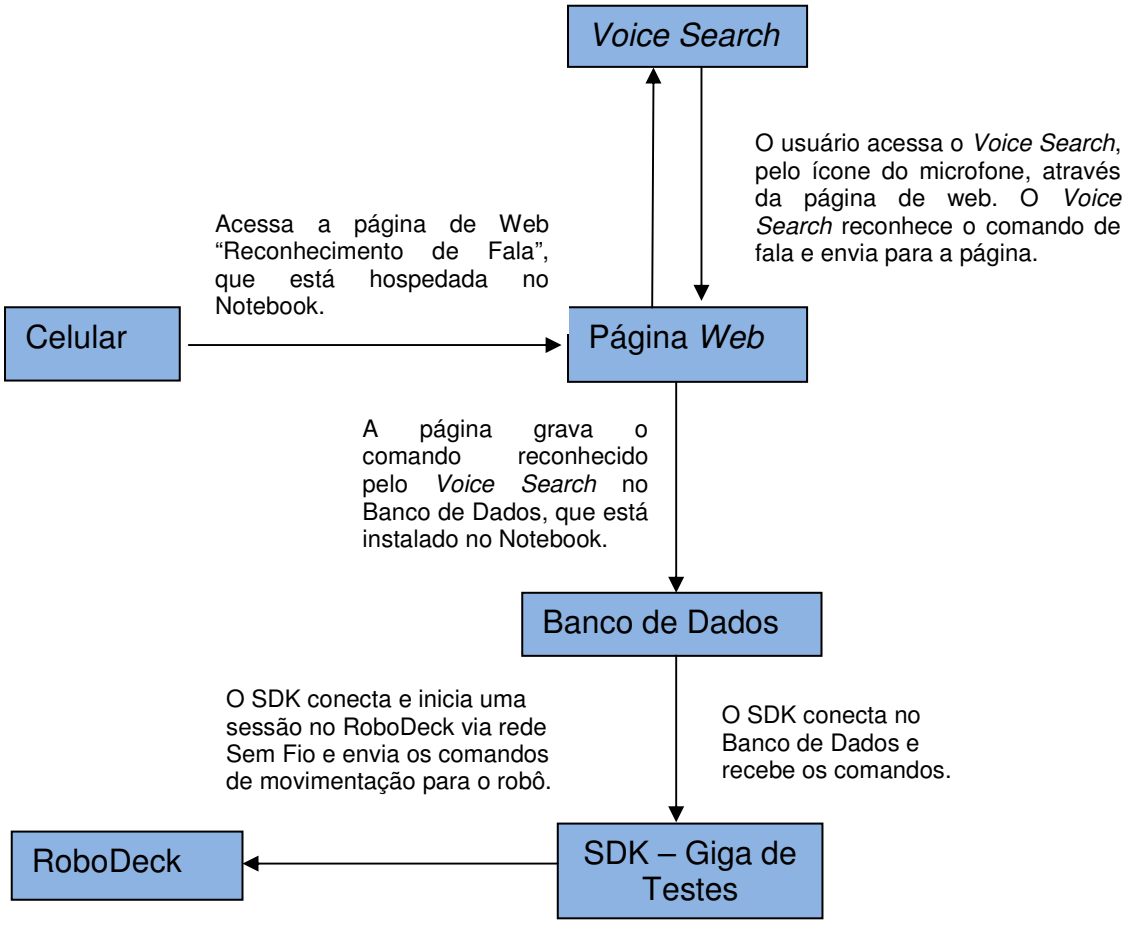


Figura 24 - Arquitetura Geral do módulo de voz por controle externo ao *RoboDeck*

Ainda dentro do SDK, gerou-se um recurso com a função de habilitar ou desabilitar a verificação de nova entrada no campo “comando” do banco de dados “falaDB”, impedindo, se necessário, a entrada de comandos não autorizados. Esse

recurso também disponibiliza uma visualização dos últimos comandos verificados e executados no SDK, admitindo, se necessário, pesquisar todos os comandos armazenados no banco de dados, informando a data, a hora e o status (vide apêndice para maiores detalhes).

As Figuras 25 e 26 ilustram as telas dos recursos citados:

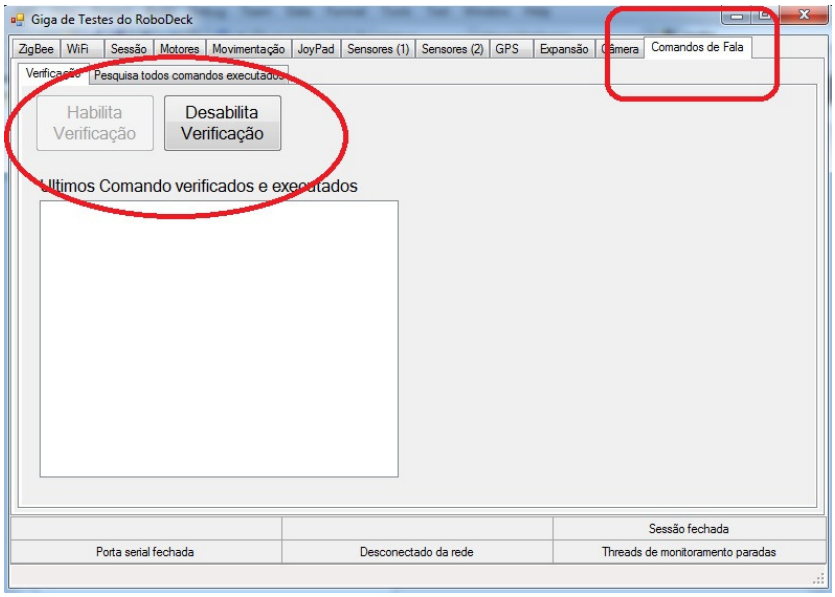


Figura 25 – Tela para habilitar ou desabilitar verificação de novas entradas de comandos

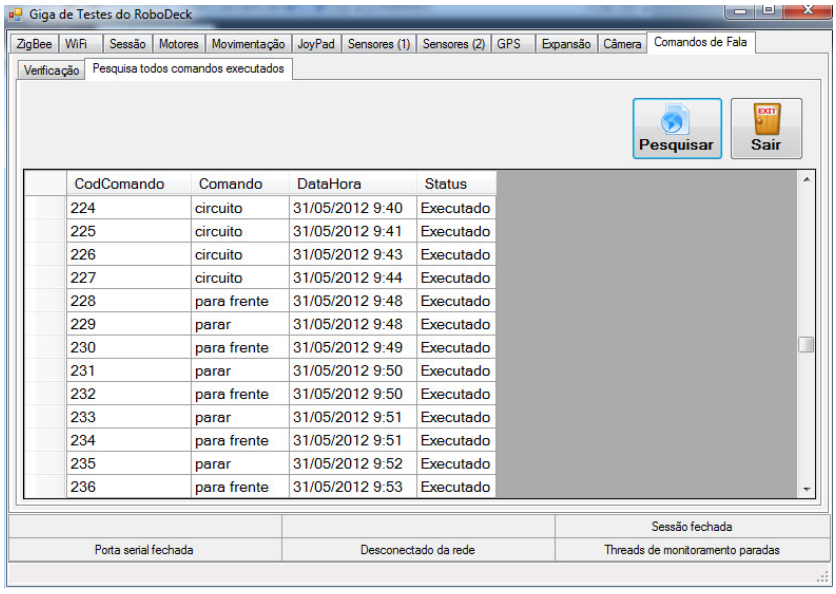


Figura 26 – Tela para pesquisar todos os comandos executados no Banco de Dados "falaBD"

No último passo, elaborou-se dentro do SDK um método para cada comando recebido pelo banco de dados (vide apêndice para maiores detalhes).

A Figura 27 mostra o código-fonte do método **“moverf1”,** chamado quando o SDK recebe os comandos (mova ou mover, para frente ou para cima) do banco de dados “falaBD”:

Figura 27 – Código-fonte do método “moverf1”

Mediante análise dessa Figura, nota-se que o método “moverf1” utilizou o comando robótico do SDK *“robodeck.move”* e a intensidade foi configurada em 32766, fazendo o robô mover-se para frente com potência máxima.

6. EXPERIMENTOS REALIZADOS

Esse capítulo relata os experimentos realizados com os módulos de voz desenvolvidos para a plataforma *RoboDeck*, a fim de testar e analisar os resultados obtidos em cada experimento. Foram realizados experimentos com os módulos de voz interno e externo, aplicando os métodos e controladores implementados.

6.1 Experimento 1: Movimentações básicas via módulo de controle de voz externo.

6.1.1 Objetivo

Demonstrar as movimentações básicas do *RoboDeck* via reconhecimento de fala, por meio de um aparelho móvel. Nesse experimento foram realizadas as seguintes movimentações todas com velocidades pré-definidas:

- ✓ Para frente;
- ✓ Para trás;
- ✓ Virar à esquerda;
- ✓ Virar à direita;
- ✓ Parar;
- ✓ Girar no próprio eixo para à esquerda;
- ✓ Girar no próprio eixo para à direita;

6.1.2 Ambiente físico

Oficina da empresa Xbot, onde o piso era de cerâmica com algumas irregularidades. O ruído sonoro era elevado, pois o ambiente tinha acesso à rua e havia vários funcionários trabalhando.

Figura 28 – Foto do *RoboDeck* na oficina da empresa *Xbot*

6.1.3 Recursos tecnológicos utilizados

Quatro recursos foram usados: um notebook, um celular, um roteador Sem Fio e um *RoboDeck*.

O notebook teve as seguintes funções:

- ✓ **Servidor da página de web Reconhecimento de Fala**: enviar a página para qualquer dispositivo móvel, via rede sem fio, empregando o Software ***"UtiDev Web Server".***
- ✓ **Servidor do banco de dados "falaBD": ** receber os comandos da página de web, utilizando o Sistema gerenciador de Banco de Dados ***"MySQL Server 5.1".***
- ✓ **Execução do novo SDK Giga de Testes com o módulo de controle por voz**: receber o comando do banco de dados **"falaBD",** conectando-se via rede sem fio com o *RoboDeck,* iniciando uma sessão e enviando os comandos de movimentação.

As funções do celular consistiram em conectar-se no servidor de web via rede sem fio, iniciando a página de web de Reconhecimento de Fala e acionando o *Voice Search*, já instalado no mesmo, para o reconhecimento dos comandos falados.

A Figura 29 mostra o celular conectado na página de web, acionando o reconhecimento de fala via *Voice Search.*

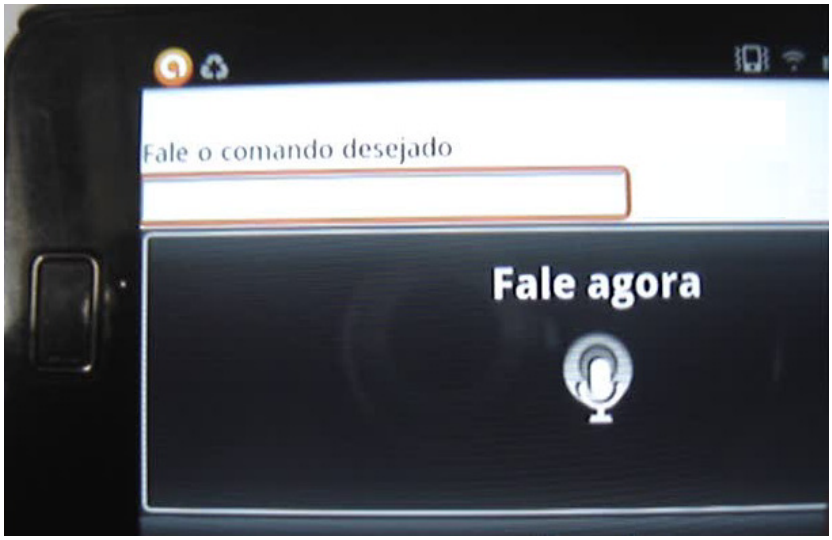


Figura 29 – Foto do Celular conectado na página de web, acionando o *Voice Search*

O Roteador Sem Fio teve a função de receber a internet na porta WAN e distribuir via rede Sem Fio, além de ter sido utilizado como meio de interligação entre o Notebook, o Celular e o *RoboDeck*.

Por fim o *RoboDeck*, recebia os comandos via rede sem fio do SDK, que estava sendo executado no Notebook e iniciava sua movimentação.

6.1.4 Comandos robóticos utilizados

Utilizaram-se quatro comandos robóticos de movimentação do SKD Giga de Testes:

- ✓ ***"robodeck.move":*** para criar dois métodos, mover o *RoboDeck* para frente, com a intensidade configurada em um valor positivo de 32760; mover o robô para trás, onde a intensidade foi configurada num valor negativo de -32760.
- ✓ ***"robodeck.brake":*** para criar o método com a função de parar a movimentação do *RoboDeck*, posicionando suas rodas em linha reta com um ângulo de 0 grau.

- ✓ ***"robodeck.turn":*** para criar dois métodos, virar o *RoboDeck* à esquerda, onde a intensidade foi configurada em um valor positivo de 15000 e o ângulo em um valor negativo de -32; virar o robô à direita, com intensidade e ângulo configurados em um valor positivo de 15000 e 32, respectivamente.
- ✓ ***"robodeck.roll":*** para criar dois métodos, girar o *RoboDeck* no seu próprio eixo à esquerda, onde a intensidade foi configurada em um valor positivo de 15000 e o ***"rdsdk.Side"*** foi ajustado em ***"LEFT"***; girar o robô no seu próprio eixo à direita, com a intensidade também configurada num valor positivo de 15000, mas ***"rdsdk.Side"*** ajustado em ***"RIGHT"***.

6.1.5 Implementação

Foram desenvolvidos sete métodos, que eram chamados através dos comandos recebidos pelo banco de dados **"falaBD"** (vide apêndice para maiores detalhes).

O primeiro foi o método **"moverf1(),** com a função de mover o *RoboDeck* para frente. A Figura 30 mostra o código-fonte de sua implementação:

```
// mover para frente

public void moverf1()
{
    short intensity = 32760;
    report(robodeck.move(
    intensity,
    new GigaCallback(this, actionCallback)));

}
```

Figura 30 – Código-fonte do método "moverf1()"

Observando a Figura, verifica-se a intensidade configurada em 32760, movendo o robô para frente com potência máxima.

É executado se a linha "comando" do banco de dados "falaBD" for igual à "mova", ou "mover", ou "para frente", ou "para cima". Então, o método "moverf1()" possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada, conforme a Figura 31:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

            if ((linha.Comando.Equals("mova")) || (linha.Comando.Equals("mover")) ||
                (linha.Comando.Equals("para frente")) || (linha.Comando.Equals("para cima")))
            {
                moverf1();
            }
```

Figura 31 – Código-fonte das condições para executar o método "moverf1()"

O segundo método foi o método **"movert1()",** cuja função é mover o *RoboDeck* para trás. A Figura 32 mostra o código fonte de sua implementação:

```
public void movert1()
{
    short intensity = -32760;
    report(robodeck.move(
    intensity,
    new GigaCallback(this, actionCallback)));

}
```

Figura 32 – Código-fonte do método "movert1()"

Observando a Figura, nota-se que a intensidade foi configurada em -32760, movendo o robô mover para trás com potência máxima.

É executado se a linha "comando" do banco de dados "falaBD" for igual à "para trás", ou "para traz", ou "traz", ou "mover para traz", ou "re", ou "ré", ou ainda "atrás", de acordo com a Figura 33:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

         if ((linha.Comando.Equals("para trás")) || (linha.Comando.Equals("para traz")) ||
             (linha.Comando.Equals("traz")) || (linha.Comando.Equals("mover para traz")) ||
             (linha.Comando.Equals("re")) || (linha.Comando.Equals("ré")) ||
             (linha.Comando.Equals("atrás")))
         {
             movert1();
         }
```

Figura 33 – Código-fonte das condições para executar o método "movert1()"

Importante destacar a necessidade de reconhecer os comandos "trás" (com s) e o "traz" (com z), mesmo que o "traz"(com z), pertença ao verbo trazer, pois dependendo da maneira que o locutor fale, o *Voice Search* pode reconhecer a palavra "trás" ou "traz". Desta forma, o método "movert1()" possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada.

O terceiro método foi **"parar1()",** que tem a função de parar qualquer movimentação do *RoboDeck*, alinhando as rodas na posição de 0 grau. A Figura 34 mostra o código-fonte de sua implementação:

```
// parar

public void parar1()
{
    report(robodeck.brake(new GigaCallback(this, actionCallback)));

}
```

Figura 34 – Código-fonte do método "parar1()"

É executado se a linha "comando" do banco de dados "falaBD" for igual à "pare", ou "parar", ou "robo pare". Assim, o método "parar1()" possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada, conforme a Figura 35:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

        if ((linha.Comando.Equals("pare")) || (linha.Comando.Equals("parar")) ||
            (linha.Comando.Equals("robo pare")))
        {
            parar1();
        }
```

Figura 35 – Código-fonte das condições para executar o método "parar1()"

O quarto método foi **"virare1()",** com a função de virar o *RoboDeck* à esquerda. A Figura 36 exibe o código-fonte de sua implementação:

```
// virar a esquerta

public void virare1()
{
    sbyte angle = -32;
    short intensity = 15000; // ou 32760
    report(robodeck.turn(
    angle,
    intensity,
    new GigaCallback(this, actionCallback)));
}
```

Figura 36 – Código-fonte do método "virare1()"

A intensidade foi configurada em 15000 e o ângulo em -32, sendo que os ângulos suportados no comando robótico ***"robodeck.turn"*** variam de -32 até 32, e os negativos referem-se à esquerda, enquanto os positivos, à direita: essa configuração faz o robô mover para frente com metade da sua potência, virando à esquerda com ângulo máximo suportado.

É executado se a linha "comando" do banco de dados "falaBD" for igual à "para esquerda", ou "esquerda", ou "vire esquerda", ou "virar a esquerda", ou "virar à esquerda", ou "virar esquerda", ou "vire à esquerda", ou "vire a esquerda". Dessa forma, o método **"virare1( )"** possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada, em conformidade com a Figura 37:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

 if ((linha.Comando.Equals("para esquerda")) || (linha.Comando.Equals("esquerda"))
     || ((linha.Comando.Equals("vire esquerda")) || (linha.Comando.Equals("virar a esquerda"))
     || (linha.Comando.Equals("virar à esquerda")) || (linha.Comando.Equals("virar esquerda"))
     || (linha.Comando.Equals("vire à esquerda")) || (linha.Comando.Equals("vire a esquerda"))))
 {
     virare1();
 }
```

Figura 37 – Código-fonte das condições para executar o método "virare1()"

O quinto método foi **"virard1()",** que tem a função de virar o *RoboDeck* à direita. A Figura 38 mostra o código-fonte de sua implementação:

```
// virar a direita

public void virard1()
{
    sbyte angle = 32;
    short intensity = 15000; // ou é 32760
    report(robodeck.turn(
    angle,
    intensity,
    new GigaCallback(this, actionCallback)));
}
```

Figura 38 – Código-fonte do método "virard1()"

Observando essa Figura, constata-se que a intensidade foi configurada em 15000 e o ângulo em 32, lembrando-se que os ângulos suportados no comando robótico ***"robodeck.turn"*** variam de -32 até 32: essa configuração faz o robô mover para frente com metade da sua potência, virando à direita no ângulo máximo suportado.

É executado se a linha "comando" do banco de dados "falaBD" for igual à "para direita", ou "direita", ou "vire direita", ou "virar a direita", ou "virar à direita", ou "virar direita", ou "vire à direita", ou "vire a direita". Diante disso, o método "virard1()" possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada, de acordo com a Figura 39:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

    if ((linha.Comando.Equals("para direita")) || (linha.Comando.Equals("direita"))
        || ((linha.Comando.Equals("vire direita")) || (linha.Comando.Equals("virar a direita"))
        || (linha.Comando.Equals("virar à direita")) || (linha.Comando.Equals("virar direita"))
        || (linha.Comando.Equals("vire à direita")) || (linha.Comando.Equals("vire a direita"))))
    {
        virard1();
    }
```

Figura 39 – Código fonte das condições para executar o método "virard1()"

O sexto método foi **"girare1()",** com a função de girar o *RoboDeck* no seu próprio eixo à esquerda. A Figura 40 exibe o código-fonte dessa implementação:

```
// girar a esquerda
public void girare1()
{
    short intensity = 15000; // ou é 32760
    report(robodeck.roll(
    rdsdk.Side.LEFT,
    intensity,
    new GigaCallback(this, actionCallback)));
}
```

Figura 40 – Código-fonte do método "girare1()"

Vê-se que a intensidade foi configurada em 15000 e o ***"rdsgk.Side"*** em "***LEFT"***: essa configuração faz o robô girar no próprio eixo à esquerda com metade da sua potência.

É executado se a linha "comando" do banco de dados "falaBD" for igual à "gire esquerda", ou "girar a esquerda", ou "girar à esquerda", ou "girar esquerda", ou "gire à esquerda", ou "gire a esquerda". Assim, o método "girare1()" possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada, conforme mostra a Figura 41.

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

  if ((linha.Comando.Equals("gire esquerda")) || (linha.Comando.Equals("girar a esquerda"))
      || (linha.Comando.Equals("girar à esquerda")) || (linha.Comando.Equals("girar esquerda"))
      || (linha.Comando.Equals("gire à esquerda")) || (linha.Comando.Equals("gire a esquerda")))
  {
      girare1();
  }
```

Figura 41 – Código-fonte das condições para executar o método "girare1()"

O sétimo e último método utilizado nesse experimento foi **"girard1()",** o qual tem a função de girar o *RoboDeck* no seu próprio eixo à direita. A Figura 42 apresenta o código-fonte de sua implementação:

```
// girar a direita
public void girard1()
{
    short intensity = 15000; // ou é 3276
    report(robodeck.roll(
    rdsdk.Side.RIGHT,
    intensity,
    new GigaCallback(this, actionCallback)));
}
```

Figura 42 – Código-fonte do método "girard1()"

A intensidade foi configurada em 15000 e o ***"rdsgk.Side"*** em "***RIGHT"***: essa configuração faz o robô girar no próprio eixo à direita com metade da sua potência.

