

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

CONTROLE POR VOZ UTILIZANDO A ENGINE JULIUS
COM FALA CONTÍNUA

DEIVID GEOVANI SANT'ANA

BLUMENAU
2014

2014/1-05

DEIVID GEOVANI SANT'ANA

CONTROLE POR VOZ UTILIZANDO A ENGINE JULIUS

COM FALA CONTÍNUA

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Profa. Joyce Martins – Orientadora

**BLUMENAU
2014**

2014/1-05

CONTROLE POR VOZ UTILIZANDO A ENGINE JULIUS COM FALA CONTÍNUA

DEIVID GEOVANI SANT'ANA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:

Profa. Joyce Martins, Mestre – Orientadora, FURB

Membro:

Prof. Aurélio Faustino Hoppe, Mestre – FURB

Membro:

Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, 09 de julho de 2014

Dedico esse trabalho aos meus pais que sempre me colocaram no caminho de uma boa educação e sempre mostraram como é importante o saber.

AGRADECIMENTOS

A Deus, por sempre me dar forças em todos os momentos.

A minha mãe Luciana Kormann, que de uma forma especial e carinhosa cuidou de mim e me concedeu uma das maiores alegrias que tive na vida, minha irmã Luana. Ao meu pai Nei Sant'ana, que sempre buscou me apresentar o melhor caminho para ser seguido.

A minha namorada Jaciara Preti, que com seu amor e paciência, soube entender e me dar forças para seguir em frente, escutando todos os termos técnicos possíveis e me auxiliando no bom português para que eu pudesse concluir este trabalho.

Aos meus amigos, que com muita vontade e alegria deram total apoio ao término desse trabalho. Principalmente o meu grande amigo de FURB e da vida Vander Bertolini, que sempre me motivou e elogiou a proposta desse trabalho, sempre presente, motivando a concluir o curso nas grandes andanças que tivemos juntos até a FURB. Ao grande amigo José Guilherme Vanz sempre presente nas idas para FURB, sempre pronto a ajudar. A minha amiga Angélica Minella, que sempre tive como uma grande irmã, apoiando desde o início esse trabalho. Ao amigo e irmão Felix Maciel, por me apoiar e me incentivar a escrever, seguindo em frente com a conclusão desse trabalho.

Um agradecimento especial para Marcio Michelluzzi e professor Aurélio Faustino Hoppe, que disponibilizaram um pouco do seu tempo para que eu pudesse utilizar o Laboratório de Robótica da FURB, sem o qual não seria possível a conclusão deste trabalho.

A minha orientadora Joyce Martins, que com toda sua paciência soube me conduzir para que eu pudesse dar o melhor de mim, tornando o desafio de produzir um trabalho de conclusão de curso prazeroso e divertido.

Tudo que somos é poeira no vento.

Kerry Livgren

RESUMO

Este trabalho tem por objetivo o controle de um robô Lego NXT por reconhecimento de voz por fala contínua. Para tanto foi utilizada a *engine* Julius encapsulada através da LapsAPI para o reconhecimento de voz. Foi elaborada uma linguagem aceita pelo robô utilizando a *Backus-Naur Form* (BNF). Os comandos de voz são processados pelo programa e enviados via *bluetooth* para o robô Lego NXT que executa a ação conforme o comando transmitido. O objetivo principal de o robô reconhecer os comandos por fala contínua foi alcançado, mas por limitações físicas relacionadas com as esteiras responsáveis pela locomoção, o robô não executa esses comandos de forma correta. O reconhecimento dos comandos foi demonstrado com a captação de voz de dez locutores com características diferentes, como idade e sexo, mostrando que a aplicação com a *engine* Julius é independente de locutor.

Palavras-chave: Reconhecimento de fala contínua. *Engine* Julius. Lego NXT.