É executado se a linha "comando" do banco de dados "falaBD" for igual à "gire direita", ou "girar a direita", ou "girar à direita", ou "girar direita", ou "gire à direita", ou "gire a direita". Desse modo, o método "girard1()" possui mais opções de fala para ser executado, diminuindo o percentual de erros no reconhecimento da sentença falada, em conformidade com a Figura 43:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

  if ((linha.Comando.Equals("gire direita")) || (linha.Comando.Equals("girar a direita"))
      || (linha.Comando.Equals("girar à direita")) || (linha.Comando.Equals("gire à direita"))
      || (linha.Comando.Equals("girar direita")) || (linha.Comando.Equals("gire a direita")))
  {
      girard1();
  }
```

Figura 43 – Código-fonte das condições para executar o método “girard1()”

6.1.6 Análise do tempo de processamento

Foram realizados vários testes, utilizando as mesmas velocidade de conexão de internet (2Mb/s) e sentença de fala (“para frente”): em cada dia, obtiveram-se tempos de processamento completamente distintos, variando de milissegundos a alguns segundos, em virtude de a API *Voice Search* (biblioteca usada no reconhecimento da sentença) depender da disponibilidade do servidor da Google, algo impossível prever.

6.1.7 Sentenças reconhecidas pelo *Voice Search*

Para obter uma amostragem de dados, utilizaram-se três pessoas, sendo dois homens “P1” e “P2” e uma mulher “P3”. Cada pessoa tinha de repetir vinte vezes uma mesma sentença para a movimentação do robô, a qual foi dividida em dois ambientes sonoros: dez sentenças num ambiente sem ruído sonoro e as outras dez, com ruídos. Assim, fez-se uma amostragem de dados com os números de sentenças reconhecidas e não reconhecidas pelo *Voice Search*.

O Gráfico 1 traz as amostras de dados das sentenças de fala “mover”, “para cima”, “mova” e “para frente”, num ambiente sem ruído sonoro.

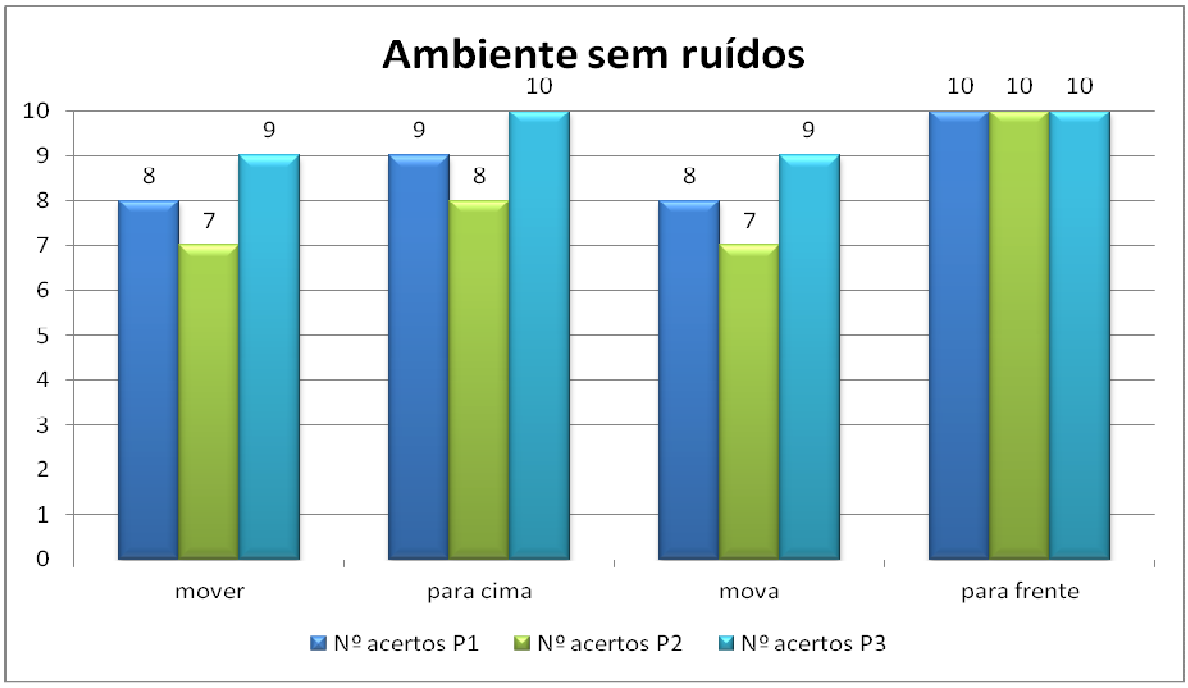


Gráfico 1 – Amostras de dados das sentenças de movimentação do robô para frente, via Voice Search, em um ambiente sem ruídos sonoros.

Observa-se que as sentenças tiveram uma média de acertos de: “mover” 80%, “para cima” 90%, “mova” 80% e “para frente” 100%, nos seus reconhecimentos. Dessa forma, a sentença “para cima” obteve a melhor média de acertos em um ambiente sem ruídos sonoros.

O Gráfico 2 apresenta as amostras de dados das sentenças de fala “mover”, “para cima”, “mova” e “para frente”, num ambiente com ruído sonoro.

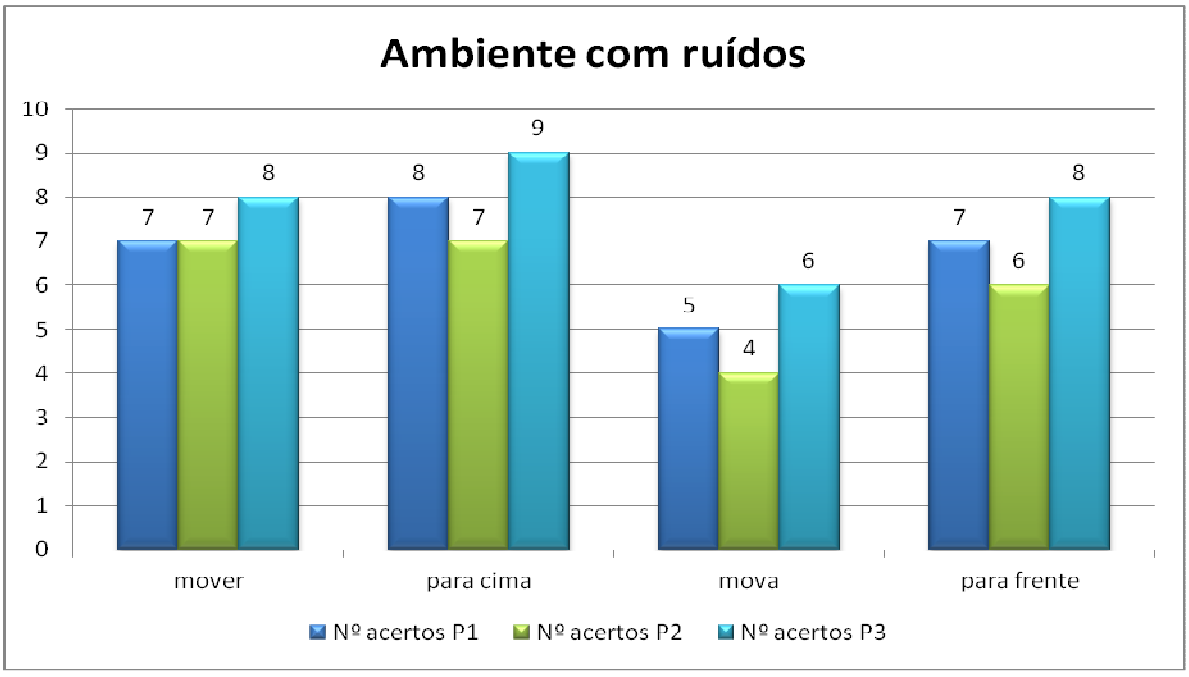


Gráfico 2 – Amostras de dados das sentenças de movimentação do robô para frente, via Voice Search, em um ambiente com ruídos sonoros.

Verifica-se uma média de acertos das sentenças de: “mover” 73%, “para cima” 80%, “mova” 60% e “para frente” 70%, nos seus reconhecimentos. Portanto, a sentença “para cima” obteve a melhor média de acertos e a sentença “para frente”, uma baixa devido aos ruídos sonoros.

O Gráfico 3 exibe as amostras de dados das sentenças de fala “para trás”, “atrás” e “mover para trás”, num ambiente sem ruído sonoro.

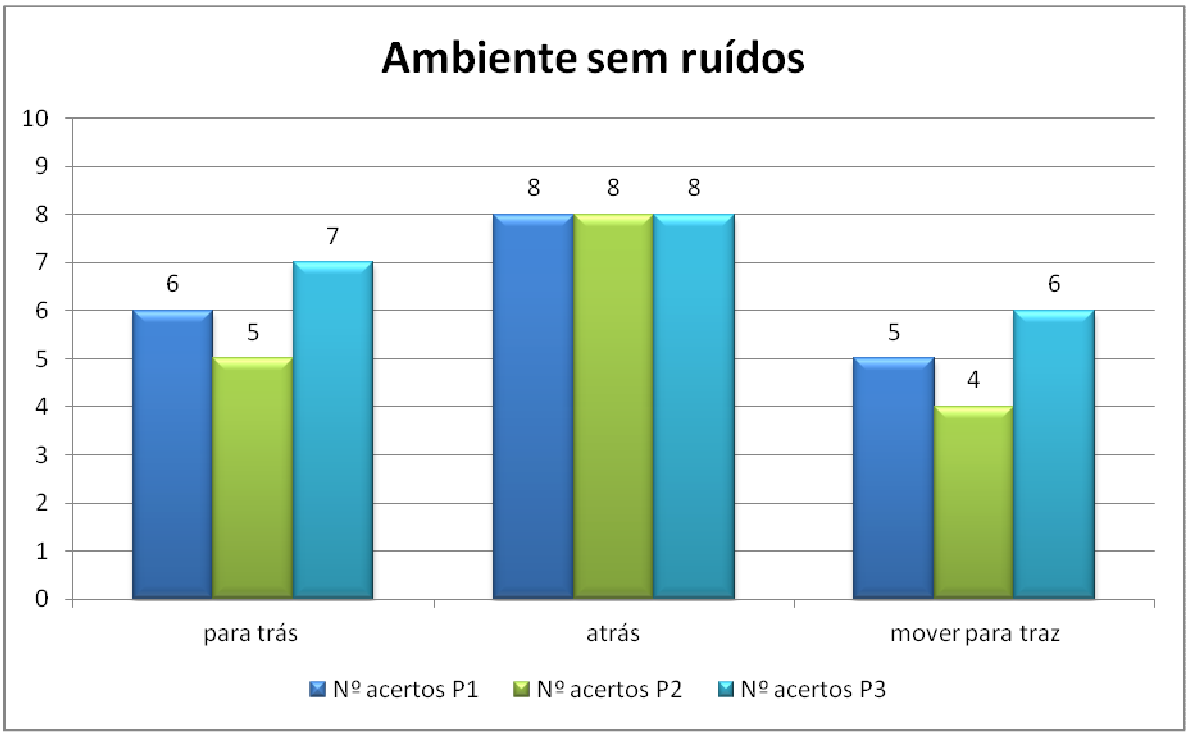


Gráfico 3 – Amostras de dados das sentenças de movimentação do robô para trás, via *Voice Search,* em um ambiente sem ruídos sonoros.

As sentenças obtiveram uma média de acertos, nos seus reconhecimentos, de: “para trás” 60%, “atrás” 80% e “mover para trás” 60%. Desse modo, a sentença “atrás” obteve a melhor média de acertos em um ambiente sem ruídos sonoros.

O Gráfico 4 expõe as amostras de dados das sentenças de fala “para trás”, “atrás” e “mover para trás”, num ambiente com ruído sonoro.

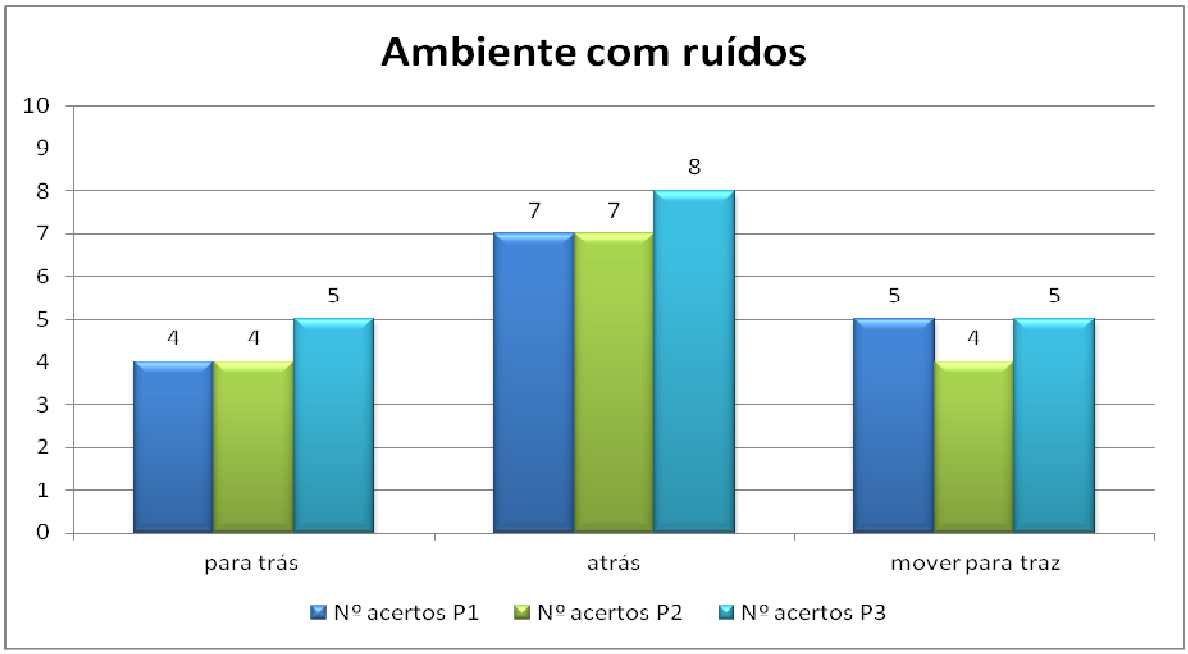


Gráfico 4 – Amostras de dados das sentenças de movimentação do robô para trás, via *Voice Search*, em um ambiente com ruídos sonoros.

Constata-se que as sentenças apresentaram uma média de acertos de: “para trás” 43%, “atrás” 73% e “mover para trás” 46%, nos seus reconhecimentos. A sentença “para trás” obteve uma média maior de erros 57%, em razão dos ruídos sonoros, enquanto a sentença “atrás” atingiu uma média maior de acertos.

Estão no Gráfico 5, as amostras de dados das sentenças de fala “parar”, “pare” e “robô pare”, num ambiente sem ruído sonoro.

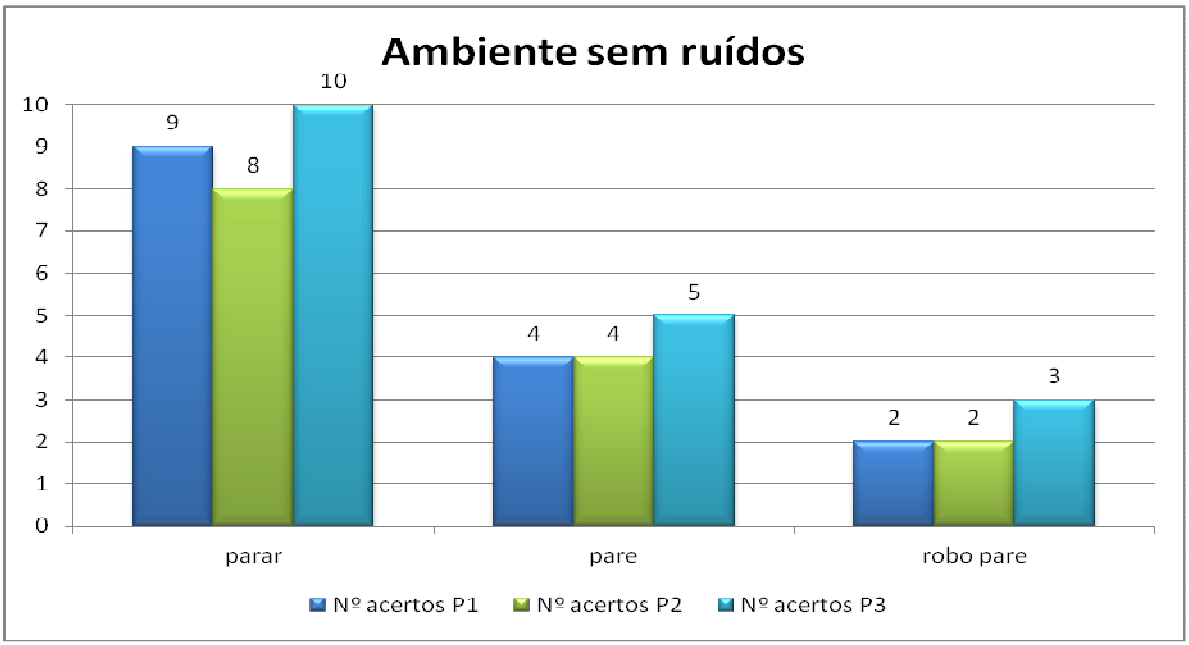


Gráfico 5 – Amostras de dados das sentenças que param a movimentação do robô via, *Voice Search*, em um ambiente sem ruídos sonoros.

As sentenças mostraram uma média de acertos de: “parar” 90%, “pare” 43% e “robô pare” 23%, nos seus reconhecimentos. Assim, a sentença “parar” alcançou a melhor média de acertos em um ambiente sem ruídos sonoros.

O Gráfico 6 aponta as amostras de dados das sentenças de fala “parar”, “pare” e “robô pare”, num ambiente com ruído sonoro.

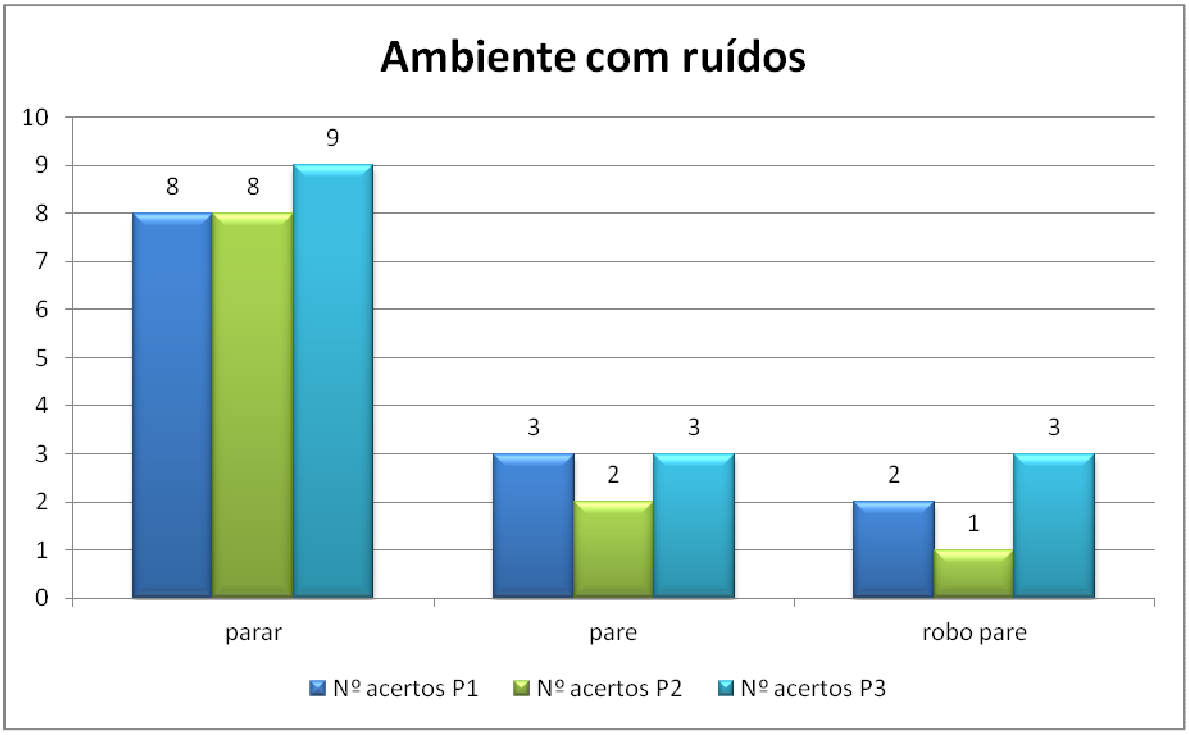


Gráfico 6 – Amostras de dados das sentenças que param a movimentação do robô, via *Voice Search,* em um ambiente com ruídos sonoros.

Observa-se uma média de acertos das sentenças de: “parar” 83%, “pare” 26% e “robô pare” 20%, nos seus reconhecimentos. Dessa forma, a sentença “parar” manteve uma média alta de acertos, mesmo no ambiente com ruído sonoro.

O Gráfico 7 traz as amostras de dados das sentenças de fala “esquerda”, “virar à esquerda”, “para esquerda” e “vire à esquerda”, num ambiente sem ruído sonoro.

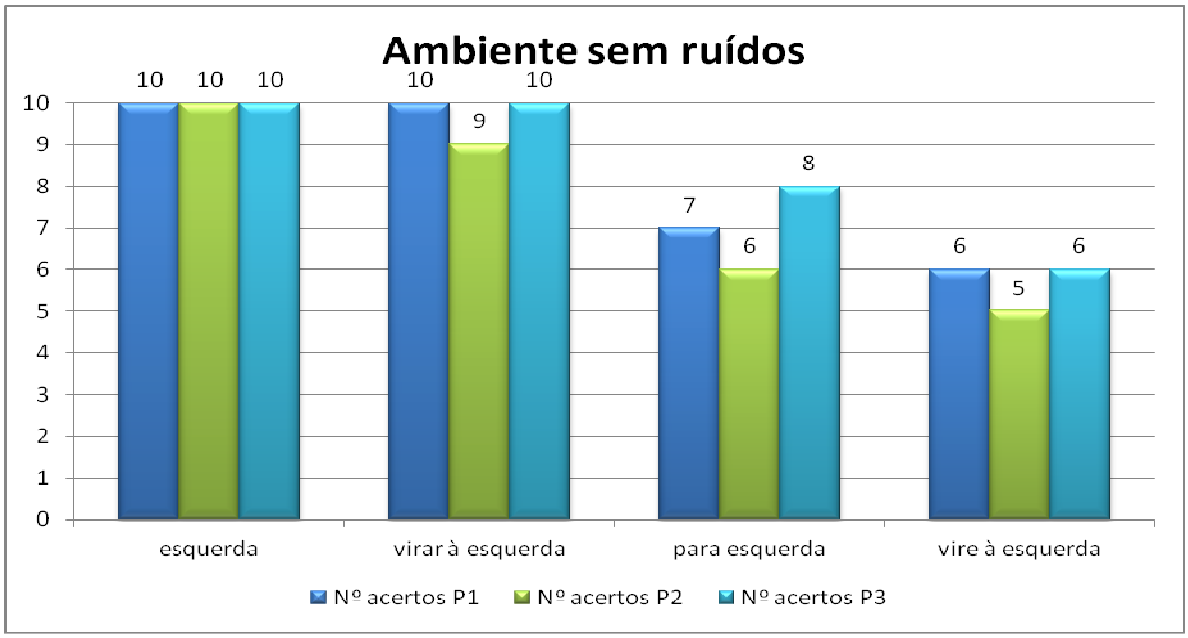


Gráfico 7– Amostras de dados das sentenças que viram o robô à esquerda via, *Voice Search,* em um ambiente sem ruídos sonoros.

As sentenças alcançaram uma média de acertos de: “esquerda” 100%, “virar à esquerda” 96%, “para esquerda” 70% e “vire à esquerda” 56%, nos seus reconhecimentos. Portanto, a sentença “esquerda” conseguiu a melhor média de acertos, em um ambiente sem ruídos sonoros.

No Gráfico 8, visualizam-se as amostras de dados das sentenças de fala “esquerda”, “virar à esquerda”, “para esquerda” e “vire à esquerda”, num ambiente com ruído sonoro.

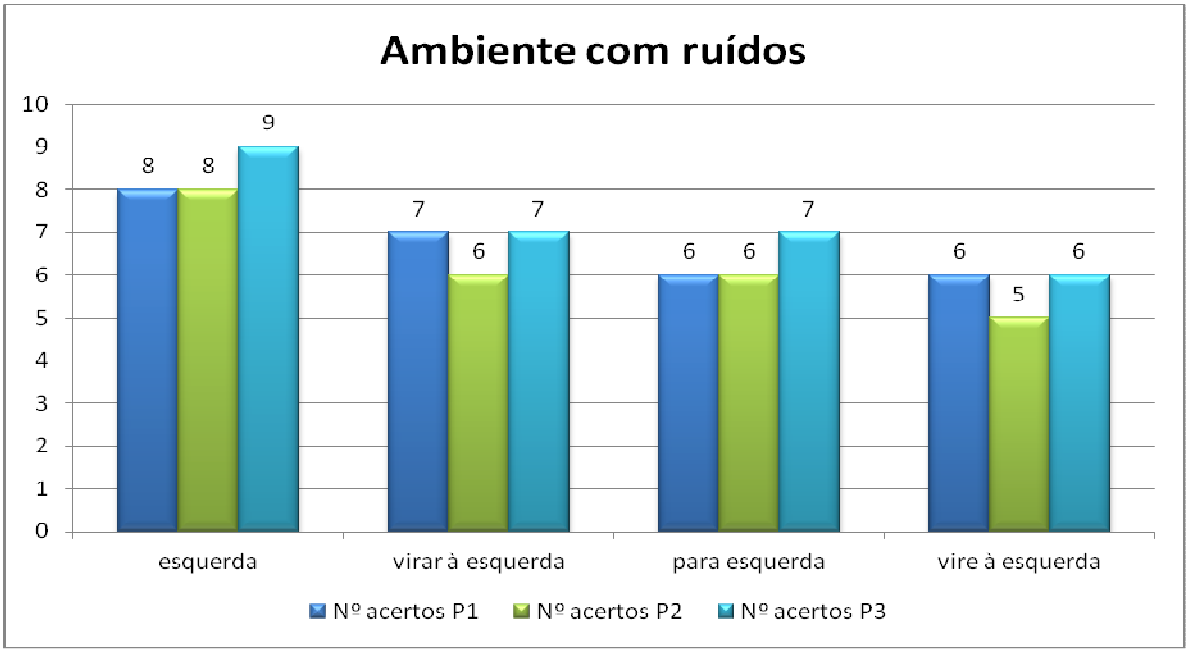


Gráfico 8 – Amostras de dados das sentenças que viram o robô à esquerda, via *Voice Search,* em um ambiente com ruídos sonoros.

As sentenças atingiram uma média de acertos de: “esquerda” 83%, “virar à esquerda” 66%, “para esquerda” 63% e “vire à esquerda” 56%, nos seus reconhecimentos. Desse modo, a sentença “esquerda” obteve a melhor média de acertos em um ambiente com ruídos sonoros.

O Gráfico 9 exibe as amostras de dados das sentenças de fala “direita”, “virar à direita”, “para direita” e “vire à direita”, num ambiente sem ruído sonoro.

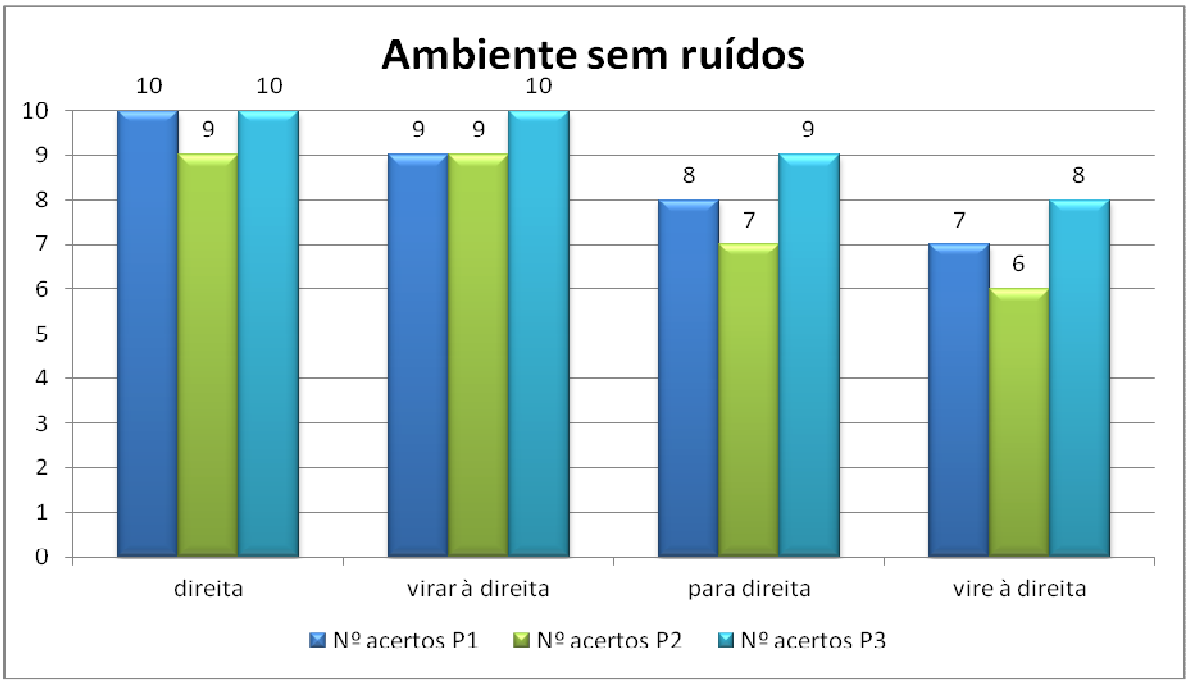


Gráfico 9 – Amostras de dados das sentenças que viram o robô à direita, via *Voice Search,* em um ambiente sem ruídos sonoros.

Nota-se que as sentenças tiveram uma média de acertos de: “direita” 96%, “virar à direita” 93%, “para direita” 80% e “vire à direita” 70%, nos seus reconhecimentos. Então, a sentença “direita” atingiu a melhor média de acertos em um ambiente sem ruídos sonoros.

O Gráfico10 expõe as amostras de dados das sentenças de fala “direita”, “virar à direita”, “para direita” e “vire à direita”, num ambiente com ruído sonoro.

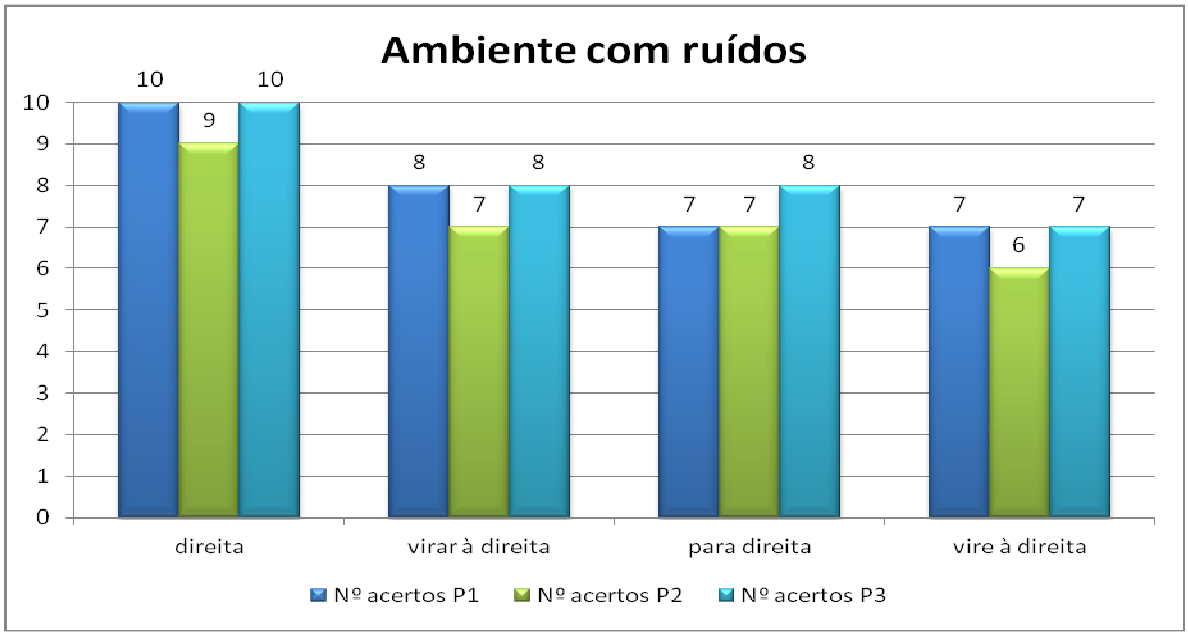


Gráfico 10 – Amostras de dados das sentenças que viram o robô à direita, via *Voice Search,* em um ambiente com ruídos sonoros.

As sentenças mostraram uma média de acertos de: “direita” 96%, “virar à direita” 76%, “para direita” 73% e “vire à direita” 66%, nos seus reconhecimentos. Assim, a sentença “direita” foi à única que manteve a mesma média de acertos, em um ambiente com ruídos sonoros.

No Gráfico 11, encontram-se as amostras de dados das sentenças de fala “girar à esquerda”, “gire esquerda”, “girar esquerda” e “gire à esquerda”, num ambiente sem ruído sonoro.

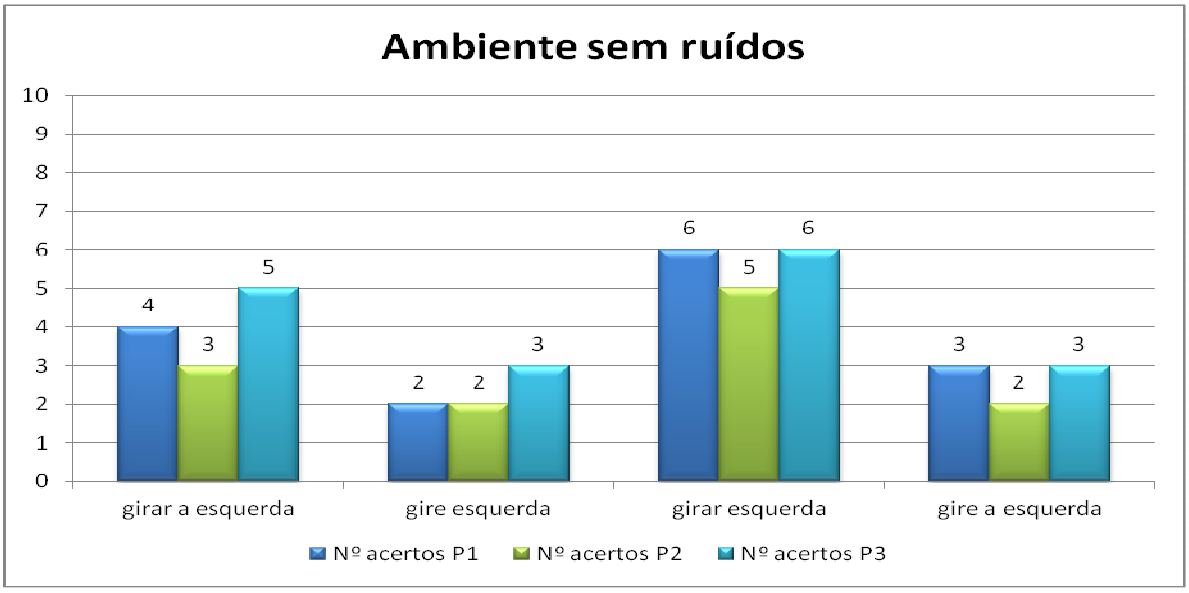


Gráfico 11 – Amostras de dados das sentenças que giram o robô no próprio eixo à esquerda, via *Voice Search,* em um ambiente sem ruídos sonoros.

As sentenças apresentaram uma média de acertos de: “girar à esquerda” 40%, “gire esquerda” 23%, “girar esquerda” 56% e “gire à esquerda” 26%, nos seus reconhecimentos. Dessa forma, a sentença “girar esquerda” obteve uma média maior de acertos, em um ambiente sem ruídos sonoros.

Seguem no Gráfico 12, as amostras de dados das sentenças de fala “girar a esquerda”, “gire esquerda”, “girar esquerda” e “gire à esquerda”, num ambiente com ruído sonoro.

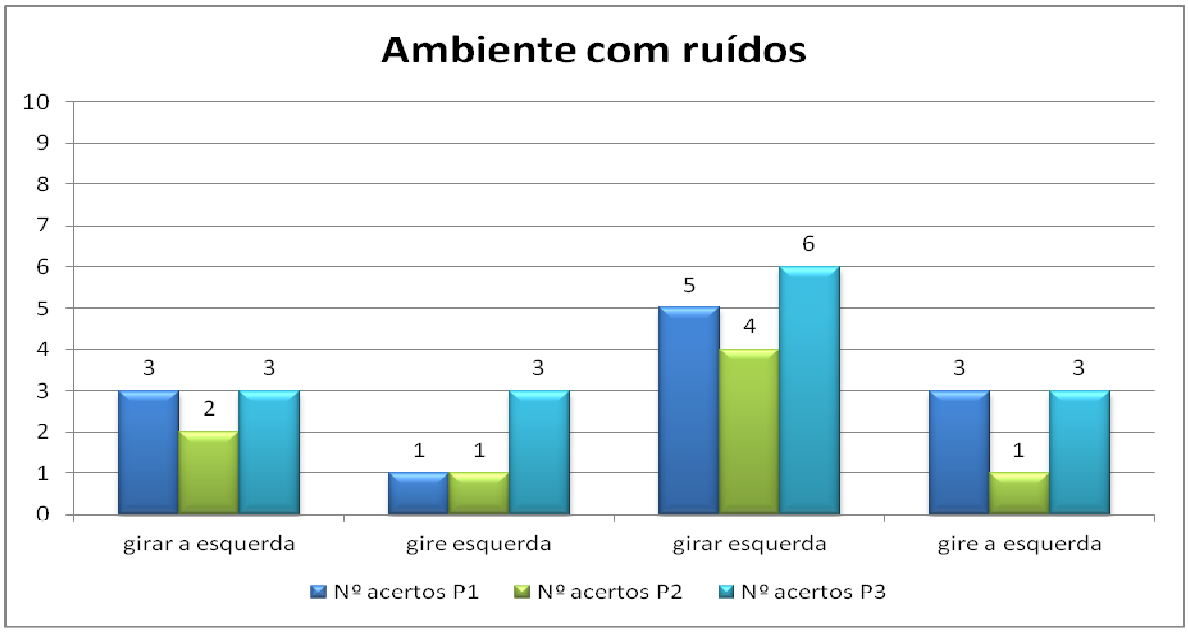


Gráfico 12– Amostras de dados das sentenças que giram o robô no próprio eixo à esquerda, via *Voice Search,* em um ambiente com ruídos sonoros.

As sentenças alcançaram uma média de acertos de: “girar a esquerda” 26%, “gire esquerda” 16%, “girar esquerda” 50% e “gire a esquerda” 23%, nos seus reconhecimentos. Portanto, quase todas as sentenças tiveram um percentual maior de erros no ambiente com ruído sonoro, sendo que a única a manter a média de acertos e erros foi a “girar esquerda”.

O Gráfico 13 representa as amostras de dados das sentenças de fala “girar a direita”, “gire direita”, “girar direita” e “gire a direita”, num ambiente sem ruído sonoro.

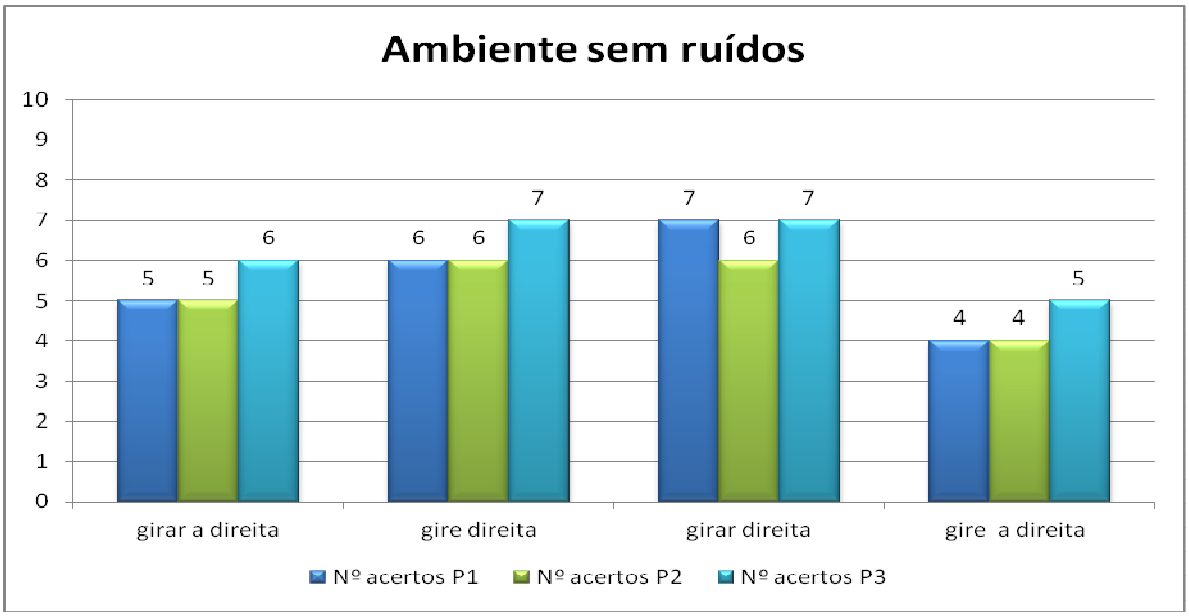


Gráfico 13 – Amostras de dados das sentenças que giram o robô no próprio eixo à direita, via *Voice Search,* em um ambiente sem ruídos sonoros.

As sentenças atingiram uma média de acertos de: “girar a direita” 53%, “gire direita” 63%, “girar direita” 66% e “gire a direita” 43%, nos seus reconhecimentos. Desse modo, a sentença “girar direita” obteve uma média maior de acertos e a sentença “gire a direita”, uma média maior de erros 57%, em um ambiente sem ruídos sonoros.

No Gráfico 14, exibem-se as amostras de dados das sentenças de fala “girar a direita”, “gire direita”, “girar direita” e “gire a direita”, num ambiente com ruído sonoro.

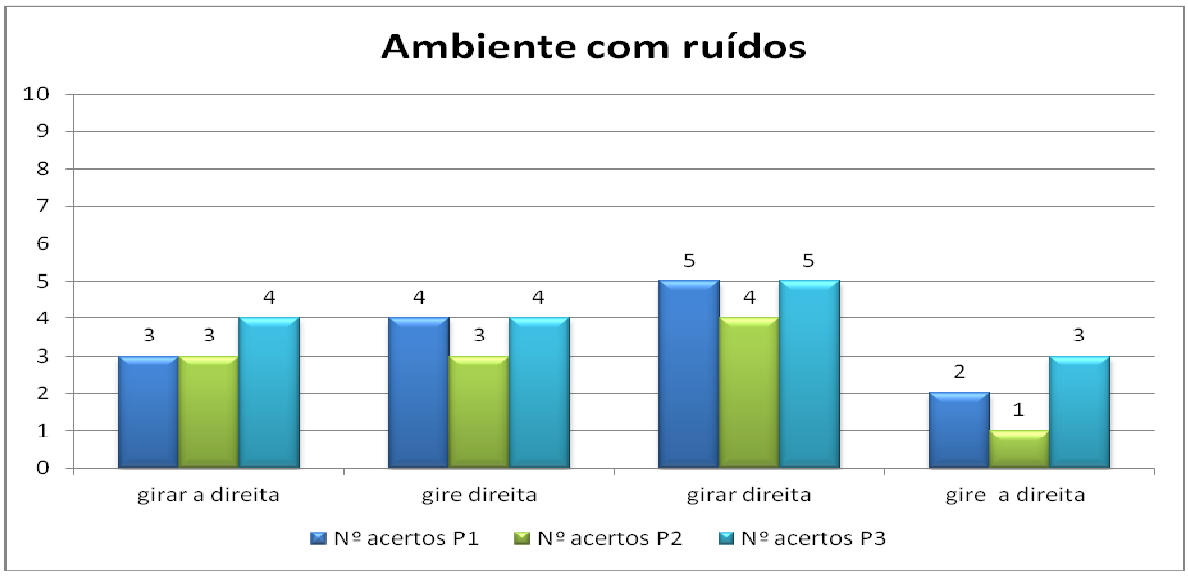


Gráfico 14 – Amostras de dados das sentenças que giram o robô no próprio eixo à direita, via *Voice Search,* em um ambiente com ruídos sonoros.

As sentenças expuseram uma média de acertos de: “girar à direita” 33%, “gire direita” 36%, “girar direita” 46% e “gire à direita” 20%, nos seus reconhecimentos. Então, todas as sentenças adquiriram um percentual maior de erros no ambiente com ruído sonoro, sendo que a sentença “girar direita” ficou com a menor média de erros 54%.

6.1.8 Dificuldades Encontradas e Análise dos Resultados

A dificuldade averiguada foi manter o *RoboDeck* em linha reta, nos comandos para frente e para trás, pois conforme ele se movimentava, ocorria um pequeno deslocamento para a esquerda. Após realizar vários testes e analisar os resultados obtidos, constatou-se que o problema estava no alinhamento das rodas do robô.

6.2 Experimento 2: Girar no próprio eixo em 90 graus via módulo de controle de voz externo.

6.2.1 Objetivo

Demonstrar a movimentação do *RoboDeck,* girando no próprio eixo em um ângulo de 90 graus à esquerda e à direita. Nesse experimento o robô girou no próprio eixo à esquerda até atingir o ângulo de 90 graus e parou, alinhando as rodas. Da mesma forma, o robô girou no próprio eixo à direita até atingir o ângulo de 90 graus e parou, alinhando as rodas.

6.2.2 Ambiente físico

Realizou-se no mesmo local, com iguais condições do experimento 1.

6.2.3 Recursos tecnológicos utilizados

Foram utilizados os mesmo quatro recursos já descritos no experimento1, ou seja, um notebook, um celular, um roteador Sem fio e um *RoboDeck*.

6.2.4 Comandos robóticos utilizados

Os dois comandos robóticos de movimentação do SKD Giga de Testes, mencionados no experimento 1, o ***"robodeck.brake"*** e o ***"robodeck.roll"***, tornaram a ser empregados.

6.2.5 Implementação

Usaram-se os três métodos citados no experimento 1: **"girare1()", "girard1()"** e **"parar1()"**.

Para girar o *RoboDeck* no seu próprio eixo à esquerda até atingir o ângulo de 90 graus e parar, alinhando as rodas, foi necessário executar o método "**girare()",** por um tempo 1600 milissegundos, sendo que esse método já estava configurado numa intensidade de 15000 e na sequencia foi executado o método "**para1()".**

Observando a Figura 44, nota-se que esse experimento era executado, se a linha "comando" do banco de dados "falaBD" fosse igual à "esquerda 90".

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

    if (linha.Comando.Equals("esquerda 90"))
    {
        girare1();// método girar à esquerda
        System.Threading.Thread.Sleep(1600);//parar a execução em 1500 milisegundos
        parar1();// método parar
    }
```