ABSTRACT

The objective of this paper is to control a Lego NXT robot by continuous voice recognition. For that, it was used Julius engine encapsulated through LapsAPI for the voice recognition. It was developed a language accepted by the robot using the Backus-Naur Form. The voice command is processed by the program and sent via bluetooth to the Lego NXT robot that performs the action accordingly to the command transmitted. The main goal of the robot, to recognize command by speech, was reached. But because of the physical limitations of the tracks responsible for the locomotion, it does not execute those commands properly. The recognition of controls was demonstrated with the voice pickup of ten speakers from different characteristics, such as age and sex, showing that the application with the Julius engine is independent of speaker.

Key-words: Speech recognition. Julius engine. Lego NXT.

LISTA DE ILUSTRAÇÕES

Figura 1 – Visão da <i>engine</i> Julius.....	17
Figura 2 – Por dentro da <i>engine</i> Julius	17
Quadro 1 – Exemplos de palavras do dicionário do LapSAM 1.5	18
Quadro 2 – Arquivo <code>.grammar</code>	19
Quadro 3 – Arquivo <code>.voca</code>	19
Figura 3 – Ligação da API com a <i>engine</i> Julius.....	20
Quadro 4 – Exemplo de gramática especificada com SRGS.....	20
Quadro 5 – Métodos para uso da LapsAPI.....	21
Figura 4 – Bloco da UCP com todos os motores e sensores conectados.....	21
Figura 5 – Sensores do Lego NXT	21
Figura 6 – Servo-motor	22
Figura 7 – Interface do reconhecimento automático de fala por computador	23
Quadro 6 – Gramática da linguagem reconhecida.....	24
Figura 8 – Interface de reconhecimento de voz em cadeira de rodas.....	25
Figura 9 – Interface de execução do comando na cadeira de rodas	25
Figura 10 – <i>Black technic, link tread</i>	28
Figura 11 – JohnNXT.....	29
Figura 12 – Robô montado	29
Figura 13 – Robô montado, visão lateral.....	30
Figura 14 – Diagrama de classes do robô.....	31
Quadro 7 – Membros x classes do pacote <code>tcc.inteligencia</code>	32
Quadro 8 – Conexão e abertura do canal <i>bluetooth</i>	32
Quadro 9 – Processamento do comando de voz recebido	33
Quadro 10 – Seleção de ações do robô.....	33
Quadro 11 – Movimentação de um servo-motor.....	34
Quadro 12 – Arquivo <code>.grammar</code> da linguagem proposta	34
Quadro 13 – Arquivo <code>.voca</code> da linguagem proposta	35
Figura 15 – Compilação dos arquivos <code>.grammar</code> e <code>.voca</code>	36
Figura 16 – Diagrama de casos de uso	37
Quadro 14 – Caso de uso: conectar com o robô Lego NXT.....	37
Quadro 15 – Caso de uso: iniciar reconhecimento	37

Quadro 16 – Caso de uso: falar comando.....	38
Figura 17 – Classe para o reconhecimento de voz	39
Figura 18 – Diagrama de sequência	39
Quadro 17 – Código de inicialização da <i>engine</i> Julius.....	40
Quadro 18 – Seleção do comando a ser enviado para o robô.....	41
Figura 19 – Ambiente Eclipse, destacando o símbolo de projeto e o botão LeJOS.....	42
Figura 20 – Interface da aplicação de reconhecimento de voz.....	42
Quadro 19 – Quadro comparativo entre os trabalhos correlatos e trabalho desenvolvido	46
Quadro 20 – Peças	51
Quadro 21 – Arquivo <code>comandos.dfa</code>	58
Quadro 22 – Arquivo <code>comandos.term</code>	59
Quadro 23 – Arquivo <code>comandos.dict</code>	60
Quadro 24 – Arquivo de configuração da <i>engine</i> Julius	61

LISTA DE TABELAS

Tabela 1 – Análise de reconhecimento de comandos.....	44
---	----

LISTA DE SIGLAS

API - *Application Programming Interface*

ARPA - *Advanced Reserch Project Agency*

BNF - *Backus Naur Form*

BSD - *Berkeley Software Distribution*

CLR - *Common Language Runtime*

DLL - *Dynamic Link Library*

FURB - Universidade Regional de Blumenau

HMM - *Hidden Markov Model*

IDE - *Integrated Development Environment*

LaPS - Laboratório de Processamento de Sinais

LeJOS - *Lego Java Operating System*

LVCSR - *Large Vocabulary Continuous Speech Recognition*

MFCC - *Mel Frequency Cepstral Coefficient*

PIC - *Peripheral Interface Controller*

RAV - Reconhecimento Automático de Voz

RF - Requisito Funcional

RNF - Requisito Não Funcional

SRGS - *Speech Recognition Grammar Specification*

UCP - Unidade Central de Processamento

UFPA - Universidade Federal do Pará

USB - *Universal Serial Bus*

W3C - *World Wide Web Consortium*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 OBJETIVOS DO TRABALHO	13
1.2 ESTRUTURA DO TRABALHO	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 RECONHECIMENTO DE VOZ	15
2.2 <i>ENGINE JULIUS</i>	16
2.2.1 Modelo acústico	18
2.2.2 Modelo de linguagem.....	19
2.2.3 LapsAPI	20
2.3 ROBÔ LEGO NXT	21
2.4 TRABALHOS CORRELATOS	23
2.4.1 Reconhecimento automático de fala por computador	23
2.4.2 Reconhecimento de voz para aplicação em cadeira de rodas	24
2.4.3 Reconhecimento de voz através de reconhecimento de padrões	25
3 DESENVOLVIMENTO	27
3.1 REQUISITOS PRINCIPAIS DO SISTEMA	27
3.2 ROBÔ LEGO NXT	27
3.2.1 Montagem do robô	27
3.2.2 Especificação da inteligência do robô.....	30
3.2.3 Implementação da inteligência do robô	32
3.3 LINGUAGEM.....	34
3.3.1 Especificação da linguagem.....	34
3.3.2 Compilação do modelo de linguagem.....	36
3.4 RECONHECIMENTO DE VOZ	36
3.4.1 Especificação do reconhecimento de voz	37
3.4.2 Implementação do reconhecimento de voz	40
3.5 IMPLEMENTAÇÃO	41
3.5.1 Técnicas e ferramentas utilizadas.....	41
3.5.2 Operacionalidade da ferramenta	42
3.6 RESULTADOS E DISCUSSÃO	43
4 CONCLUSÕES.....	47

4.1 EXTENSÕES	47
REFERÊNCIAS	49
APÊNDICE A – Quadro das peças para a montagem do robô.....	51
APÊNDICE B – Arquivo comandos.dfa	58
APÊNDICE C – Arquivo comandos.term.....	59
APÊNDICE D – Arquivo comandos.dict	60
ANEXO A – Arquivo de configuração da engine Julius.....	61

1 INTRODUÇÃO

Desde a pré-história o homem tem usado sua inteligência para aumentar seu conforto e reduzir seus esforços. Neste sentido, os serviços digitais são uma ótima solução para o homem moderno. Na sociedade atual a demanda é pelo crescente avanço tecnológico, através da busca de novos meios para disseminar informações com alta velocidade e desempenho. Uma destas tecnologias usa o controle de voz para possibilitar a execução de tarefas simples em paralelo, como, por exemplo, dirigir um carro e atender o telefone celular, controlar um ou vários robôs na execução de tarefas ou movimentar objetos, permitindo comandar aparelhos eletrônicos sem contato físico direto. Pode-se citar também a possibilidade de uma pessoa com deficiência de locomoção controlar por voz uma cadeira de rodas (BARCELOS et al., 2008). Sistemas deste tipo são conhecidos como de Reconhecimento Automático de Voz (RAV).