Figura 44 – Código-fonte da condição para executar o experimento girar 90 graus à esquerda

Para girar o *RoboDeck* no seu próprio eixo à direita até atingir o ângulo de 90 graus e parar, alinhando as rodas, executou-se o método "**girard()",** por um tempo 1550 milissegundos, estando também esse método configurado com uma intensidade de 15000 e na sequencia foi executado o método "**para1()".**

Com base na Figura 45, verifica-se que esse experimento era executado, se a linha "comando" do banco de dados "falaBD" fosse igual à "direita 90".

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

    if (linha.Comando.Equals("direita 90"))
    {
        girard1();// método girar à direita
        System.Threading.Thread.Sleep(1550);//para a execução em 1500 milisegundos
        parar1();// método parar
    }
```

Figura 45 – Código-fonte da condição para executar o experimento girar 90 graus à direita

6.2.6 Sentenças reconhecidas pelo *Voice Search*

Os critérios, para obter as amostras de dados nesse experimento, foram os mesmos já descritos no Experimento1.

O Gráfico 15 apresenta as amostras de dados das sentenças de fala “esquerda 90”, em um ambiente sem ruído sonoro.

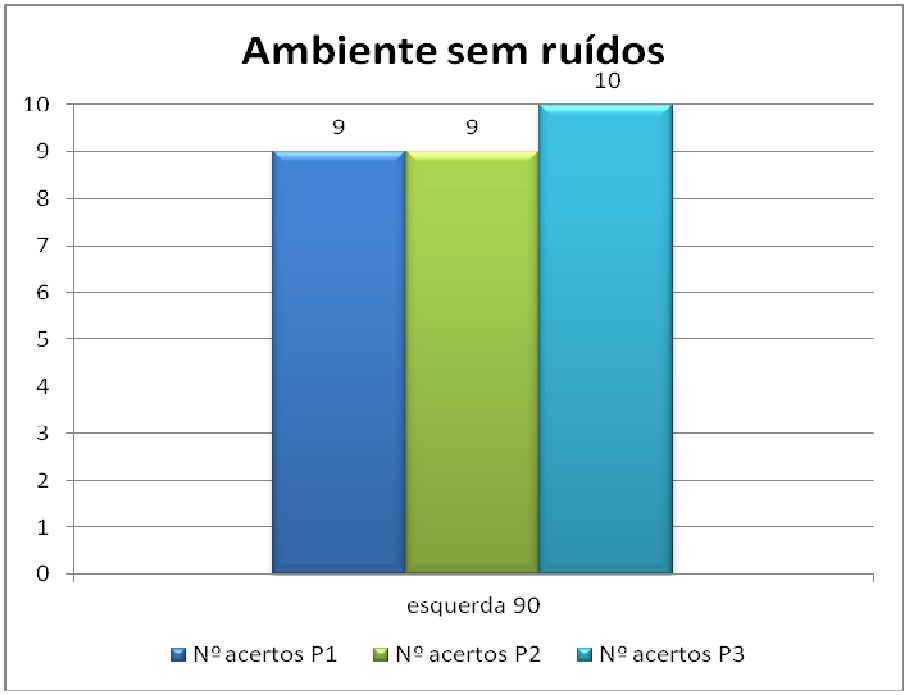


Gráfico 15– Amostras de dados das sentenças que giram o robô no próprio eixo 90 graus à esquerda, via *Voice Search,* em um ambiente sem ruídos sonoros.

A sentença “esquerda 90” atingiu uma média de acertos de 93%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

Encontram-se no Gráfico 16, as amostras de dados das sentenças de fala “esquerda 90”, num ambiente com ruído sonoro.

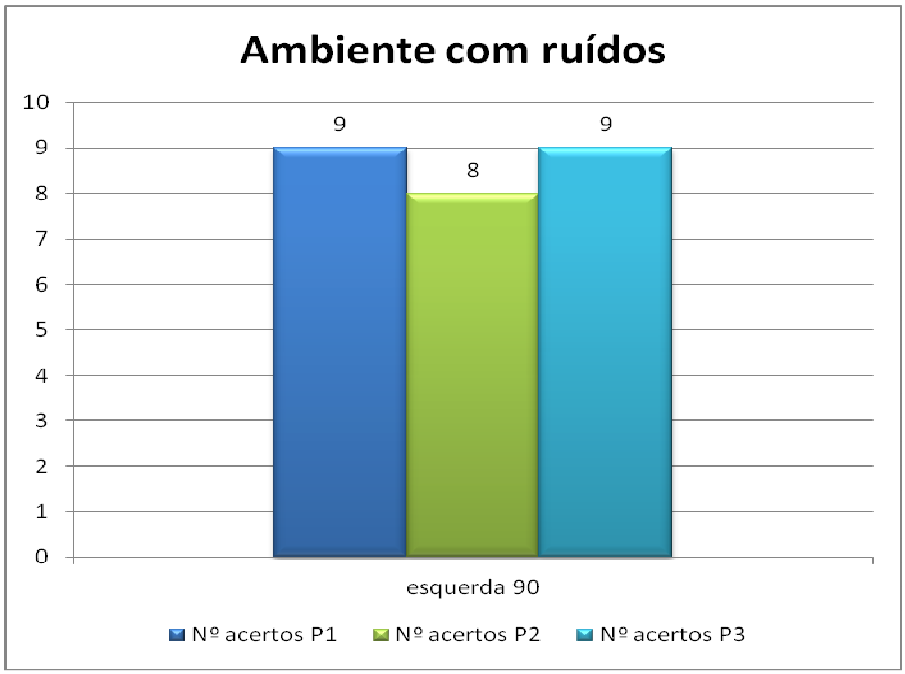


Gráfico 16 – Amostras de dados das sentenças que giram o robô no próprio eixo 90 graus à esquerda, *via Voice Search,* em um ambiente com ruídos sonoros.

A sentença “esquerda 90” alcançou uma média de acertos de 86%, no reconhecimento de fala em um ambiente com ruído sonoro.

O Gráfico 17 expõe as amostras de dados das sentenças de fala “direita 90”, num ambiente sem ruído sonoro.

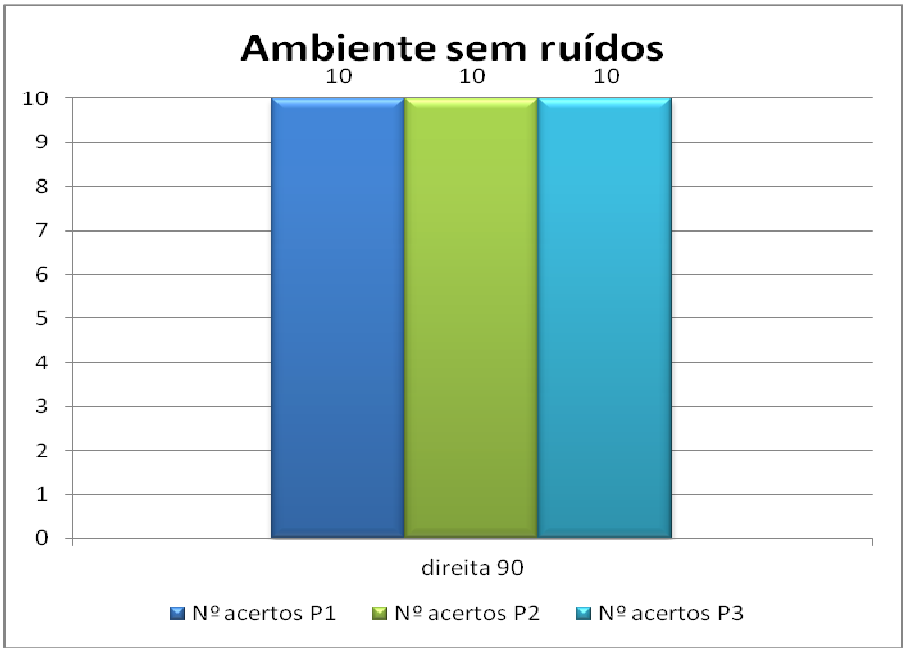


Gráfico 17– Amostras de dados das sentenças que giram o robô no próprio eixo 90 graus à direita, via *Voice Search,* em um ambiente sem ruídos sonoros.

A sentença “direita 90” teve uma média de acertos de 100%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

No Gráfico 18, visualizam-se as amostras de dados das sentenças de fala “direita 90”, num ambiente com ruído sonoro.

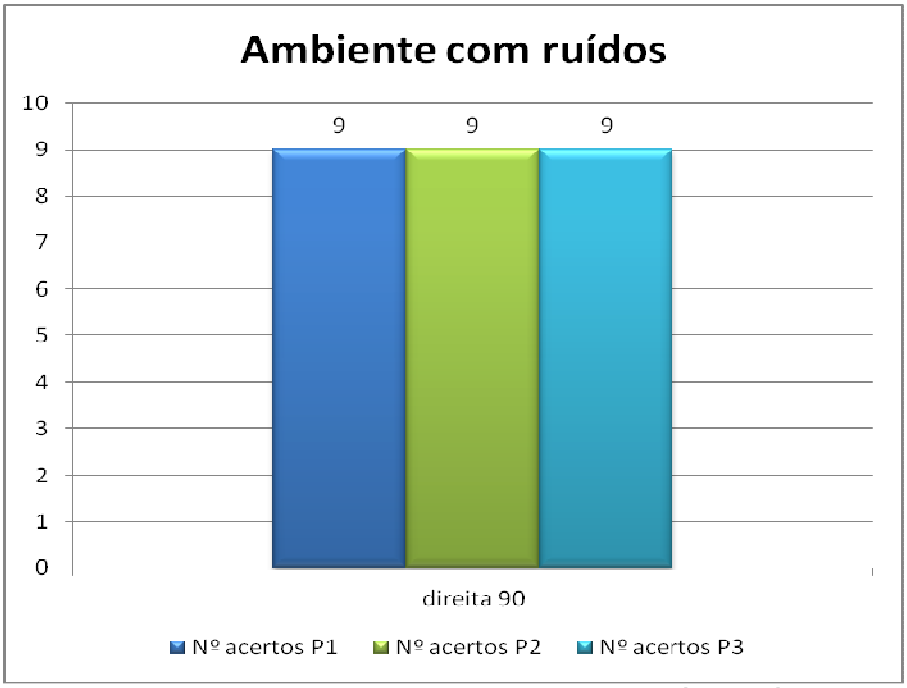


Gráfico 18 – Amostras de dados das sentenças que giram o robô no próprio eixo 90 graus à direita, via *Voice Search,* em um ambiente com ruídos sonoros.

Verifica-se que a sentença “direita 90” apresentou uma média de acertos de 90%, em um ambiente com ruído sonoro.

6.2.7 Dificuldades Encontradas e Análise dos Resultados

Houve dificuldade em definir o tempo de execução do método girard1() ou girare1(), uma vez que esse tempo tinha de ser exato, para que o robô pudesse atingir o ângulo de 90 graus, pois o ângulo desejado era obtido em função do tempo de acionamento do motor. Mas se a intensidade fosse alterada, o tempo necessário para se conseguir o ângulo também deveria ser modificado.

Depois de realizar vários testes e analisar os resultados obtidos, compreendeu-se a necessidade de levar em consideração, além da intensidade de 15000 do método, o tempo que o *RoboDeck* precisava para responder aos comandos.

6.3 Experimento 3: Movimentação em circuito via módulo de controle de voz externo.

6.3.1 Objetivo

Demonstrar a movimentação do *RoboDeck* num circuito com formato quadrado, via reconhecimento de fala por meio de um aparelho móvel. Nesse experimento, o robô sai de um ponto, movimentando-se para frente por um tempo, para, gira no próprio eixo 90 graus à esquerda, para, volta a se movimentar para frente por um mesmo tempo, repetindo tal sequência até completar o circuito.

6.3.2 Ambiente físico

Ocorreu no mesmo local, com iguais condições do experimento 1.

6.3.3 Recursos tecnológicos utilizados

Utilizaram-se os quatro recursos já descritos no experimento1: um notebook, um celular, um roteador Sem fio e um *RoboDeck*.

6.3.4 Comandos robóticos utilizados

Três comandos robóticos de movimentação do SDK Giga de Testes, já descritos no experimento 1, ***"robodeck.brake", "robodeck.move"*** e ***"robodeck.roll"***, voltaram a ser aplicados.

6.3.5 Implementação

Usaram-se três métodos presentes no experimento 1, **"moverf1()", "parar1()"** e **"girare1()".**

O experimento utilizou uma sequencia de métodos já desenvolvidos nos experimentos 1 e 2, cumprindo a seguinte rotina: mover o robô para frente com intensidade de 32760 por 5000 milissegundos, em seguida parar, girar o *RoboDeck* no seu próprio eixo à esquerda com intensidade de 15000, por um tempo de 1500 milissegundos, atingindo o ângulo de 90 graus e parar, alinhando as rodas. Essa sequencia foi repetida até completar o quadrado.

A Figura 46 relata a quantidade de sequencias necessária para completar o quadrado e que o experimento era executado, se a linha “comando” do banco de dados “falaBD” fosse igual a “circuito”:

```
// comando andar no circuito em forma de um quadrado
if (linha.Comando.Equals("circuito"))
{
    moverf1();  // para frente
    System.Threading.Thread.Sleep(5000);//para a execução em 5000 milisegundos
    parar1(); // parar
    girare1();// girar esquerda
    System.Threading.Thread.Sleep(1500);//para a execução em 1500 milisegundos
    parar1(); // parar
    moverf1();  // para frente
    System.Threading.Thread.Sleep(5000);//para a execução em 5000 milisegundos
    parar1(); // parar
    girare1();// girar esquerda
    System.Threading.Thread.Sleep(1500);//para a execução em 1500 milisegundos
    parar1(); // parar
    moverf1();  // para frente
    System.Threading.Thread.Sleep(5000);//para a execução em 5000 milisegundos
    parar1(); // parar
    girare1();// girar esquerda
    System.Threading.Thread.Sleep(1500);//para a execução em 1500 milisegundos
    parar1();
    moverf1();  //para frente
    System.Threading.Thread.Sleep(5000);//para a execução em 5000 milisegundos
    parar1(); // parar
    girare1();// girar esquerda
    System.Threading.Thread.Sleep(1500);//para a execução em 1500 milisegundos
    parar1();

}
```

Figura 46 – Código-fonte da condição para executar o experimento circuito

6.3.6 Sentença reconhecida pelo *Voice Search*

Os critérios, para obter as amostras de dados nesse experimento, foram os mesmos já descritos no Experimento1.

O Gráfico 19 exibe as amostras de dados da sentença de fala “circuito”, num ambiente sem ruído sonoro.

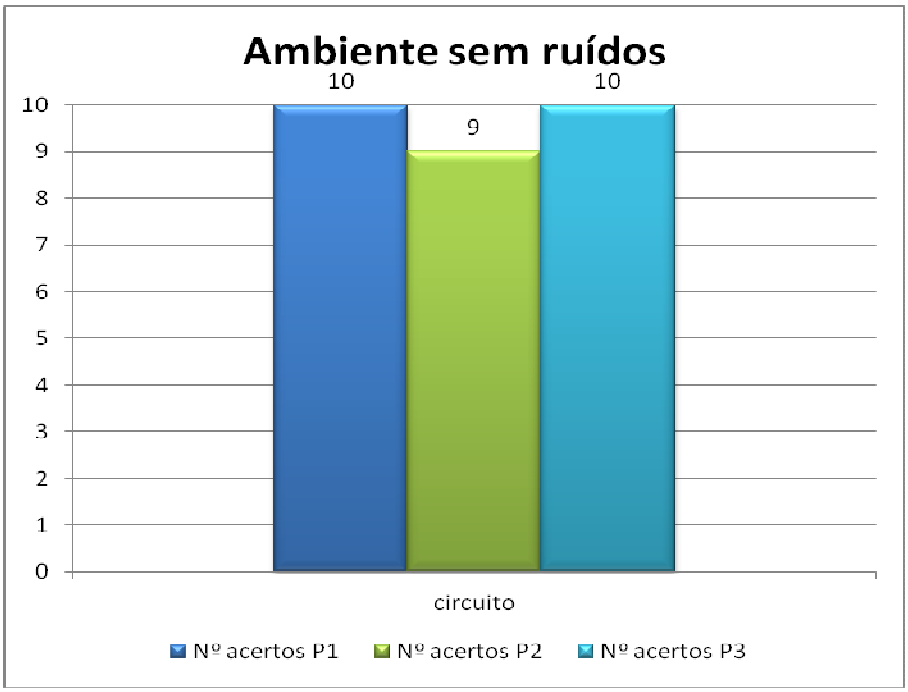


Gráfico 19 – Amostras de dados da sentença que faz o robô se movimentar no circuito, via *Voice Search,* em um ambiente sem ruídos sonoros.

A sentença "circuito" atingiu uma média de acertos de 96%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

O Gráfico 20 ilustra as amostras de dados da sentença de fala "circuito", num ambiente com ruído sonoro.

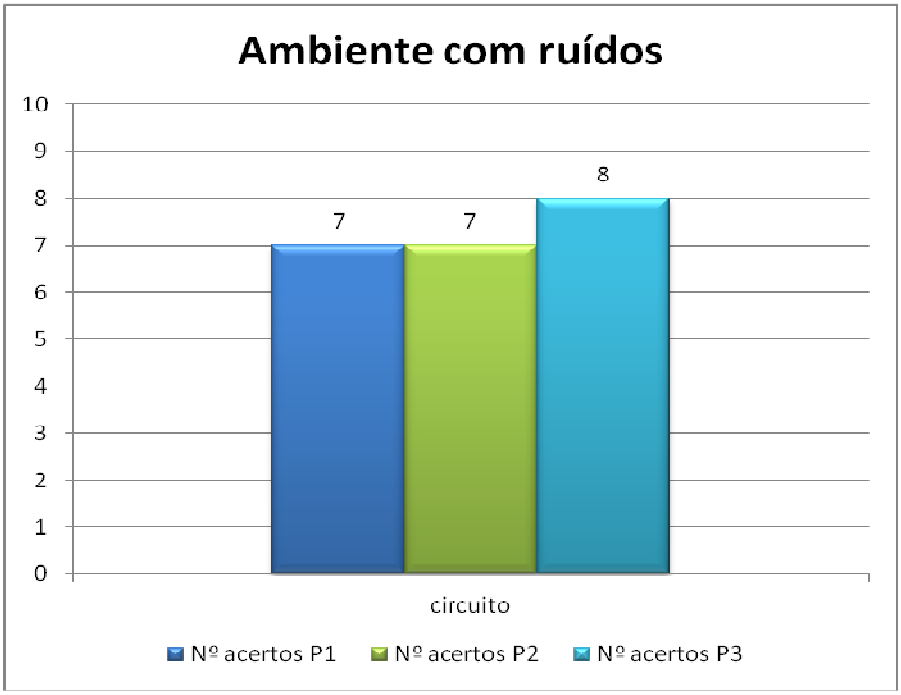


Gráfico 20 – Amostras de dados da sentença que faz o robô se movimentar no circuito, via *Voice Search,* em um ambiente com ruídos sonoros.

A sentença “circuito” teve uma média de acertos de 73%, no ambiente com ruído sonoro.

6.3.7 Dificuldades Encontradas e Análise dos Resultados

Duas dificuldades foram percebidas: manter o *RoboDeck* em linha reta, no comando para frente, pois conforme ele se movimentava, ocorria um pequeno deslocamento à esquerda, sendo detectado, após vários testes, que o problema estava no alinhamento das rodas do robô; definir o tempo de execução do método girare1(), pois o tempo tinha de ser exato, para o robô atingir o ângulo de 90 graus. Portanto, foi necessário levar em consideração, além da intensidade de 15000 do método, o tempo que o *RoboDeck* precisava para responder aos comandos.

6.4 Experimento 4: Mover para frente até chegar na pessoa via módulo de controle de voz externo.

6.4.1 Objetivo

Demonstrar o uso do sensor de ultrassom frontal, a fim de impedir que o *RoboDeck* se colida com alguma pessoa ou objeto. Nesse experimento, o robô deve mover-se para frente enquanto a distância do objeto ou pessoa for maior que 50 centímetros e parar, quando for menor.

Figura 47 – Foto do *RoboDeck* parando próximo da pessoa

6.4.2 Ambiente físico

Mesmo local, com idênticas condições ao experimento 1.

6.4.3 Recursos tecnológicos utilizados

Novamente, os quatro recursos já descritos no experimento1: um notebook, um celular, um roteador Sem fio e um *RoboDeck*.

6.4.4 Comandos robóticos utilizados

Três comandos robóticos do SKD Giga de Testes, sendo dois de movimentação, já descritos no experimento 1: ***"robodeck.brake"*** e ***"robodeck.move"***, e um chamado **"robodeck.getUltrasonicSensors",** utilizado para criar o método responsável pela leitura do sensor de ultrassom frontal e armazenamento em uma variável.

6.4.5 Implementação

Cinco métodos foram aplicados: **"moverf1()"** e **"parar1()"** descritos no experimento 1, ***"lersensorf()"*** e ***"lersenfron()"*** tiveram de ser criados para executar a leitura no sensor de ultrassom frontal do *RoboDeck* e armazenar o valor obtido em uma variável chamada **"sensorf".**

A Figura 48 mostra o código-fonte dos métodos ***"lersensorf()"*** e ***"lersenfron()"***:

```
// ler sensor frontal
float sensorf;

public void lersensorf()
{
    report(robodeck.getUltrasonicSensors().readFrontDistance(
            new GigaCallback(this, lersenfron)));
}

public void lersenfron()
{
    sensorf = 0;
    sensorf = robodeck.getUltrasonicSensors().getFrontDistance();
    statusStripLabel.Text = "Comando executado.";

}
```

Figura 48 – Código-fonte dos métodos "lersensorf()" e "lersenfron()"

E o quinto método **“venha()”,** criado através da utilização dos métodos **“lersensorf()”, “moverf1()” e “parar1()”.** Sua função é executar a leitura do sensor frontal, armazenando-a na variável **“sensorf”**, sendo que essa variável representa a distância que o robô está do objeto ou da pessoa.

Dessa maneira, enquanto o valor da variável **“sensorf”** for maior que 50 centímetros, executa-se o método **“moverf1()”,** movendo-se o robô para frente, e quando a distância do objeto ou pessoa for menor, executa-se o método **“parar1()”** (vide apêndice para maiores detalhes).

A Figura 49 apresenta o código-fonte do método **“venha()”**:

```
public void venha()
{
            lersensorf();//executar leitura do sensor frontal
            while (sensorf > 50) // enquanto a distancia for maior que 50 centimetros
            {
                moverf1();  //mova para frente
                System.Threading.Thread.Sleep(100);//para a execução em 100 milisegundos
                lersensorf();//executar leitura do sensor frontal
            }
            parar1(); // para o Robodeck
        }
```

Figura 49 – Código-fonte do método “venha()”

É executado, se a linha “comando” do banco de dados “falaBD” for igual à “venha”, de acordo com a Figura 50:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

        if (linha.Comando.Equals("venha"))
        {
            venha();
        }
```

Figura 50 – Código-fonte da condição para executar o método “venha()”

6.4.6 Sentença reconhecida pelo *Voice Search*

Os critérios para obter as amostras de dados nesse experimento, foram os mesmos já descritos no Experimento1.

O Gráfico 21 expõe as amostras de dados da sentença de fala “venha”, num ambiente sem ruído sonoro.