Segundo Ynoguti (1999, p. 8), um sistema RAV consiste em mapear um sinal acústico capturado de um transdutor (usualmente um microfone) em um conjunto de palavras. A partir deste conjunto de palavras pode-se originar comandos para que determinado objeto realize uma ação. Geralmente um sistema RAV tem como entrada um sinal de fala e como saída as palavras reconhecidas. Existem algumas soluções para sistemas RAV, entre as quais, cita-se Dragon (NUANCE COMMUNICATION, 2013), que reconhece sequências de palavras e não somente palavras isoladas, e a *engine* Julius (LEE, 2009), que reconhece desde palavras isoladas até sequências contínuas de palavras, podendo trabalhar com um grande vocabulário.

A *engine* Julius funciona como uma *Dynamic Link Library* (DLL), sendo, de acordo com Lee (2009), um decodificador de alta performance para *Large Vocabulary Continuous Speech Recognition* (LVCSR). Julius utiliza dois modelos para o reconhecimento de voz, o modelo acústico e o modelo de linguagem. O modelo acústico determina um modelo matemático que representa a palavra a ser processada, enquanto o modelo de linguagem permite que o processamento possa ser feito de forma mais rápida e precisa.

Diante do exposto, foi proposto o desenvolvimento de uma aplicação com o objetivo de mostrar a capacidade de reconhecimento de voz da *engine* Julius para fala contínua. Para tanto, foi definido um modelo de linguagem para comandar um robô Lego NXT.

1.1 OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo principal desenvolver uma aplicação para efetuar controle por voz utilizando a *engine* Julius com fala contínua.

Os objetivos específicos do trabalho são:

- a) controlar por voz um robô Lego NXT;
- b) disponibilizar um vocabulário para comandar o robô;
- c) enviar os comandos de voz para o robô através de conexão *bluetooth*.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos, sendo eles: introdução, fundamentação teórica, desenvolvimento e conclusão. O capítulo 2 apresenta a fundamentação teórica, expondo o histórico do reconhecimento de voz e os problemas clássicos do mesmo, passando pela visão geral da *engine* Julius e o seu encapsulamento com a LapsAPI, chegando ao Lego NXT e o LeJOS, e, por fim, descrevendo três trabalhos correlatos a esse. O capítulo 3 mostra o desenvolvimento desse trabalho, incluindo montagem, especificação e implementação do robô Lego NXT; especificação e compilação da linguagem proposta para controlar o robô; especificação e implementação da aplicação de reconhecimento de voz; além das ferramentas utilizadas, da operacionalidade e dos resultados obtidos. O capítulo 4 encerra com a conclusão, dando ideias sobre possíveis extensões para esse trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está dividido em quatro seções. A seção 2.1 trata brevemente do histórico do reconhecimento de voz, além de enumerar alguns problemas relacionados ao reconhecimento de voz. A seção 2.2 apresenta a *engine* Julius e o seu funcionamento. A seção 2.3 traz uma visão geral sobre o Lego NXT e a ferramenta LeJOS. Por fim, a seção 2.4 mostra três trabalhos correlatos a este.

2.1 RECONHECIMENTO DE VOZ

Antes mesmo da computação ser o que se conhece hoje, muitos esforços foram feitos para que aparelhos eletrônicos reconhecessem a voz humana. Percursor nesse tipo de pesquisa, o *Bell Telephone Laboratories* desenvolveu em 1952 um circuito que reconhecia os dígitos de 0 a 9. Este reconhecimento era baseado nas medidas de ressonâncias espectrais das vogais de cada dígito (DAVIS; BIDULPH; BALASHEK, 1952).

Na década de 70 o Departamento de Defesa dos Estados Unidos da América através da *Advanced Reserch Project Agency* (ARPA) investiu muito tempo e dinheiro para melhorar as aplicações de reconhecimento de voz. A ARPA tinha objetivos claros de simplificar o problema de reconhecimento de voz aplicando regras sintáticas e semânticas, além de melhorar a tecnologia já existente. Surgiram analisadores léxico, sintático e semântico para o reconhecimento de palavras, além de um grande avanço na forma como sistemas RAV são desenvolvidos (KLATT, 1990, p. 563).

Nos anos 80, vários pesquisadores iniciaram pesquisas para reconhecimento de palavras conectadas utilizando métodos de modelos estatísticos, sendo o maior destaque para os modelos ocultos de Markov (*Hidden Markov Models* (HMM)) usados para modelar séries temporais, como por exemplo a voz. Além disso, um estudo mais profundo mostrou a possibilidade de aplicação de redes neurais na classificação de sinais. (SILVA, 2010, p. 6).

Segundo Silva (2010, p. 6), já existem bancos de dados de voz com mais de 180 horas de treinamento para o inglês, mas essa não é a realidade do português do Brasil.

À primeira vista, o processo de reconhecer a voz humana pode parecer simples por ser algo que o ser humano é capaz de fazer naturalmente e sem esforço. Contudo, fazer com que uma máquina seja capaz de reconhecer a voz humana é um problema muito complexo, envolvendo diversas áreas de conhecimento. (OLIVEIRA, 2002, p. 10).

O maior problema do reconhecimento de voz não é transcrição do sinal analógico para o sinal digital, mas sim o do reconhecimento deste sinal digital em modelos conhecidos de palavra. Para lidar com reconhecimento de voz é necessário interpretar a fala através da manipulação da representação do conhecimento fonético-fonológico (VIEIRA; LIMA, 2001).

Segundo Silva (2010, p. 9), mesmo com tantos avanços na área da computação e do reconhecimento de palavras, o entendimento completo da fala humana por um computador é um processo complexo. Alguns fatores relacionados são:

- a) a mesma palavra pronunciada várias vezes pode apresentar características diferentes devido à articulação dos órgãos do aparelho fonador (SILVA, 2010, p. 9);
- b) o sotaque do locutor tem grande influência na forma como a palavra é pronunciada (LOUZADA, 2010, p. 13), o que dificulta a implementação do reconhecimento do português do Brasil que, por ser um país de tamanho continental e ter sofrido influência na colonização por vários povos, tem sotaques muito variados;
- c) em idiomas que possuem um vocabulário muito extenso, como o português do Brasil, existem várias palavras com a mesma pronúncia e significados diferentes, como por exemplo: sessão e cessão, concerto e conserto, mas e mais, entre outras (SILVA, 2010, p. 9);
- d) a pronúncia de uma palavra pode variar de locutor para locutor, sendo que alguns podem falar mais rápido ou “engolir” sílabas e letras (LOUZADA, 2010, p. 13);
- e) a relação sinal-ruído influencia o reconhecimento de voz, pois quanto menos ruído perto da fonte de entrada de áudio, melhor será o reconhecimento, enquanto ambientes com muito ruído prejudicam o desempenho do reconhecedor de voz (SILVA, 2010, p. 9).

Atualmente existem muitas ferramentas de reconhecimento de voz, geralmente adaptadas para reconhecer idiomas como o inglês e o mandarim. Entre estas ferramentas, pode-se citar a *engine Julius*.