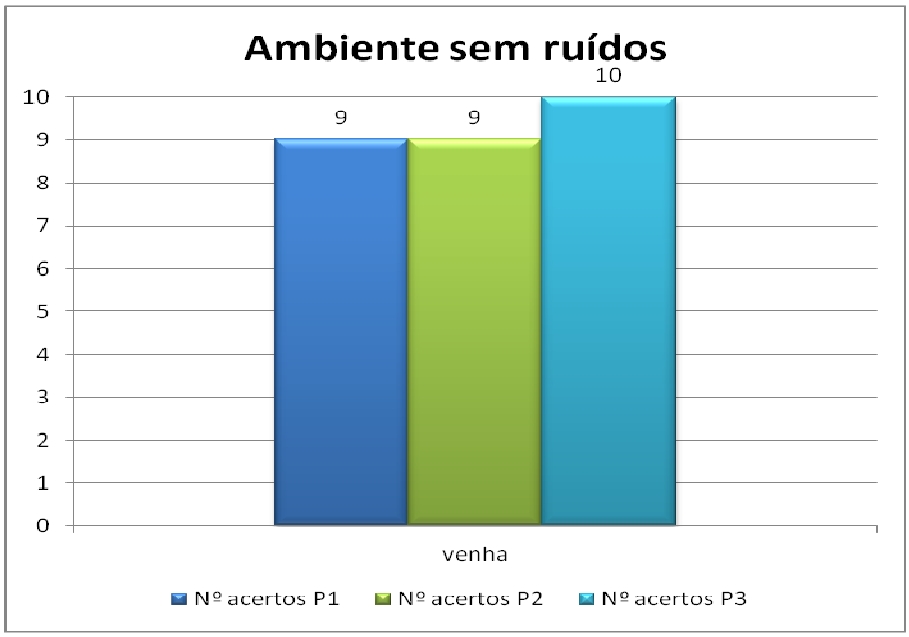


Gráfico 21 – Amostras de dados da sentença que faz o robô se movimentar até a pessoa, via *Voice Search,* em um ambiente sem ruídos sonoros.

A sentença “venha” conseguiu uma média de acertos de 93%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

Seguem no Gráfico 22, as amostras de dados da sentença de fala “venha”, num ambiente com ruído sonoro.

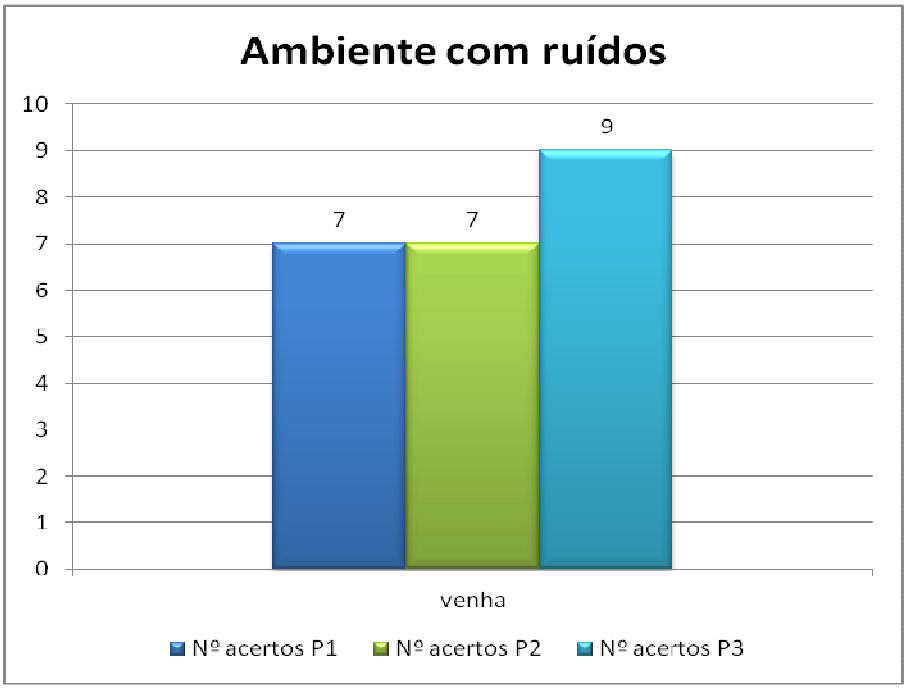


Gráfico 22– Amostras de dados da sentença que faz o robô se movimentar até a pessoa, via *Voice Search,* em um ambiente com ruídos sonoros.

A sentença “venha” adquiriu uma média de acertos de 76%, no ambiente com ruído sonoro.

6.4.7 Dificuldades Encontradas e Análise dos Resultados

Constataram-se duas dificuldades: manter o *RoboDeck* em linha reta, no comando para frente, pois conforme ele se movimentava, ocorria um pequeno deslocamento à esquerda, sendo notado que, após vários testes e análise dos resultados, o problema estava no alinhamento das rodas do robô; definir a distância correta para que o robô não se colidisse com objeto ou pessoa, pois foi necessário levar em consideração o tempo que o *RoboDeck* precisava para responder aos comandos. Inicialmente, foram definidos 50 centímetros de distância, mas, depois de vários testes e análise dos resultados, verificou-se que a distância real ficava em torno de 30 centímetros, devido ao tempo de resposta do robô.

6.5 Experimento 5: Desviar de um objeto via módulo de controle de voz externo.

6.5.1 Objetivo

Demonstrar a movimentação do *Robodeck* em linha reta, desviando de um objeto quando necessário e voltando para seu curso, também em linha reta. Assim, o robô deve mover-se para frente, enquanto a distância do objeto for maior que 50 centímetros e quando menor, parar, girar no próprio eixo 90 graus à direita, parar, voltar a se movimentar para frente por um determinado tempo, parar, girar no próprio eixo 90 graus à esquerda, parar, voltar a se movimentar para frente por um mesmo tempo, parar, girar novamente no próprio eixo 90 graus à esquerda, parar, voltar a se movimentar para frente por um mesmo tempo, parar, girar novamente no próprio eixo 90 graus à direita, parar e voltar a se movimentar para frente.

Figura 51 – Foto do *RoboDeck* desviando do objeto

6.5.2 Ambiente físico

Realizou-se, esse experimento, no pátio da Escola de Engenharia Civil da Politécnica de São Paulo, onde o piso possuía algumas irregularidades e o ruído sonoro era elevado, porque havia vários alunos nesse local.

Figura 52 – Foto do Robodeck no Pátio da Escola Politécnica

6.5.3 Recursos tecnológicos utilizados

Os quatro recursos descritos no experimento1: um notebook, um celular, um roteador Sem fio e um *RoboDeck*.

6.5.4 Comandos robóticos utilizados

Foram utilizados quatro comandos robóticos do SKD Giga de Testes, descritos nos experimentos 1 e 4: ***"robodeck.brake", "robodeck.move", "robodeck.roll"*** e **"robodeck.getUltrasonicSensors"**.

6.5.5 Implementação

Sete métodos participaram, sendo seis descritos nos experimentos 1 e 4, **"moverf1()", "parar1()", "girard1()", "girare1()",** ***"lersensorf()", "lersenfron()"*** e o sétimo método **"obj()",** com a função de executar a leitura do sensor frontal, armazenando-a na variável **"sensorf"**, a qual representa a distância entre o robô e o objeto. Desse modo, enquanto o valor da variável **"sensorf"** for maior que 50 centímetros, executa-se o método **"moverf1()",** movendo-se o robô para frente, mas quando a distância do objeto ou pessoa for menor, emprega-se o método

“parar1()”. Após o robô parar, é executada uma sequencia de métodos para desviar do objeto e continuar movendo-se para frente (vide apêndice para maiores detalhes).

Na Figura 53 encontra-se o código-fonte do método **“obj()”**:

```
public void obj()
{
    lersensorf();//executar leitura do sensor frontal
       while (sensorf > 50)
    {
        moverf1(); //mover para frente
        System.Threading.Thread.Sleep(100);//para a execução em 100 milisegundos
        lersensorf();//executar leitura do sensor frontal
    }
    parar1(); // para
    girard1();// girar a direita
    System.Threading.Thread.Sleep(1550);//para a execução em 1550 milisegundos
    parar1();// parar o robo
    moverf1();  //mover para frente
    System.Threading.Thread.Sleep(3000);//para a execução em 3000 milisegundos
    parar1();// parar o robo
    girare1();// girar a esquerda
    System.Threading.Thread.Sleep(1700);//para a execução em 1700 milisegundos
    parar1();// parar o robo
    moverf1();  //mover para frente
    System.Threading.Thread.Sleep(5000);//para a execução em 5000 milisegundos
    parar1();// parar o robo
    girare1();// girar a esquerda
    System.Threading.Thread.Sleep(1900);//para a execução em 1900 milisegundos
    parar1();// parar o robo
    moverf1();  //mover para frente
    System.Threading.Thread.Sleep(3000);//para a execução em 3000 milisegundos
    parar1();// parar o robo
    girard1();// girar a direita
    System.Threading.Thread.Sleep(100);//para a execução em 100 milisegundos
    parar1();// parar o robo
    moverf1();  //mover para frente
    System.Threading.Thread.Sleep(10000);//para a execução em 10000 milisegundos
    parar1();// parar o robo
}
```

Figura 53 – Código-fonte do método “obj()”

É executado, se a linha “comando” do banco de dados “falaBD” for igual a “objeto”, conforme a Figura 54:

```
private void timer1_Tick(object sender, EventArgs e)
{
  try
    {

        dsBD.comandosRow linha = null;
        dsBD.comandosDataTable tb = null;

        tb = comandosTableAdapter1.GetDataByStatusVazio();

        for (int i = 0; i < tb.Count; i++)
        {
            linha = tb[i];
            listBox1.Items.Add("Comando '" + linha.Comando + "' executado as " + linha.DataHora);
            comandosTableAdapter1.ExecutaComando(linha.CodComando);

    if (linha.Comando.Equals("objeto"))
    {
        obj();
    }
```

Figura 54 – Código-fonte da condição para executar o método "obj()"

6.5.6 Sentença reconhecida pelo *Voice Search*

Os critérios, para obter as amostras de dados nesse experimento, foram os mesmos já descritos no Experimento1.

No Gráfico 23, visualizam-se as amostras de dados da sentença de fala "objeto", num ambiente sem ruído sonoro.

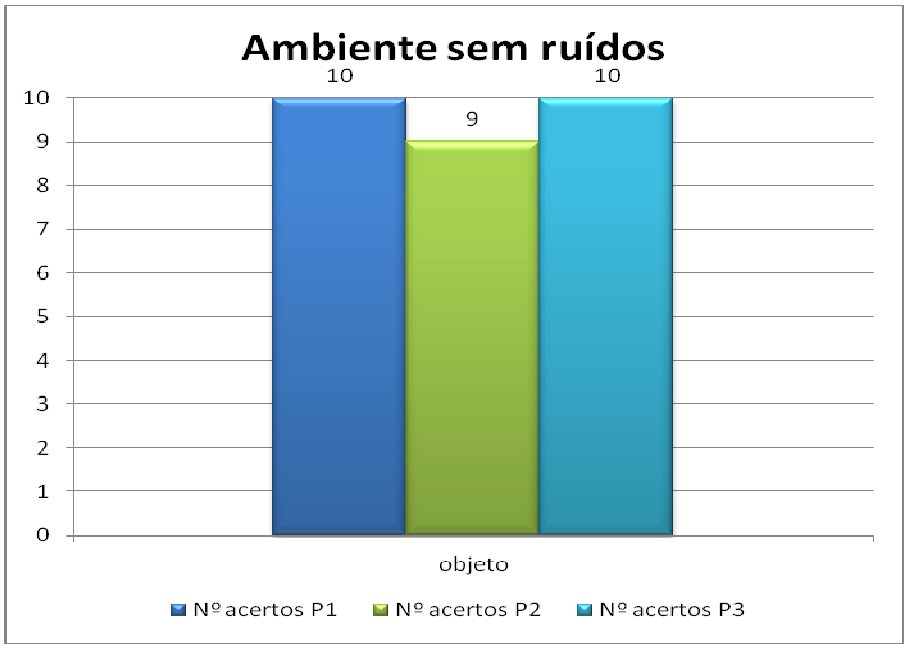


Gráfico 23 – Amostras de dados da sentença que faz o robô desviar de um objeto, via *Voice Search,* em um ambiente sem ruídos sonoros.

A sentença "objeto" atingiu uma média de acertos de 96%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

O Gráfico 24 traz as amostras de dados da sentença de fala “objeto”, num ambiente com ruído sonoro.

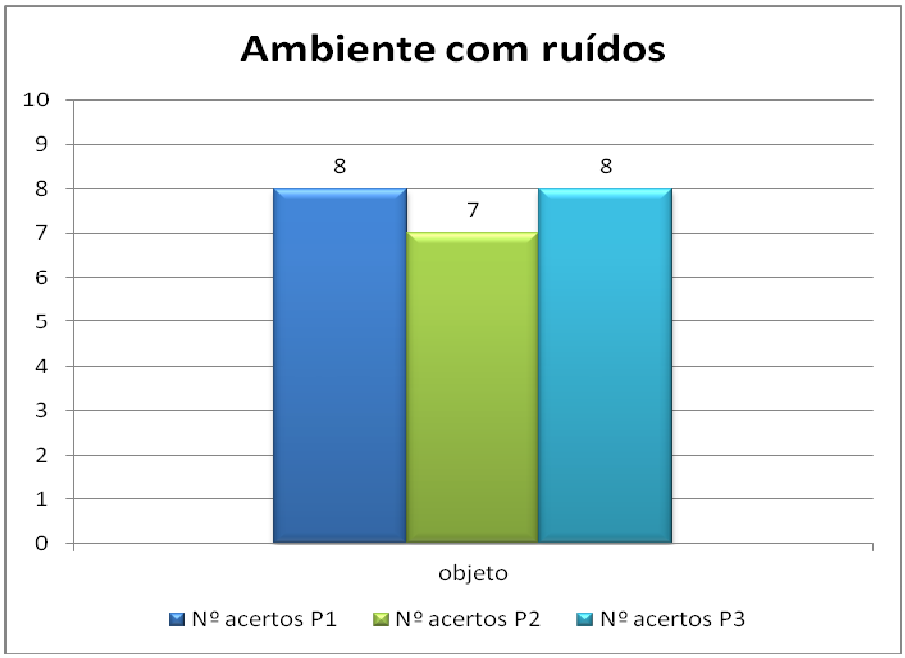


Gráfico 24 – Amostras de dados da sentença que faz o robô desviar de um objeto, via *Voice Search,* em um ambiente com ruídos sonoros.

A sentença “objeto” alcançou uma média de acertos de 76%, no ambiente com ruído sonoro.

6.5.7 Dificuldades Encontradas e Análise dos Resultados

Três dificuldades ocorreram: manter o *RoboDeck* em linha reta, no comando para frente, pois conforme ele se movimentava, ocorria um pequeno deslocamento à esquerda, observando-se que, após realizar vários testes e analisar os resultados, o problema estava no alinhamento das rodas do robô; definir a distância correta para que o robô não se colidisse com o objeto, levando-se em consideração o tempo que o *RoboDeck* precisava para responder aos comandos, estabelecendo-se, em princípio, 50 centímetros de distância, mas, deixando-o, depois dos testes e análises, em torno de 30 centímetros, devido ao tempo de resposta do robô; fazer o *RoboDeck* desviar do objeto e continuar se movimentando para frente em linha reta, buscando determinar o tempo exato de execução em todos os métodos utilizados, pois o robô tem uma pequena variação na intensidade da potência e na velocidade de realização de cada método, influenciando seus ângulos.

6.6 Experimento 6: Movimentações básicas via módulo de controle de voz interno.

6.6.1 Objetivo

Demonstrar as movimentações básicas do *RoboDeck* via reconhecimento de fala, por meio de um microfone conectado no robô. Nesse experimento demonstraram-se as seguintes movimentações, todas com velocidades pré-definidas:

- ✓ Para frente;
- ✓ Virar à esquerda;
- ✓ Virar à direita;
- ✓ Parar.

6.6.2 Ambiente físico

Para realizar esse experimento havia a necessidade de instalar o *CvoiceControl* e seus pacotes dependentes, no Sistema Operacional Debian do RoboDeck, fato que alteraria as configurações de um sistema já em uso. Como se tratava de um robô cedido pela Escola Politécnica da USP, os testes no módulo de voz interno ocorreram de forma simulada, numa sala fechada, em um computador que possuía o *CVoiceControl* e o MAP (Módulo de Alta Performance) do *RoboDeck*.

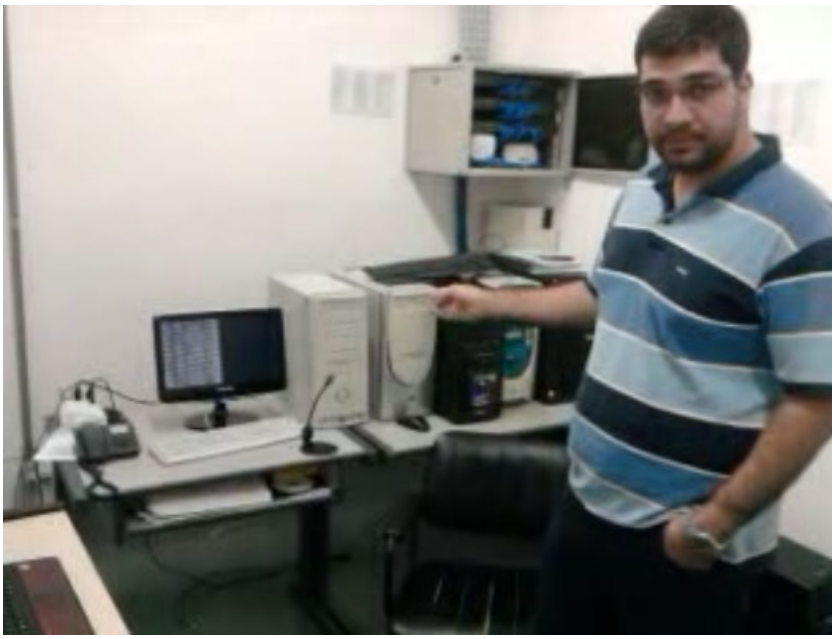


Figura 55 – Foto da sala e da maquina que foi simulado este experimento

6.6.3 Recursos tecnológicos utilizados

Usou-se um microcomputador para simular o funcionamento do *RoboDeck*, com o Sistema Operacional Debian, onde foram instalados e configurados o sistema de reconhecimento de fala *CvoiceControl* e o MAP(Módulo de Alta Performance) do robô.

6.6.4 Comandos robóticos utilizados

Foram utilizadas duas Bibliotecas desenvolvidas por Orlandini (2012): a "***RoboDeckInfo.hh***", cujas funções são estabelecer a conexão e iniciar uma sessão com o MAP (Modulo de Alta Performance) do *RoboDeck*, para envio de comandos; e a ***"RoboDeckMoving.hh",*** com a finalidade de enviar os comandos responsáveis pela movimentação do robô ao MAP.

6.6.5 Implementação

Desenvolveram-se quatro controladores internos, chamados através do *CvoiceControl*, estando cada um associado à uma referência de fala.

O primeiro controlador denominou-se **"Ande",** responsável pela movimentação do robô para frente e associado à referência de fala **"Mova"**. A Figura 56 mostra o código-fonte de sua implementação:

```
#include <iostream>
#include <stdio.h>
#include "RoboDeckMoving.hh"
#include "RoboDeckInfo.hh"
using namespace std;

RoboDeckMoving roboDeckMoving;
RoboDeckInfo roboDeckInfo;
RoboDeck robo;

int main(int argc, char *argv[])
{
  roboDeckInfo.openConnection("localhost","2000", robo);
  roboDeckInfo.openSession(robo);

      bool moveuRobo = roboDeckMoving.moveRobot(32000,0,robo);

      cout <<"Robot Comunication Protocol Version: "
<<roboDeckInfo.getRobotProtocolVersion(robo) << endl;
           roboDeckInfo.closeSession(robo);
           return 0;
}
```

Figura 56 – Código-fonte do controlador "Ande"

Observando-se a Figura, nota-se a utilização do comando "***roboDeckInfo.openConnection",*** para estabelecer a conexão com MAP; do comando ***"roboDeckInfo.openSession",*** para iniciar a sessão com o mesmo; e do comando ***"roboDeckMoving.moveRobot",*** para mover o robô para frente, cuja intensidade foi configurada em **32000**, movendo-o com potência máxima.

O segundo controlador foi **"Parar"**, responsável pela parada da movimentação do robô, associado à referência de fala **"Pare"**. A Figura 57 exibe o código-fonte de sua implementação:

```
#include <iostream>
#include <stdio.h>
#include "RoboDeckMoving.hh"
#include "RoboDeckInfo.hh"

using namespace std;

RoboDeckMoving roboDeckMoving;
RoboDeckInfo roboDeckInfo;
RoboDeck robo;

int main(int argc, char *argv[])
{
  roboDeckInfo.openConnection("localhost","2000", robo);
  roboDeckInfo.openSession(robo);

  bool moveuRobo = roboDeckMoving.moveRobot(0,0,robo);

cout <<"Robot Comunication Protocol Version: "<<
roboDeckInfo.getRobotProtocolVersion(robo) << endl;
          roboDeckInfo.closeSession(robo);
  return 0;
}
```

Figura 57 – Código-fonte do controlador "Parar"

Baseado nessa Figura, constata-se o emprego do comando "***roboDeckInfo.openConnection",*** para estabelecer a conexão com MAP; do comando ***"roboDeckInfo.openSession",*** para iniciar a sessão com o mesmo; e do comando ***"roboDeckMoving.moveRobot",*** com intensidade zero, objetivando parar o robô.

O terceiro controlador chamou-se **"Esquerda",** responsável por virar o robô a esse lado, foi associado à referência de fala **"Esquerda".** Na Figura 58, visualiza-se o código-fonte de sua implementação:

```
#include <iostream>
#include <stdio.h>
#include "RoboDeckMoving.hh"
#include "RoboDeckInfo.hh"

using namespace std;

RoboDeckMoving roboDeckMoving;
RoboDeckInfo roboDeckInfo;
RoboDeck robo;

int main(int argc, char *argv[])
{
 roboDeckInfo.openConnection("localhost","2000", robo);
 roboDeckInfo.openSession(robo);

 bool virouRobo = roboDeckMoving.turnRobot(-32,15000,15000,robo);

cout <<"Robot Comunication Protocol Version: "<<
roboDeckInfo.getRobotProtocolVersion(robo) << endl;
          roboDeckInfo.closeSession(robo);
 return 0;
}
```

Figura 58 – Código-fonte do controlador "Esquerda"

A Figura mostra a utilização do comando "***roboDeckInfo.openConnection",*** para estabelecer a conexão com MAP; do comando ***"roboDeckInfo.openSession",*** para iniciar a sessão com o mesmo; e do comando **"roboDeckMoving.turnRobot",** que tem como parâmetros o ângulo, a intensidade do motor da direita e a intensidade do motor da esquerda. As intensidades foram configuradas em 15000 e o ângulo em -32, sendo que os ângulos suportados no comando robótico **"roboDeckMoving.turnRobot"** variam de -32 até 32, onde os ângulos negativos referem-se à esquerda e os positivos, à direita. Assim, tal configuração move o robô para frente com metade da sua potência, virando à esquerda com o ângulo máximo suportado.

O quarto e último controlador, **"Direita",** responsável por virar o robô à direita, associou-se à referência de fala **"Direita".** A Figura 59 expõe o código-fonte de sua implementação:

```
#include <iostream>
#include <stdio.h>
#include "RoboDeckMoving.hh"
#include "RoboDeckInfo.hh"

using namespace std;

RoboDeckMoving roboDeckMoving;
RoboDeckInfo roboDeckInfo;
RoboDeck robo;

int main(int argc, char *argv[])
{
 roboDeckInfo.openConnection("localhost","2000", robo);
 roboDeckInfo.openSession(robo);

 bool virouRobo = roboDeckMoving.turnRobot( 32,15000,15000,robo);

cout <<"Robot Comunication Protocol Version: "<<
roboDeckInfo.getRobotProtocolVersion(robo) << endl;
           roboDeckInfo.closeSession(robo);
 return 0;
}
```

Figura 59 – Código-fonte do controlador "Direita"

Observando essa Figura, verifica-se o uso do comando "***roboDeckInfo.openConnection",*** para estabelecer a conexão com MAP; do comando ***"roboDeckInfo.openSession",*** para iniciar a sessão com o mesmo; e do comando **"roboDeckMoving.turnRobot",** que possui como parâmetros o ângulo, a intensidade do motor da direita e a intensidade do motor da esquerda. As intensidades foram configuradas em 15000 e o ângulo em 32, sendo que os ângulos suportados no comando robótico **"roboDeckMoving.turnRobot"** variam de -32 até 32, com os ângulos negativos referindo-se à esquerda e os positivos, à direita. Essa configuração move o robô para frente com metade da sua potência, virando à direita com o ângulo máximo suportado.

6.6.6 Sentenças reconhecidas pelo *CvoiceControl*

Para obter uma amostragem de dados, utilizaram-se três pessoas, sendo dois homens "P1" e "P2" e uma mulher "P3". Cada pessoa tinha de repetir vinte vezes uma mesma sentença para a movimentação do robô, a qual foi dividida em dois ambientes sonoros: dez sentenças em um ambiente sem ruído sonoro e as outras dez, com ruídos. Assim, fez-se uma amostra de dados com os números de sentenças reconhecidas e não reconhecidas pelo *CvoiceControl*.

O Gráfico 25 apresenta as amostras de dados da sentença de fala "mova", num ambiente sem ruído sonoro.

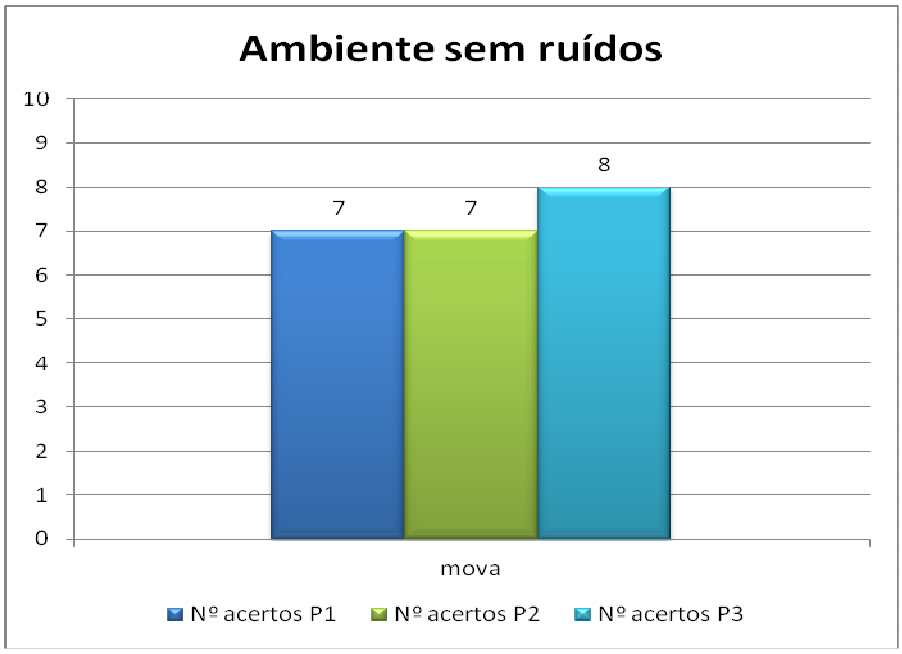


Gráfico 25 – Amostras de dados da sentença que faz o robô se movimentar para frente, via *CvoiceControl,* em um ambiente sem ruídos sonoros.

A sentença “mova” obteve uma média de acertos de 73%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

No Gráfico 26, encontram-se as amostras de dados da sentença de fala “mova”, num ambiente com ruído sonoro.

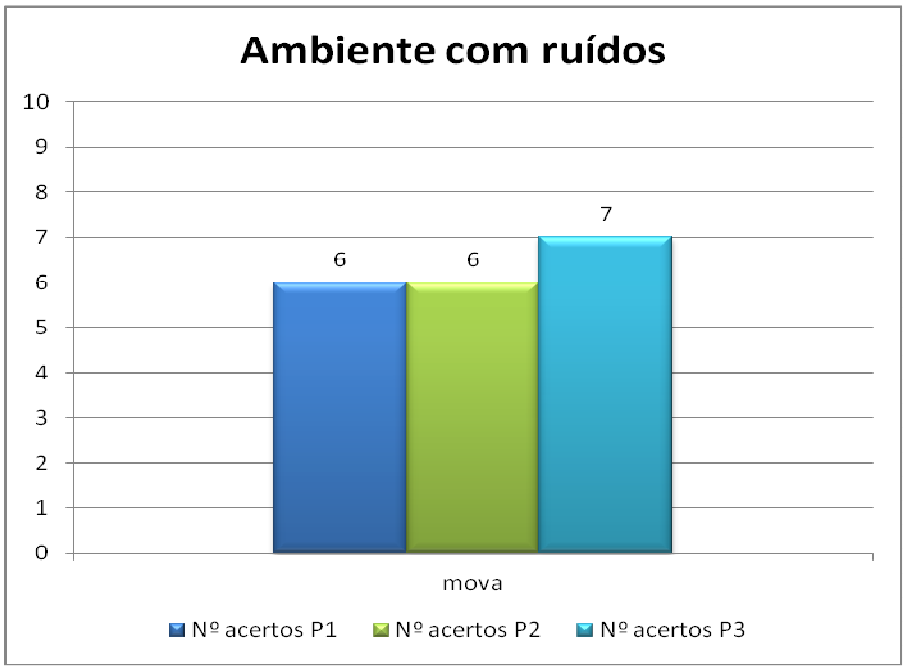


Gráfico 26 – Amostras de dados da sentença que faz o robô se movimentar para frente, via *CvoiceControl*, em um ambiente com ruídos sonoros.

A sentença “mova” atingiu uma média de acertos de 63%, no ambiente com ruído sonoro.

O Gráfico 27 exibe as amostras de dados das sentenças de fala “para”, num ambiente sem ruído sonoro.

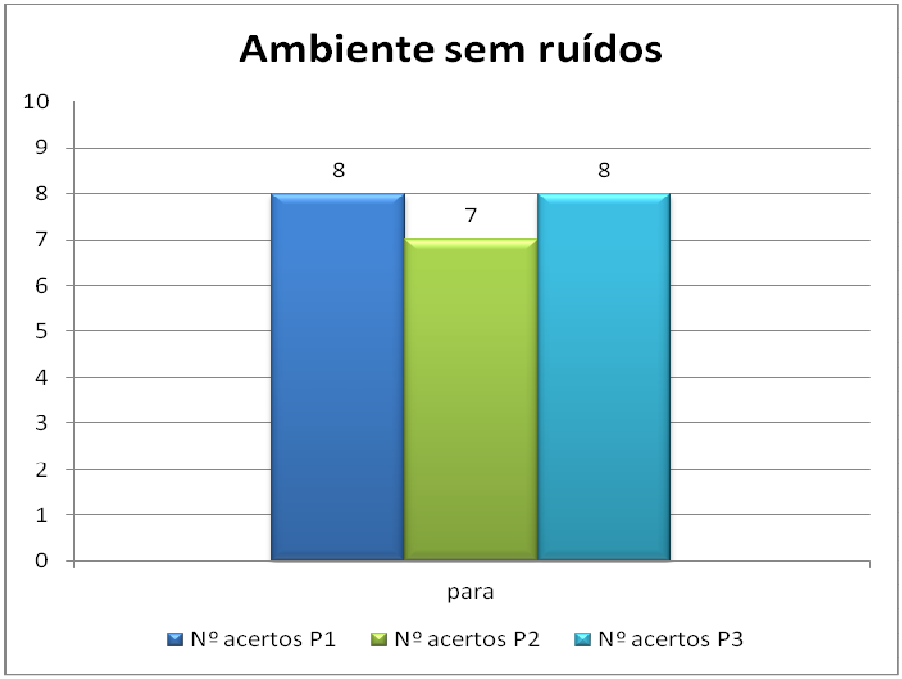


Gráfico 27 – Amostras de dados da sentença que faz o robô parar, via *CvoiceControl*, em um ambiente sem ruídos sonoros.

A sentença “para” teve uma média de acertos de 76%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

O Gráfico 28 ilustra as amostras de dados da sentença de fala “para”, num ambiente com ruído sonoro.

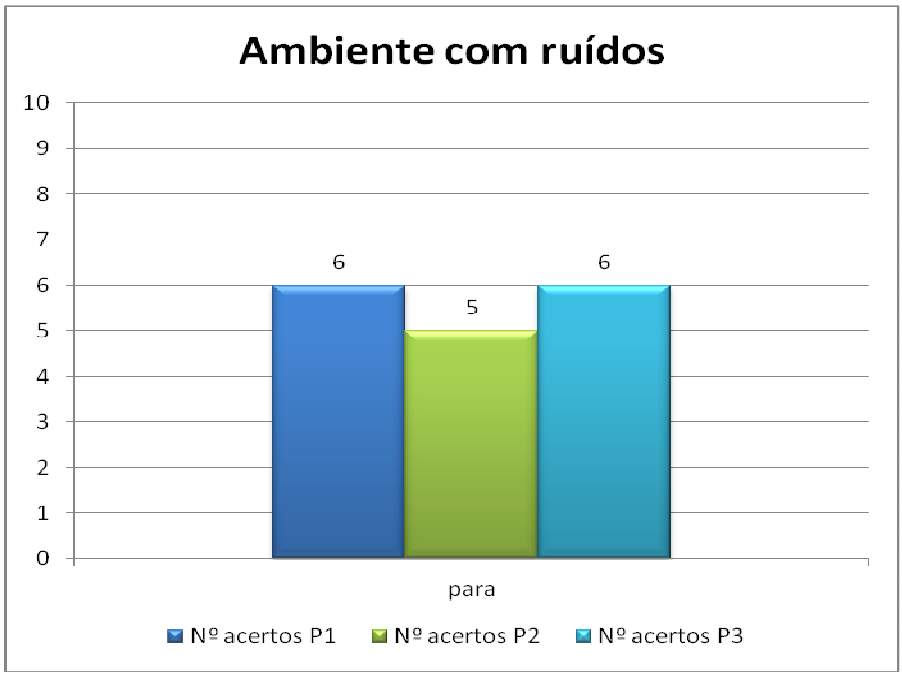


Gráfico 28 – Amostras de dados da sentença que faz o robô se movimentar para frente, via *CvoiceControl*, em um ambiente com ruídos sonoros.

A sentença “para” conseguiu uma média de acertos de 56%, no ambiente com ruído sonoro.

No Gráfico 29, verificam-se as amostras de dados das sentenças de fala “esquerda”, num ambiente sem ruído sonoro.

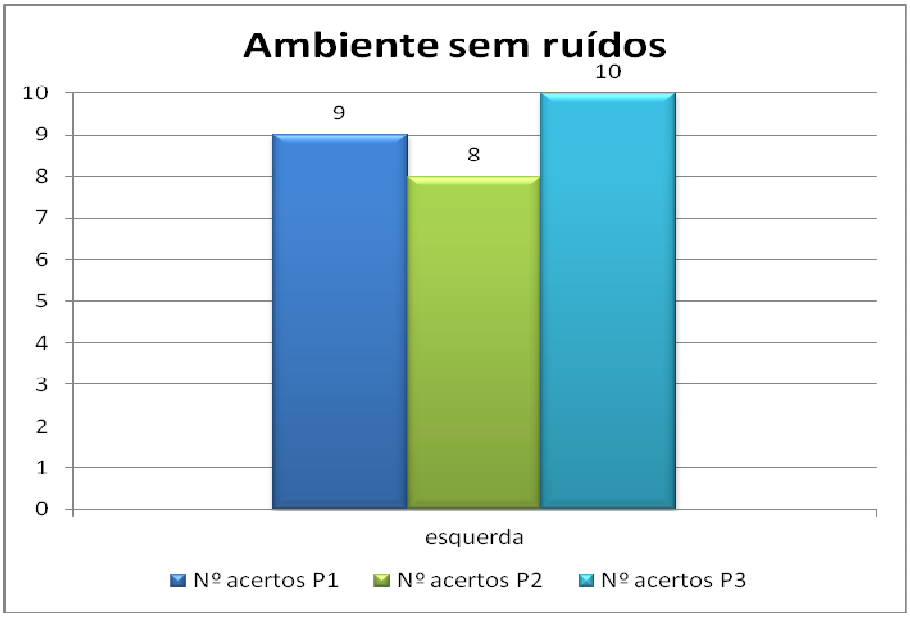


Gráfico 29 – Amostras de dados da sentença que faz o robô virar à esquerda, via *CvoiceControl*, em um ambiente sem ruídos sonoros.

A sentença “esquerda” mostrou uma média de acertos de 90%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

O Gráfico 30 expõe as amostras de dados da sentença de fala “esquerda”, num ambiente com ruído sonoro.

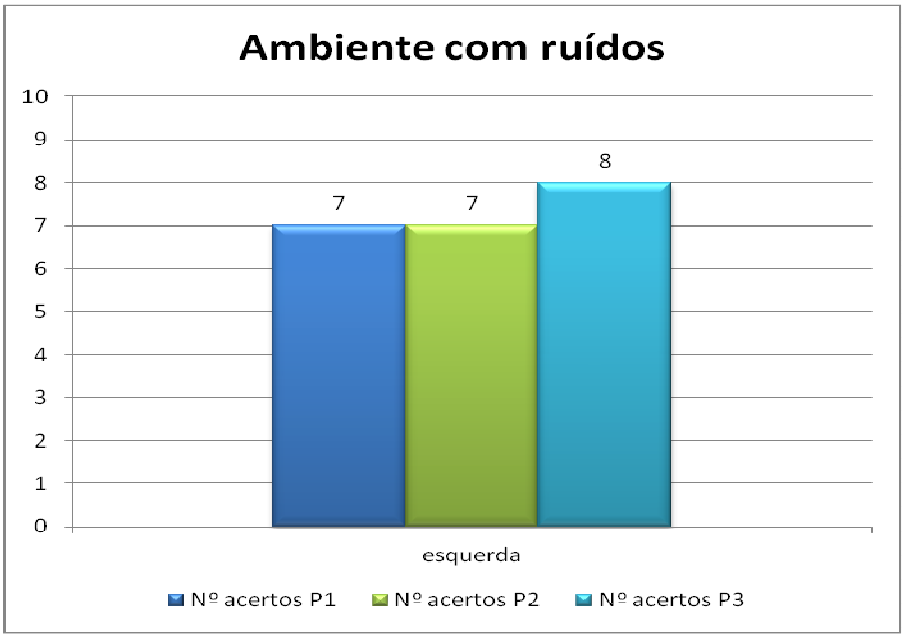


Gráfico 30 – Amostras de dados da sentença que faz o robô virar à esquerda, via *CvoiceControl*, em um ambiente com ruídos sonoros.

A sentença “esquerda” atingiu uma média de acertos de 73%, no ambiente com ruído sonoro.

No Gráfico 31, estão as amostras de dados das sentenças de fala “direita”, num ambiente sem ruído sonoro.

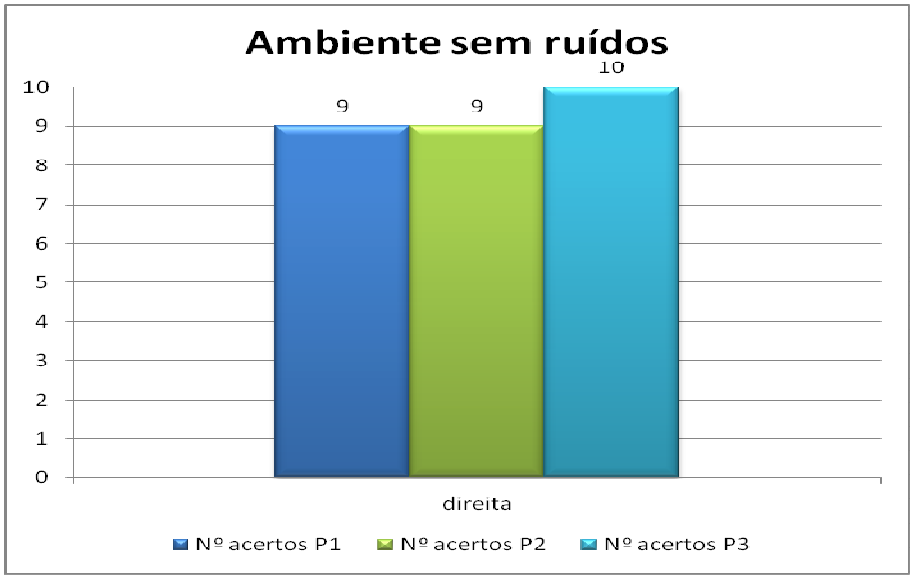


Gráfico 31 – Amostras de dados da sentença que faz o robô virar à direita, via *CvoiceControl*, em um ambiente sem ruídos sonoros.

A sentença “direita” apresentou uma média de acertos de 93%, no reconhecimento de fala em um ambiente sem ruídos sonoros.

Seguem no Gráfico 32, as amostras de dados da sentença de fala “direita”, num ambiente com ruído sonoro.

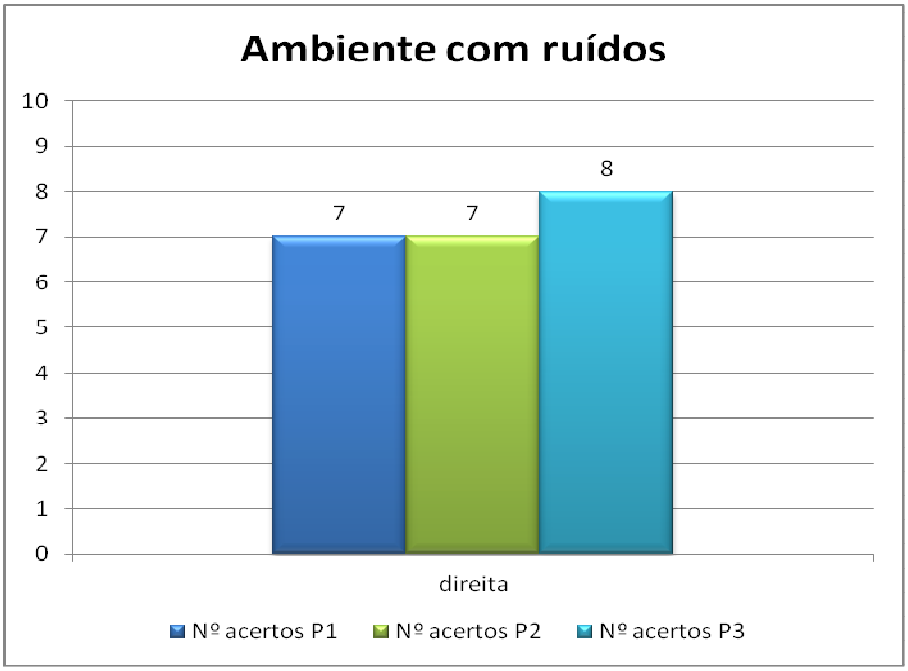


Gráfico 32 – Amostras de dados da sentença que faz o robô virar à direita, via *CvoiceControl*, em um ambiente com ruídos sonoros.

A sentença “direita” obteve uma média de acertos de 73%, no ambiente com ruído sonoro.

6.6.7 Dificuldades Encontradas e Análise dos Resultados

A dificuldade constituiu-se em visualizar o comportamento real do *Robodeck* para cada comando de movimentação, pois o experimento foi realizado numa máquina que simulava o robô, obtendo-se apenas os endereços enviados e recebidos por todos os comandos executados.

7. CONCLUSÃO

No momento, existem no Brasil poucos trabalhos de pesquisa envolvendo reconhecimento de fala aplicada na robótica móvel, contribuindo a presente dissertação para aumentar as linhas de pesquisa afins.

Os módulos de reconhecimento de voz desenvolvidos permitiram executar comandos para movimentar o *Robodeck* através da fala, oferecendo um potencial enorme à área que está em grande expansão, mediante a criação de robôs com várias finalidades na indústria, em operações militares, na vigilância, no entretenimento e até nos trabalhos de limpeza.

Com uma interface de reconhecimento de fala, o homem pode comandar o robô, usando sua própria voz, a qual é transformada em comandos reconhecidos por ele, proporcionando melhor acessibilidade e permitindo que pessoas com deficiência física, por exemplo, controlem um robô móvel, através de comandos simples de fala.

Com base na plataforma robótica móvel *RoboDeck* e por se tratar de um robô fabricado no Brasil, era essencial que o mesmo reconhecesse comandos em Português: o obstáculo é a escassez dos pacotes de reconhecimento de fala em Língua Portuguesa.

Realizou-se, portanto, um estudo entre os diversos sistemas de reconhecimento de fala e dois se destacaram: o *CvoiceControl* e o *Voice Search*.

O *CvoiceControl* foi usado no desenvolvimento do módulo de controle por voz interno, por ser compatível com o sistema operacional Linux, utilizado pelo *RoboDeck*. Porém, surgiram algumas limitações como a necessidade de treinar o locutor para o seu uso, o que impedia a utilização por qualquer pessoa, sem antes capturar as referências de voz e associá-las aos comandos; o acionamento dos comandos de fala ocorrer somente por um microfone conectado ao *RoboDeck*, fixando a distância entre a pessoa e o robô.

Diante dessa realidade, observou-se a necessidade de criar um segundo módulo de controle por voz, só que externo, visando a possibilitar o acionamento dos comandos por voz a qualquer distância do robô, através de dispositivo móvel como um celular, desde que tivesse acesso à Internet e o *Voice Search* instalado. Com o emprego desse sistema, resolveu-se a questão de treinamento dos locutores, porque o *Voice Search* dispensa-o e reconhece mais de 28 línguas diferentes, inclusive o Português.

A única restrição do módulo de controle de voz externo é a indispensável utilização de um computador ou um notebook, como servidor: o dispositivo móvel “Celular” tem de se conectar à página de reconhecimento de Fala, hospedada em um computador, que encaminha os comandos de movimentação para o *RoboDeck*, via rede sem fio, utilizando o SDK “Giga de Testes”.

Nesse contexto, o módulo de controle por voz externo resolveu os problemas da necessidade de treinar um locutor e da limitação de distância entre a pessoa e o robô, tornando o processo de reconhecimento de fala prático e interativo, mas exigiu a presença de um computador externo para funcionar. E o módulo de controle por voz interno que não necessita de dispositivo externo, depende de pré-treinamento toda vez que uma nova pessoa for utilizar o robô, transformando esse módulo pouco usual.