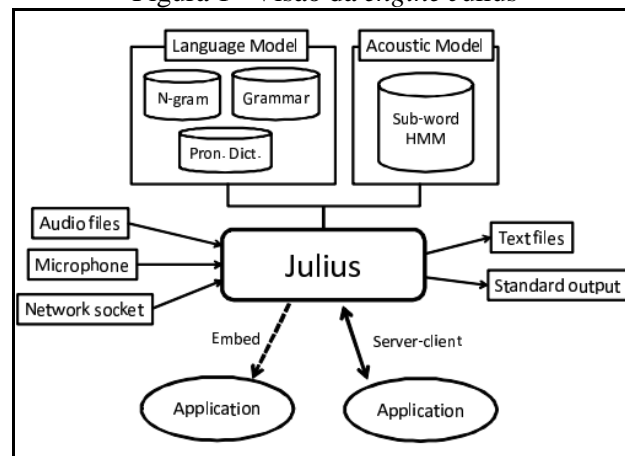
2.2 ENGINE JULIUS

A *engine Julius* é um software *open-source* de reconhecimento de voz, eficaz no processamento de voz em tempo real, utilizada tanto no meio acadêmico como industrial (LEE, 2009), que vem sendo desenvolvida desde 1998 na Kyoto University. Muitos institutos do Japão utilizam-na em pesquisas em vários idiomas, como inglês, francês, mandarim, esloveno, coreano, entre outros (LEE; KAWAHARA, 2009, p. 131).

A *engine Julius* foi escrita em C puro, rodando nas plataformas Windows, Linux, Mac OS e iOS. É distribuída através de uma licença parecida com a *Berkeley Software Distribution* (BSD), onde não há restrição na utilização para o desenvolvimento de pesquisa, comercial e industrial (LEE; KAWAHARA, 2009, p. 131).

Na Figura 1 tem-se uma visão da *engine*.

Figura 1 –Visão da *engine* Julius



Fonte: Lee e Kawahara (2009, p. 131).

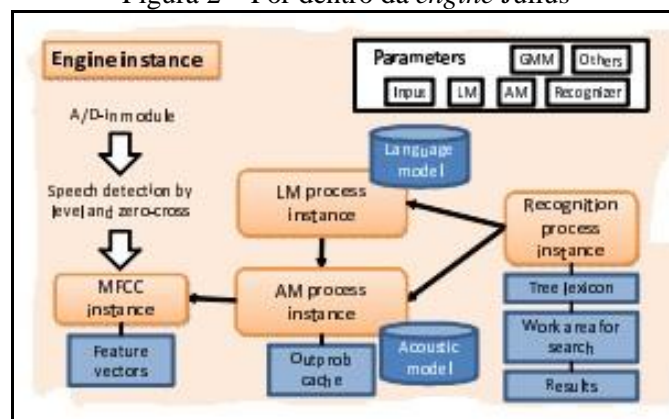
A entrada de dados para a *engine* pode ser através de microfone, de arquivos de áudio ou de *sockets* de rede. A saída pode ser um arquivo texto com as palavras reconhecidas ou a saída padrão da linguagem de programação utilizada, em geral mostrando na tela as palavras reconhecidas.

A aplicação que utiliza a *engine* Julius pode interagir de duas formas: através de um *socket* baseado em cliente-servidor ou através de uma DLL (LEE; KAWAHARA, 2009, p. 131). No entanto, a maneira mais difundida é a utilização da *engine* Julius encapsulada como DLL, sendo as funções de reconhecimento chamadas pela aplicação que a utiliza.

Um modelo acústico (*Acoustic Model*) e um modelo de linguagem (*Language Model*) são necessários para executar o reconhecimento de voz (LEE, 2009).

Na Figura 2 pode-se observar como a *engine* Julius funciona internamente.

Figura 2 – Por dentro da *engine* Julius



Fonte: Lee e Kawahara (2009, p. 133).

A instância da *engine* contém os seguintes módulos: entrada de áudio (A/D-in), detecção de voz (Speech detection), extração de características (MFCC), modelo de

linguagem (LM process), modelo acústico (AM process) e processo de reconhecimento de palavras (Recognition process). A instância do modelo acústico tem um HMM, modelo estatístico utilizado para reconhecer o padrão de palavras faladas através de fonemas. O modelo acústico cria uma instância do *Mel Frequency Cepstral Coefficient* (MFCC), ou seja, um filtro de ruído que extrai as características da forma de onda imputada pela entrada de áudio. A instância do modelo de linguagem calcula a probabilidade linguística de cada palavra. Essas duas instâncias são utilizadas pela instância do processo de reconhecimento de palavras para apresentar o resultado desejado, ou seja, a palavra reconhecida (LEE; KAWAHARA, 2009, p. 132).