Como trabalho futuro fica o desafio de integrar o módulo de controle por voz externo dentro do *RoboDeck*, dispensando o uso de computador adicional. Para isso será necessário instalar, na placa de alta performance do robô, um servidor de Internet e um servidor de banco de dados, além de migrar todo o SDK “Giga de Testes” para o Sistema Operacional Linux (Debian), produzido no Visual Studio 2010 em C#, produto da Microsoft.

REFERÊNCIAS BIBLIOGRÁFICAS

ADEPT, MobilerRobots. ***Specification Manual Pionner 3 - AT.*** 2011. Disponível em: < http://www.mobilerobots.com/Libraries/Downloads/Pioneer3AT-P3AT-RevA.sflb.ashx >. Acesso em: 14 out. 2011.

BEKET, G A. ***Autonomous Robots: From Biological Inspiration to Implementation and Control.*** *In: Cambridge, USA : The MIT Press, 2005.*

BOERSMA, P**.** ***Praat, a system for doing phonetics by computer.*** In: Glot International. v.5:9/10, p. 341–345, 2001.

CAMPBELL, J P. ***Speaker Recognition.*** In: Proceedings of the IEEE. 85:1437-1462,1997.

CHENGALVARAYAN, R. ***Hierarchical Subband Linear Predictive Cepstral Features for HMM-Basead Speech Recognition.*** In: International Conference on Acoustics, and Signal Processing, 1999.

CALEY, Richard., BLACK, W Alan., TAYLOR, Paul. ***System manual documentation Edition 1,4.*** 1999. Disponível em: < http://www.cstr.ed.ac.uk/projects/festival/manual/ >. Acesso em: 06 out. 2011.

CREEMER, David Z., DORIS, Tom., CRAFT, Brian. ***Xvoice is Copyright (c) on-line documentation.*** 1999. Disponível em: <http://xvoice.sourceforge.net>. Acesso em: 15 out. 2011.

DEVILLIERS, E M. ***Implementing Voice Recognition And Natural Language Processing In The NPSNET Networked Virtual Environment.*** In: Naval Postgraduate school. California, 1996.

DODDINGTON,G R*.* ***Speaker rcognition:*** **Idetifying People by their Voices*.*** In: Proccedings of the IEEE. Vol 73, No11, pages 1651-1664,1985.

FECHINE,M Joseana. ***Reconhecimento Automático de Identidade Vocal utilizando Modelagem Híbrida:*** **Paramétrica e Estatística.** In: Tese de Doutorado, Universidade Federal da Paríba, 2000.

FURUI, S. ***Digital Speech Processing*****, Synthesis and Recognition**. Marcel Dekker, New York,1989.

Geoffrois, Edouard., Boudahmane, Karim. ***TranscriberAG user manual****.* 2011. Disponível em: <http://transag.sourceforge.net/index.php?content=manual>. Acesso em: 16 out. 2011.

HUGO, Marcel. ***Uma Interface de Reconhecimento de Voz para o Sistema de Gerenciamento de Central de Informação de Fretes.*** In: Dissertação de Mestrado, Universidade Federal de Santa Catarina, 1995.

IBM. ***Manual do usuário do IBM ViaVoice 9****. 2008.*

IROBOT. ***iRobot Create Owner's guide.*** 2006. Disponível em: < http://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Manual_Final.pdf >. Acesso em: 14 out. 2011.

KIECZA, Daniel. ***CVoiceControl on-line documentation***. 2002. Disponível em: < http://www.kiecza.net/daniel/linux/cvoicecontrol/index.html />. Acesso em: 02 out. 2011.

LEGO. ***LEGO Mindstorms User Guid.*** 2009. Disponível em: < http://cache.lego.com/upload/contentTemplating/Mindstorms2BuildingInstructions/otherfiles/downloadA0B7A698E231D3E619C43ECCFFE2F27F.pdf >. Acesso em: 15 out. 2011.

MACIEL, A M A. ***Investigação de um Ambiente para o processamento de voz utilizando VoiceXML.*** In: Dissertação de mestrado, Universidade Federal de Pernambuco, Pernambuco, 2007.

MOREIRA, F Luís. ***Reconhecimento Automático de Fala Contínua***. 2006. Disponível em: <http://www.ipb.pt/~fmoreira/Itens/RecFalaCont.pdf>. Acesso em: 09 set. 2011.

MUÑOZ, S Mauro. ***Projeto do software do RoboDeck***. Versão 0.2. São Carlos, 2011.

NUANCE, Communications. ***Dragon NaturllySpeaking Home on-line documentation***. 2011. Disponível em: < http://www.nuance.com/for-individuals/by-product/dragon-for-pc/home-version/index.htm >. Acesso em: 06 out. 2011.

ORLANDINI, Guilherme. ***Desenvolvimento de Aplicativos Baseados em Técnicas de Visão Computacional para Robô Móvel Autônomo*** In: Dissertação de mestrado, Universidade Metodista de Piracicaba, Piracicaba, 2012.

PIZZOLATO, E B. ***Reconhecimento de Fala, Conceitos, Fundamentos e Técnicas***. In: Iniciação Científica, Universidade Federal de São Carlos, São Carlos, 2001.

POZZEBON, R. ***Google Voice Search.*** Disponível em: <http://www.oficinadanet.com.br/artigo/internet/google-voice-search> Acesso em 28 maio 2012

RABINER, L R. ***Applications of Voice Processing to Telecommunications.*** *In:* Proceedings of IEEE. Vol 82, Issue 2, 1994.

XBOT. ***Manual do usuário RoboDeck.*** Versão 1.0. São Carlos, 2011.

ROE, D B., WILPON J G. ***Voice communication between humans and machines.*** In: National Academy of Sciences, 1994.

SIMÕES, O Flavio. ***Implementação de um sistema de conversão Texto-Fala para o português do Brasil.*** In: Dissertação de mestrado, Universidade estadual de Campinas, Campinas, 1999.

TATHAM, M. ***Speech Recognition.*** Disponível em: <http://www.essex.ac.uk/ speech/teaching/erasmus/recognit.html > Acesso em 15 Jul. 2011.

VIEIRA, M N. ***Módulo Frontal para um Sistema de reconhecimento Automático de Voz.*** In: Dissertação de Mestrado, Universidade estadual de Campinas, Campinas,1984.

ZELTER D., JOHNSON, M B. ***Interacting with Virtual Environments.*** In: Ed. John Wiley & Sons, New York, 1994.

APÊNDICE

A seguir, apresentam-se listados os códigos-fontes usados para o desenvolvimento do módulo de controle de voz externo:

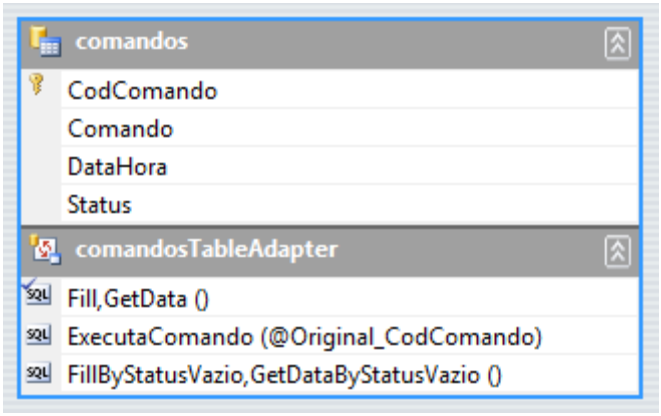
1. app.config

Tem a função de conectar o SDK Giga de Testes ao banco de dados “falabd”.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="GigaDeTestes.Properties.Settings.ConexaoBD"
connectionString="server=localhost;User Id=root;password=1234;Persist Security
Info=True;database=falabd"
            providerName="MySql.Data.MySqlClient" />
    </connectionStrings>
</configuration>
```

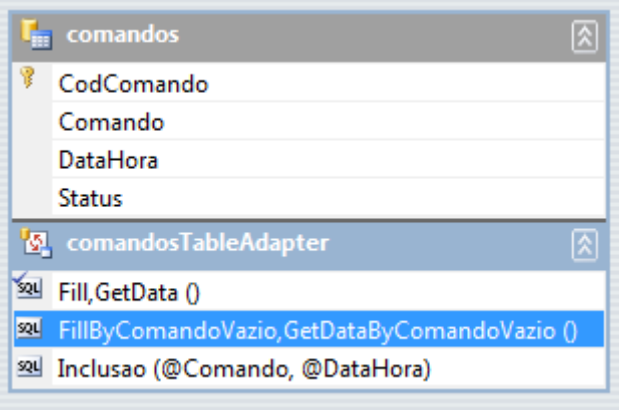
2. dsDB.xsd - *DataSet* Tipado da aplicação “ControladorRobo”

O *data table* “comandos” representa a tabela no banco de dados. O “comandos*TableAdapter*” executa os comandos SQL no banco de dados. O método ExecutaComando executa um *update* no banco dados onde é alterado o status para “executado”. O método *FilByStatus*Vazio executa um *select* para consultar os comandos disponíveis à realização dos movimentos do robô.



3. dsDB.xsd - *DataSet* Tipado da aplicação “WebRobo”

O método inclusão, pega o comando reconhecido pela fala, e armazena no banco de dados juntamente com a data e a hora da execução.



4. index.htm

Código fonte do índex.htm referente à página de reconhecimento de fala contendo o *JavaScript* para verificação a cada 1000ms

```
<HTML>
<HEAD>
<TITLE>Reconhecimento de Fala</TITLE>

    <SCRIPT>        //inicio do script

        //função javascript para verificação de texto no campo
        function verificaCampos(form)
        {
               //variável que recebe a referencia do campo no formulario
            var campo = form.campo;
            //verifica se o campo esta vazio
                 if (campo.value == "")
            {
                //envia o foco (cursor) de volta ao campo
                        campo.focus();
                 }
            else
            {
                //submete o formulário para a pagina "processa.aspx”
                     form.submit();
                 }
            //função recursiva que chama novamente a função verificaCampos a cada
1000ms
               setTimeout("verificaCampos(form)",1000);
        }
    </SCRIPT>

</HEAD>
<BODY onLoad="verificaCampos(form);">
    <FORM id="form" action="processa.aspx">
        <h1>Reconhecimento de Fala</h1>
        Fale o comando desejado<BR><INPUT name=campo>  
    </FORM>
</BODY>
</HTML>
```

5. processa.aspx

Código-fonte do processa.aspx, cuja função é receber o formulário da página de índex.htm e realizar a inclusão no banco de dados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace FalaWeb
{
    public partial class Default2 : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            try
            {
                //declara e instancia um objeto para inclusão do comando no BD
                 FalaWeb.dsBDTableAdapters.comandosTableAdapter tb = new
FalaWeb.dsBDTableAdapters.comandosTableAdapter();

                //declara a variável campo, que recebe o valor digitado da pagina
index.html"
                 string campo = Request["campo"];

                //se o campo NAO for vazio
                 if ( !campo.Equals("") )
                 {
                     //executa o método de inclusão no BD, passando o comando recebido
e a data/hora atual
                    if (tb.Inclusao(campo, DateTime.Now) > 0)
                    {
                        //redireciona a pagina de volta para o index.html
                        Response.Redirect("index.htm");
                    }
                    else
                    {
                        //se não conectar ao banco, exibe a mensagem abaixo
                        lblComando.Text = "Não foi inserido no BD !";
                    }
                }
            }
            catch (Exception ex)
            {
                //em caso de qualquer erro, exibe a mensagem abaixo
                lblComando.Text = "Ocorreu o seguinte erro: " + ex.Message;
            }
        }
    }
}
```

6. MainForm.cs*

Código-fonte de todas as alterações realizadas no formulário principal do SDK Giga de Testes do RoboDeck, para implementar o “Comandos de Fala”.

```
// comando para reconhecimento da fala

// mover para frente

public void moverf1()
{
    short intensity = 12276; // ou é 32760
    report(robodeck.move(
    intensity,
    new GigaCallback(this, actionCallback)));

}

// mover para traz

public void movert1()
{
    short intensity = -12276; // ou é 32760
    report(robodeck.move(
    intensity,
    new GigaCallback(this, actionCallback)));

}

// parar

public void parar1()
{
    report(robodeck.brake(new GigaCallback(this, actionCallback)));

}

// virar a esquerta

public void virare1()
{
    sbyte angle = -32;
    short intensity = 15000; // ou é 32760
    report(robodeck.turn(
    angle,
    intensity,
    new GigaCallback(this, actionCallback)));
}

// virar a direita

public void virard1()
{
    sbyte angle = 32;
    short intensity = 15000; // ou é 32760
    report(robodeck.turn(
    angle,
    intensity,
    new GigaCallback(this, actionCallback)));
}
```

```
        // girar a esquerda
        public void girare1()
        {
            short intensity = 15000; // ou é 32760
            report(robodeck.roll(
            rdsdk.Side.LEFT,
            intensity,
            new GigaCallback(this, actionCallback)));
        }

        // girar a direita
        public void girard1()
        {
            short intensity = 15000; // ou é 3276
            report(robodeck.roll(
            rdsdk.Side.RIGHT,
            intensity,
            new GigaCallback(this, actionCallback)));
        }


        // ler sensor frontal
      float sensorf;


        public void lersensorf()
        {
            report(robodeck.getUltrasonicSensors().readFrontDistance(
                    new GigaCallback(this, lersenfron)));
        }

        public void lersenfron()
        {
            sensorf = 0;
            sensorf = robodeck.getUltrasonicSensors().getFrontDistance();
            statusStripLabel.Text = "Comando executado.";

        }


        //método venha

        public void venha()
        {
                        lersensorf();//executar leitura do sensor frontal
                        while (sensorf > 50)
                        {
                            moverf1();  //mova
                            System.Threading.Thread.Sleep(100);//para a execução em
100 milisegundos
                            lersensorf();//executar leitura do sensor frontal
                        }
                        parar1(); // para
                    }


           // método objeto

        public void obj()
        {
```

```
            lersensorf();//executar leitura do sensor frontal
            while (sensorf > 50)
            {
                moverf1();  //mova
                System.Threading.Thread.Sleep(100);//para a execução em 1000
milisegundos
                lersensorf();//executar leitura do sensor frontal
            }
            parar1(); // para
            girard1();
            System.Threading.Thread.Sleep(1550);//para a execução em 1550 milisegundos
            parar1();
            moverf1();  //mova
            System.Threading.Thread.Sleep(3000);//para a execução em 5000 milisegundos
            parar1();
            girare1();
            System.Threading.Thread.Sleep(1700);//para a execução em 1700 milisegundos
            parar1();
            moverf1();  //mova
            System.Threading.Thread.Sleep(5000);//para a execução em 5000 milisegundos
            parar1();
            girare1();
            System.Threading.Thread.Sleep(1900);//para a execução em 1900 milisegundos
            parar1();
            moverf1();  //mova
            System.Threading.Thread.Sleep(3000);//para a execução em 3000 milisegundos
            parar1();
            girard1();
            System.Threading.Thread.Sleep(100);//para a execução em 100 milisegundos
            parar1();
            moverf1();  //mova
            System.Threading.Thread.Sleep(10000);//para a execução em 10000
milisegundos
            parar1();
        }


  //programação a ser executado no robô, no evento do Timer a cada 1000ms
        private void timer1_Tick(object sender, EventArgs e)
        {
          try
            {

                dsBD.comandosRow linha = null;
                dsBD.comandosDataTable tb = null;

              //faz um select no banco de dados, para verificar dos próximos comandos
                tb = comandosTableAdapter1.GetDataByStatusVazio();

              //percorre todas as linhas retornandas no select acima
                for (int i = 0; i < tb.Count; i++)
                {
                    //le uma linha da tabela, e armazena na variável linha
                    linha = tb[i];
                    listBox1.Items.Add("Comando '" + linha.Comando + "' executado as "
+ linha.DataHora);

                    //altera o status do comando para "executado"
                    comandosTableAdapter1.ExecutaComando(linha.CodComando);


//condição para executar o método moverf1()
```

```
                        if ((linha.Comando.Equals("mova")) ||
(linha.Comando.Equals("mover")) || (linha.Comando.Equals("para frente")) ||
(linha.Comando.Equals("para cima")))
                    {
                        moverf1();
                    }

//condição para executar o método movert1()
                    if ((linha.Comando.Equals("para trás")) ||
(linha.Comando.Equals("para traz")) || (linha.Comando.Equals("traz")) ||
(linha.Comando.Equals("mover para traz")) || (linha.Comando.Equals("re")) ||
(linha.Comando.Equals("ré")) || (linha.Comando.Equals("atrás")))
                    {
                        movert1();
                    }

//condição para executar o método parar1()

                    if ((linha.Comando.Equals("pare")) ||
(linha.Comando.Equals("parar")) || (linha.Comando.Equals("robo pare")))
                    {
                        parar1();
                    }

//condição para executar o método virare1()
                    if ((linha.Comando.Equals("para esquerda")) ||
(linha.Comando.Equals("esquerda")) || ((linha.Comando.Equals("vire esquerda")) ||
(linha.Comando.Equals("virar a esquerda")) || (linha.Comando.Equals("virar à
esquerda")) || (linha.Comando.Equals("virar esquerda")) || (linha.Comando.Equals("vire
à esquerda")) || (linha.Comando.Equals("vire a esquerda"))))
                    {
                        virare1();
                    }

//condição para executar o método virard1()
                    if ((linha.Comando.Equals("para direita")) ||
(linha.Comando.Equals("direita")) || ((linha.Comando.Equals("vire direita")) ||
(linha.Comando.Equals("virar a direita")) || (linha.Comando.Equals("virar à direita"))
|| (linha.Comando.Equals("virar direita")) || (linha.Comando.Equals("vire à direita"))
|| (linha.Comando.Equals("vire a direita"))))
                    {
                        virard1();
                    }

//condição para executar o método girare1()
                    if ((linha.Comando.Equals("gire esquerda")) ||
(linha.Comando.Equals("girar a esquerda")) || (linha.Comando.Equals("girar à
esquerda")) || (linha.Comando.Equals("girar esquerda")) || (linha.Comando.Equals("gire
à esquerda")) || (linha.Comando.Equals("gire a esquerda")))
                    {
                        girare1();
                    }

//condição para executar o método girard1()
                    if ((linha.Comando.Equals("gire direita")) ||
(linha.Comando.Equals("girar a direita")) || (linha.Comando.Equals("girar à direita"))
|| (linha.Comando.Equals("gire à direita")) || (linha.Comando.Equals("girar direita"))
|| (linha.Comando.Equals("gire a direita")))
                    {
                        girard1();
                    }
```

```
//condição para executar o comando gire no próprio eixo 90 graus a esquerda
                    if (linha.Comando.Equals("esquerda 90"))
                    {
//comando gire no próprio eixo 90 graus a esquerda
                        girare1();
                        System.Threading.Thread.Sleep(1600);//para a execução em 1500
milisegundos
                        parar1();
                    }

//condição para executar o comando gire no próprio eixo 90 graus a direita
                    if (linha.Comando.Equals("direita 90"))
                    {
//comando gire no próprio eixo 90 graus a direita
                        girard1();
                        System.Threading.Thread.Sleep(1550);//para a execução em 1500
milisegundos
                        parar1();
                    }

//condição para executar o método obj();
                    if (linha.Comando.Equals("objeto"))
                    {
                        obj();
                    }

//condição para executar o método venha();
                    if (linha.Comando.Equals("venha"))
                    {
                        venha();
                    }

                    // condição para executar o comando andar no circuito em forma de
um quadrado
                    if (linha.Comando.Equals("circuito"))
                    {
//comando andar no circuito em forma de um quadrado
                        moverf1();  //mova
                        System.Threading.Thread.Sleep(5000);//para a execução em 5000
milisegundos
                        parar1(); // parar
                        girare1();// girar esquerda
                        System.Threading.Thread.Sleep(1500);//para a execução em 1500
milisegundos
                        parar1(); // parar
                        moverf1();  //mova
                        System.Threading.Thread.Sleep(5000);//para a execução em 5000
milisegundos
                        parar1(); // parar
                        girare1();// girar esquerda
                        System.Threading.Thread.Sleep(1500);//para a execução em 1500
milisegundos
                        parar1(); // parar
                        moverf1();  //mova
                        System.Threading.Thread.Sleep(5000);//para a execução em 5000
milisegundos
                        parar1(); // parar
                        girare1();// girar esquerda
                        System.Threading.Thread.Sleep(1500);//para a execução em 1500
milisegundos
                        parar1();
                        moverf1();  //mova
```

```
                        System.Threading.Thread.Sleep(5000);//para a execução em 5000
milisegundos
                        parar1(); // parar
                        girare1();// girar esquerda
                        System.Threading.Thread.Sleep(1500);//para a execução em 1500
milisegundos
                        parar1();

                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Ocorreu o erro ao consultar: " + ex.Message);
            }
        }

        //habilita a verificação no banco
        private void btnHabilita_Click_1(object sender, EventArgs e)
        {
            timer1.Enabled = true;
            btnHabilita.Enabled = false;
            btnDesabilita.Enabled = true;
        }

        //desabilita a verificação no banco
      private void btnDesabilita_Click_1(object sender, EventArgs e)
        {
            timer1.Enabled = false;
            btnHabilita.Enabled = true;
            btnDesabilita.Enabled = false;

        }

        //faz uma pesquisa no banco de dados, exibindo todos os comandos que já foram
executados
        private void btnPesquisar_Click(object sender, EventArgs e)
        {
            try
            {
                comandosTableAdapter1.Fill(dsBD1.comandos);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

7. Web.config

Tem a função de conectar o SDK Giga de Testes ao banco de dados “falabd”.

```
<?xml version="1.0"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->
<configuration>
  <configSections>
    <sectionGroup name="system.web.extensions"
type="System.Web.Configuration.SystemWebExtensionsSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
      <sectionGroup name="scripting"
type="System.Web.Configuration.ScriptingSectionGroup, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35">
        <section name="scriptResourceHandler"
type="System.Web.Configuration.ScriptingScriptResourceHandlerSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>
        <sectionGroup name="webServices"
type="System.Web.Configuration.ScriptingWebServicesSectionGroup,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35">
          <section name="jsonSerialization"
type="System.Web.Configuration.ScriptingJsonSerializationSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="Everywhere"/>
          <section name="profileService"
type="System.Web.Configuration.ScriptingProfileServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false" allowDefinition="MachineToApplication"/>
          <section name="authenticationService"
type="System.Web.Configuration.ScriptingAuthenticationServiceSection,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" requirePermission="false"
allowDefinition="MachineToApplication"/>
          <section name="roleService"
type="System.Web.Configuration.ScriptingRoleServiceSection, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"
requirePermission="false" allowDefinition="MachineToApplication"/>
        </sectionGroup>
      </sectionGroup>
    </sectionGroup>
  </configSections>
  <connectionStrings>
    <add name="conexaoBD" connectionString="server=localhost;User
Id=root;password=1234;Persist Security Info=True;database=falabd"
providerName="MySql.Data.MySqlClient"/>
  </connectionStrings>
  <system.web>
    <compilation debug="true">
      <assemblies>
        <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
```

```
        <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
        <add assembly="System.Xml.Linq, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=B77A5C561934E089"/>
        <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
      </assemblies>
    </compilation>
    <pages>
      <controls>
        <add tagPrefix="asp" namespace="System.Web.UI"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
        <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      </controls>
    </pages>
    <httpHandlers>
      <remove verb="*" path="*.asmx"/>
      <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      <add verb="GET,HEAD" path="ScriptResource.axd" validate="false"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    </httpHandlers>
    <httpModules>
      <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
    </httpModules>
  </system.web>
  <system.codedom>
      <compilers>
        <compiler language="c#;cs;csharp" extension=".cs"
type="Microsoft.CSharp.CSharpCodeProvider, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" warningLevel="4">
          <providerOption name="CompilerVersion" value="v3.5"/>
          <providerOption name="WarnAsError" value="false"/>
      </compiler>
    </compilers>
  </system.codedom>
  <system.webServer>
      <validation validateIntegratedModeConfiguration="false"/>
    <modules>
      <remove name="ScriptModule"/>
      <add name="ScriptModule" preCondition="managedHandler"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=3.5.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    </modules>
    <handlers>
      <remove name="WebServiceHandlerFactory-Integrated"/>
      <remove name="ScriptHandlerFactory"/>
      <remove name="ScriptHandlerFactoryAppServices"/>
      <remove name="ScriptResource"/>
      <add name="ScriptHandlerFactory" verb="*" path="*.asmx"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
```

```
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      <add name="ScriptHandlerFactoryAppServices" verb="*" path="*_AppService.axd"
preCondition="integratedMode" type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
      <add name="ScriptResource" verb="GET,HEAD" path="ScriptResource.axd"
preCondition="integratedMode" type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"/>
    </handlers>
  </system.webServer>
  <runtime>
    <assemblyBinding appliesTo="v2.0.50727" xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Extensions"
publicKeyToken="31bf3856ad364e35"/>
        <bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="System.Web.Extensions.Design"
publicKeyToken="31bf3856ad364e35"/>
        <bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="3.5.0.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

ANEXO

Instalação do *CvoiceControl 9 alpha*

Para instalar o *CvoiceControl* a partir dos fontes, deve-se criar uma pasta com o nome *cvoicecontrol* e nela efetuar o download com o comando *wget*, como no exemplo abaixo:

```
aluno@debian:~$ su
Senha:
root@debian:/home/aluno# cd ..
root@debian:/home# cd ..
root@debian:/# cd ..
root@debian:/# mkdir cvoice
root@debian:/# cd cvoice
root@debian:/cvoice# wget
http://www.kiecza.net/daniel/linux/cvoicecontrol-0.9alpha.tar.gz
```

Após o download, deve-se descompactar o arquivo *cvoicecontrol-0.9alpha.tar.gz*, com o comando "***tar -zxvf nome-do-arquivo-tar****",* como no exemplo abaixo:

```
root@debian:/cvoice#  tar -zxvf cvoicecontrol-0.9alpha.tar.gz
```

Em seguida, é necessário instalar os pacotes ***libncurses5-dev*** *e* ***MAKE***, utilizados na instalação do programa. Para tanto, digita-se:

```
root@debian:/cvoice#  apt-get install libncurses5-dev
```

Na sequência, instala-se o pacote *MAKE*, com o comando:

```
root@debian:/cvoice#  apt-get install make
```

Outro pacote que precisa ser instalado é o *KMIX*, ligado à mixagem de som e usado pelo *Microphone_config*: se isso não ocorrer, no momento da execução do comando *Microphone_config*, aparecerá a seguinte mensagem:

```
No mixer devices available!
Please purchase a sound card and install it!
```

Esse erro ocorre mesmo se o seu computador estiver com o som funcionando normalmente, pois ele acontece por falta de um dispositivo de Mixagem instalado, para instalar o Kmix digite:

```
root@debian:/cvoice#  apt-get install kmix
```

OBS: Para instalar tais pacotes, é necessário estar conectado à internet e com o CD de instalação do *Debian,* no Cd-Rom.

Depois da instalação de todos os pacotes, deve-se abrir a pasta *cvoicecontrol-0.9alpha* e executar o comando ***./configure*** para gerar o arquivo ***Makefile.*** Deve-se também usar os comandos ***make*** e ***make install***, os quais compilarão e instalarão o *CVoiceControl*. Veja o exemplo abaixo:

```
root@debian:/cvoice#  cd cvoicecontrol-0.9alpha
// rodando o comando ./configure
root@debian:/cvoice/cvoicecontrol-0.9alpha# ./configure
// rodando o comando make
root@debian:/cvoice/cvoicecontrol-0.9alpha# make
// rodando o comando make install
root@debian:/cvoice/cvoicecontrol-0.9alpha# make install
```

Calibrando o microfone

O *CvoiceControl* é composto de 3 binários:

- ✓ *microphone_config*
- ✓ *model_editor*
- ✓ *cvoicecontrol*

No primeiro passo, deve-se calibrar o microfone para o recurso de reconhecimento de voz, através do *microphone_config*. Após, utiliza-se o programa *model_editor* para criar modelos de voz/comandos.

Concluídos esses passos, encontram-se presentes os dois principais objetos para efetuar o processo de reconhecimento de voz, por meio do programa

CV*oiceControl*, responsável por reconhecer e executar o comando ***$ microphone_config.***

O problema verificado é que ao executar o *microphone_config*, gravar as configurações do *microphone_config* e gerar o arquivo ***Config*** no diretório ~/**cvoice**/**cvoicecontrol-0.9alpha**/**cvoicecontrol,** aparece a mensagem abaixo:

```
w-p      00024000      08:01      1638560      /lib/tls/i686/cmov/libm-2.10.1.so
00e47000-00e63000     r-xp     00000000     08:01     1622039     /lib/libgcc_s.so.1
00e63000-00e64000     r--p     0001b000     08:01     1622039     /lib/libgcc_s.so.1
00e64000-00e65000     rw-p     0001c000     08:01     1622039     /lib/libgcc_s.so.1
00e7e000-00e7f000         r-xp         00000000         00:00         0         [vdso]
08048000-08051000  r-xp  00000000  08:01  2009468  /usr/local/bin/microphone_config
08051000-08052000  r--p  00008000  08:01  2009468  /usr/local/bin/microphone_config
08052000-08053000  rw-p  00009000  08:01  2009468  /usr/local/bin/microphone_config
095a1000-095ca000         rw-p         00000000         00:00         0         [heap]
b77c7000-b77c9000             rw-p             00000000             00:00             0
b77e3000-b77e5000             rw-p             00000000             00:00             0
bfbea000-bfc1e000         rw-p         00000000         00:00         0         [stack]
Abortado
```

O programa tem um ***bug*** e para resolvê-lo, é necessário apagar ou comentar as linhas 1073 ***home = getenv("HOME")*** e a 1103 ***free(home),*** do arquivo ***microphone_config.c.***

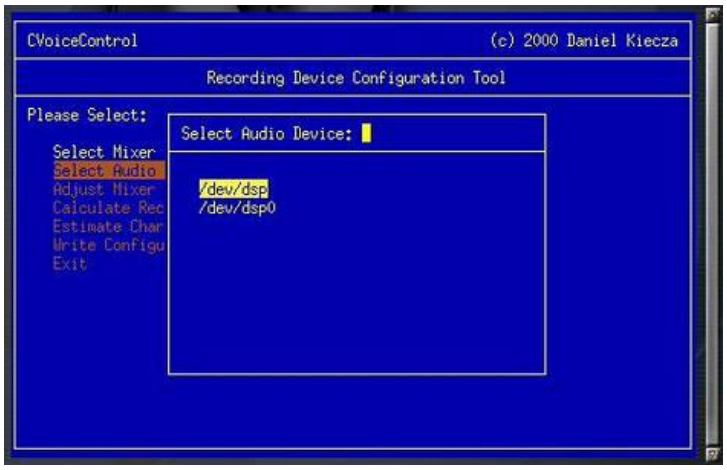
```
 //abra a pasta cvoicecontrol
root@debian:/cvoice/cvoicecontrol-0.9alpha# cd cvoicecontrol
//Digite vi microphone_config.c para abrir o modo de edição do
arquivo
root@debian:/cvoice/cvoicecontrol-0.9alpha/cvoicecontrol/vi
microphone_config.c
```

Posteriormente, pressionam-se as teclas **(esc : wq)**, para gravar as alterações feitas no arquivo ***microphone_config.c*** e recopila o programa novamente, através dos comandos ***Make*** e ***Make Install***, conseguindo-se, finalmente, executar o comando **$ *microphone_config***:

```
root@debian:/cvoice/cvoicecontrol-0.9alpha $ microphone_config
```

Seleciona-se a opção *"Select Mixer Device",* para escolher o dispositivo *Mixer* e sua placa de som *(/dev/mixer)*:

Selecione-se a opção "*Select Audio Device*", para definir o dispositivo da sua placa de som *(/dev/dsp)*:



Seleciona-se a opção *"Adjust Mixer Levels"*, para ajustar automaticamente o volume do seu microfone através do *mixer*:

Ao pressionar qualquer tecla, obtêm-se as telas ilustradas abaixo, momento em que se deve falar em voz alta diante o microfone (risadas altas colaboram para a avaliação do volume no *Mixer*). Esse procedimento é realizado para encontrar o melhor nível de volume de seu microfone, no ambiente que se encontra:

Seleciona-se "*Calculate Recording Thresholds"* para encontrar o nível do sinal sonoro mínimo, com objetivo de iniciar a captura de áudio. Na primeira fase deve-se permanecer em silêncio, para a captação de todos os sons externos proporcionados no ambiente. Na segunda, deve-se sustentar uma conversa até que o ciclo seja concluído com sucesso:


CVoiceControl
(c) 2000 Daniel Kiecza
Recording Device Configuration Tool
Please Select:
Select Mixer
Select Audio
Adjust Mixer
Calculate Rec
Estimate Char
Write Configu
Exit
Calculate Thresholds:
I'll calculate some values now. First,
I need to know the silence level of the
microphone. Please be silent until I
say I'm done:
Press any key to start ...


CVoiceControl
(c) 2000 Daniel Kiecza
Recording Device Configuration Tool
Please Select:
Select Mixer
Select Audio
Adjust Mixer
Calculate Rec
Estimate Char
Write Configu
Exit
Calculate Thresholds:
stand by ...


CVoiceControl
(c) 2000 Daniel Kiecza
Recording Device Configuration Tool
Please Select:
Select Mixer
Select Audio
Adjust Mixer
Calculate Rec
Estimate Char
Write Configu
Exit
Calculate Thresholds:
Now, I'll define the thresholds at which
to start/stop recording. Please talk
at a conversational volume (speech
recognition volume) until I say stop!
Press any key to start ...

Seleciona-se *"Estimative Characteristics of Recording Channel"* para encontrar o nível mínimo do volume do microfone, através da análise dos ruídos do ambiente. Para efetuar essa verificação, deve-se permanecer em silêncio novamente com a finalidade de captação de todos os sons externos proporcionados no ambiente:

Pressiona-se (*ENTER*) na opção *"Write Configuration",* para salvar a calibração do microfone e, logo após, seleciona-se *Exit* para sair desse módulo.

Antes de aparecer a tela de **configurações salvas com sucesso (*"Save Configuration Success"),*** surgirá um erro exibindo uma mensagem de que o diretório home não foi localizado e que o arquivo ***Config*** será gravado no diretório ***/tmp/config*,** devendo, então, teclar (*ENTER*):

Dessa forma o arquivo **Config**, que possui as configurações de calibração do microfone, estará salvo no diretório /**tmp**. Faz-se necessário copiar o arquivo **config** desse diretório para o ***/cvoice/cvoicecontrol-0.9alpha/cvoicecontrol/***, com o comando **CP origem destino:**

```
root@debian:/cvoice/cvoicecontrol-0.9alpha# cp /tmp/config
/cvoice/cvoicecontrol-0.9alpha/cvoicecontrol/
```

ou

```
debian:/tmp# cp config /root/.cvoicecontrol/
```

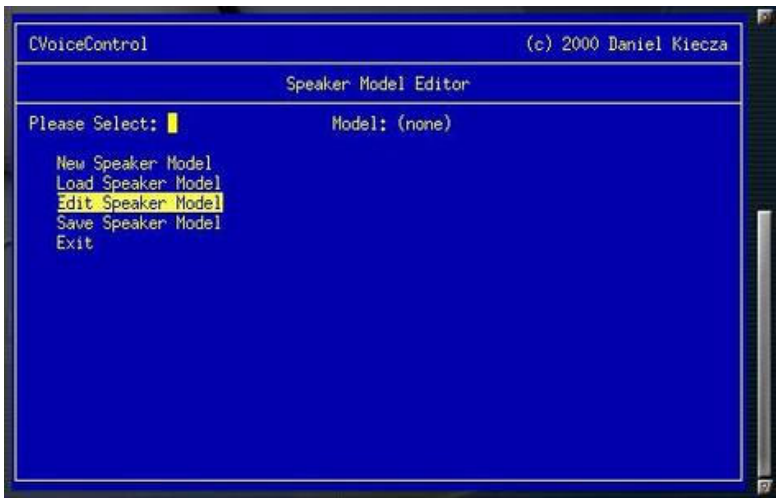
Confirma-se a cópia do arquivo para o destino: se todas estas etapas foram concluídas, o microfone foi calibrado com sucesso.

Aprendizado de comandos e controle através do reconhecimento de voz

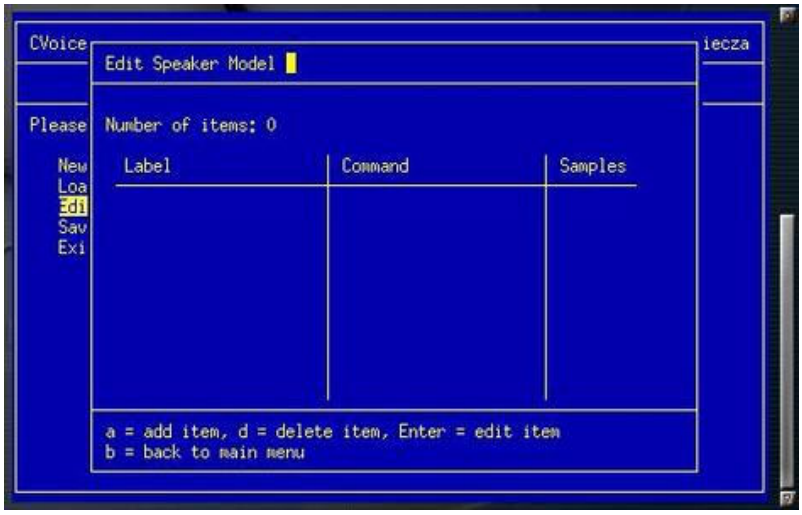
Nessa etapa, serão associadas amostragens sonoras a um determinado comando *Linux* /Unix: cada comando deve possuir no MÍNIMO quatro amostragens.

```
root@debian:/cvoice/cvoicecontrol-0.9alpha# model_editor
```

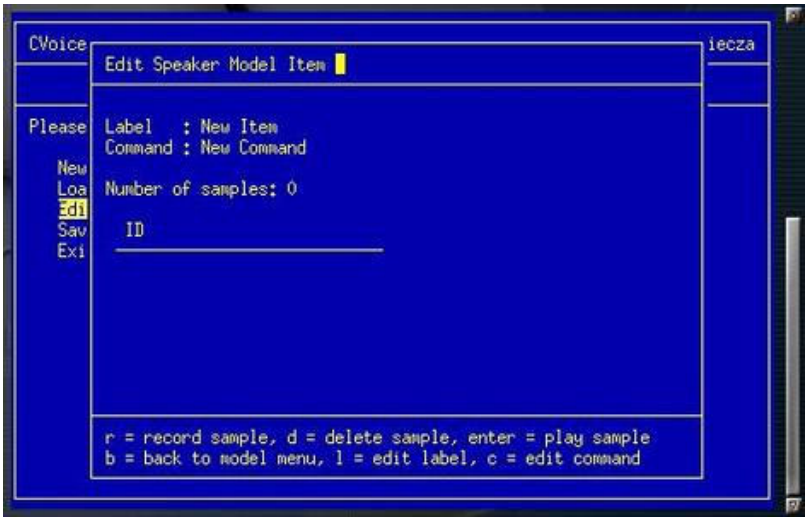
Ao executar o comando *model_editor*, deve-se selecionar a opção *"Edit Speaker Model*", como no exemplo abaixo:



Pressiona-se a tecla (a), para adicionar um comando ao arquivo de modelos:



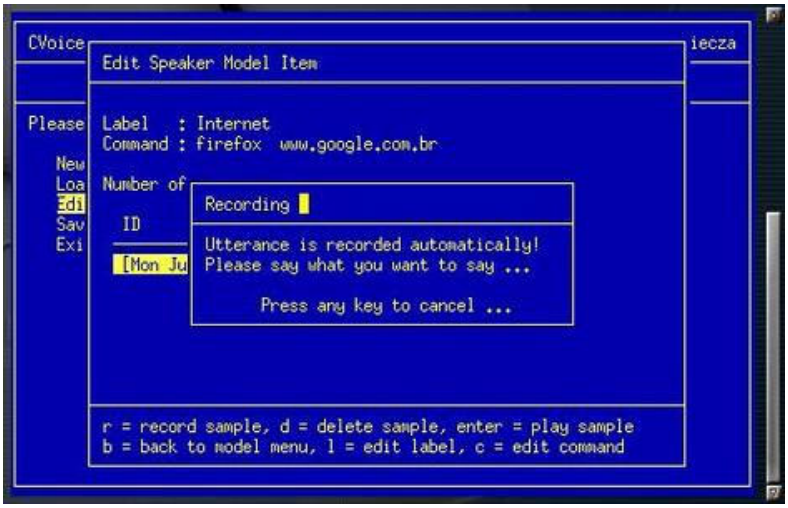
Pressiona-se (ENTER), para editar o novo modelo de comando:

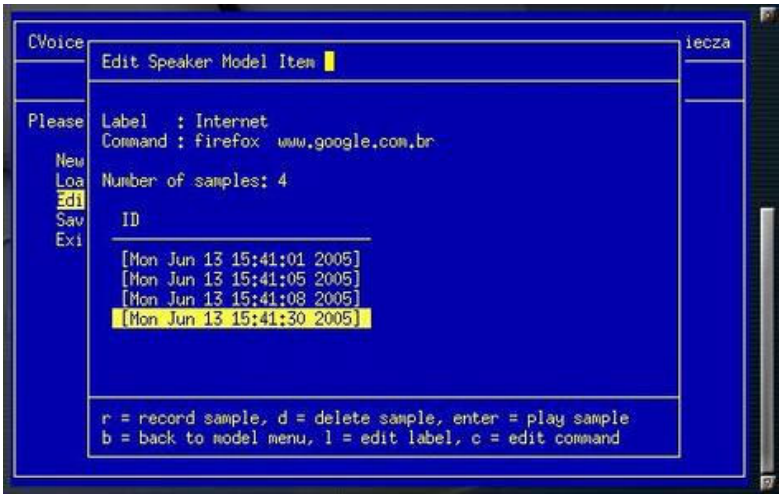


Pressiona-se (L), para editar o Título do comando:

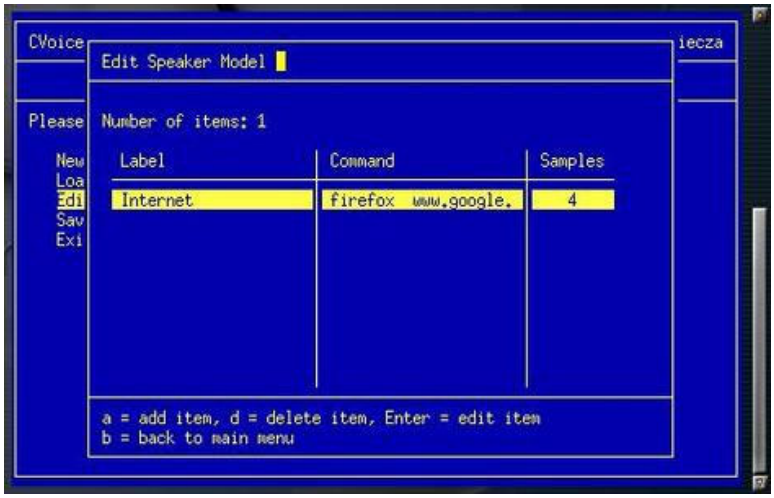
Pressiona-se (C), para editar o comando a ser executado:

Pressiona-se (R), visando a gravar uma amostragem de voz para o comando atual. Repete-se esta operação, no mínimo quatro vezes:

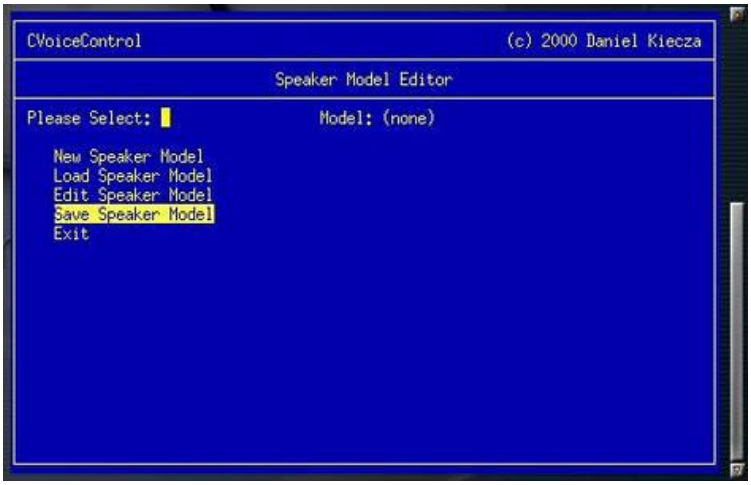




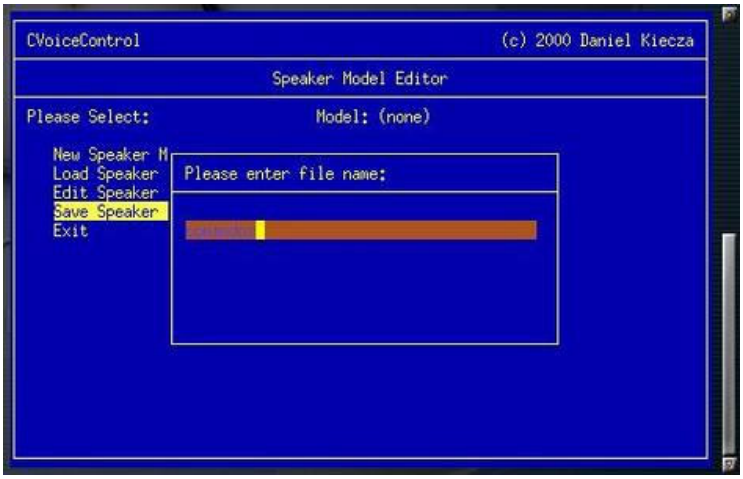
Pressiona-se (B), para voltar ao menu anterior:



Agora, seleciona-se a opção "Save Speaker Model", para salvar os comandos inseridos e/ou alterados:



Digita-se o nome do arquivo que deseja salvar e pressiona-se (ENTER):



Nesse exemplo, os comandos foram salvos com o nome **comandos.cvc.** Para testar o seu Linux, obedecendo-se os comandos de voz, basta executar o C*VoiceControl* <nome-arquivo-de-comandos>, conforme demonstração abaixo: **root@debian:/cvoice/cvoicecontrol-0.9alpha# cvoicecontrol comandos.cvc**