2.2.1 Modelo acústico

O modelo acústico é definido através de HMM, sendo a representação matemática gerada a partir de um sinal acústico. As características deste sinal são extraídas e interpretadas para determinar a palavra correspondente.

Os modelos acústicos são dependentes de linguagem. Sendo assim, para ter um sistema que reconheça o português do Brasil é necessário ter um modelo acústico próprio para o idioma. No Brasil, os esforços para a implementação de um modelo acústico são do Laboratório de Processamento de Sinais (LaPS) da Universidade Federal do Pará (UFPA). Segundo Klautau et al. (2012), os pesquisadores elaboraram um corpora com mais de quinze horas de treinamento em português do Brasil, criando uma grande base de dados de um modelo acústico do idioma, denominado LaPSAM 1.5.

O dicionário do LaPSAM 1.5 tem um total de 65.531 palavras. Ele contém a palavra propriamente dita juntamente com os fonemas que a compõem. O Quadro 1 ilustra como o dicionário é organizado.

Quadro 1 – Exemplos de palavras do dicionário do LapSAM 1.5

1.	...
2.	azzurra a z z u R a
3.	baba b a b a
4.	babá b a b a
5.	babaca b a b a k a
6.	...
7.	mun <u>do</u> m u ~ d u
8.	...
9.	oi o j
10.	...
11.	olá o l a
12.	...
13.	saturno s a t u R n u
14.	...

Fonte: Klautau et al. (2012).

2.2.2 Modelo de linguagem

Segundo Louzada (2010, p. 21), somente o modelo acústico não é a maneira mais fácil de obter um bom desempenho do sistema, pois nem sempre uma palavra é dita de forma isolada. Sendo assim, o modelo de linguagem tem como objetivo estimar de forma mais confiável a probabilidade de ocorrência de uma determinada sequência de palavras.

O modelo de linguagem consiste em um dicionário de pronúncias (arquivo `.voça`) e restrições sintáticas (arquivo `.grammar`). As restrições sintáticas podem ser regras baseadas em gramáticas ou uma simples lista de palavras isoladas (LEE, 2009). O Quadro 2 apresenta um exemplo de restrições sintáticas para um modelo de linguagem, especificadas através da notação BNF.

Quadro 2 – Arquivo `.grammar`

1.	FALA: INICIO FRASE FIM
2.	FRASE: SAUDACAO MUNDO

Pode-se observar no Quadro 2 que existe um símbolo inicial `FALA` que faz referência às palavras reservadas `INICIO` e `FIM` e ao símbolo `FRASE`, que, por sua vez, tem duas palavras reservadas (`SAUDACAO` e `MUNDO`).

O arquivo `.voça` (Quadro 3) descreve cada palavra reservada, sendo que a cada uma delas podem estar associadas várias palavras. O arquivo possui: a palavra reservada usada na especificação das restrições sintáticas e a palavra que é propriamente reconhecida, acompanhada por seu conjunto de fonemas. O reconhecedor do Julius utiliza esses fonemas para o reconhecimento da palavra.

Quadro 3 – Arquivo `.voça`

1.	% INICIO	
2.	<s>	sil
3.		
4.	% FIM	
5.	<\s>	sil
6.		
7.	% SAUDACAO	
8.	olá	o l a
9.	oi	o j
10.		
11.	% MUNDO	
12.	Mundo	m u~ d u
13.	Marte	m a X tS i
14.	Lua	l u a
15.	Saturno	s a t u R n u

No Quadro 3 têm-se as seguintes especificações: as palavras reservadas `INICIO` e `FIM` representam o silêncio inicial e final de cada sentença, representado pela sequência `sil`; a palavra reservada `SAUDACAO` usada quando do reconhecimento de `olá` e `oi`; a palavra reservada `MUNDO` usada quando do reconhecimento das palavras `Mundo`, `Marte`, `Lua` e `Saturno`.

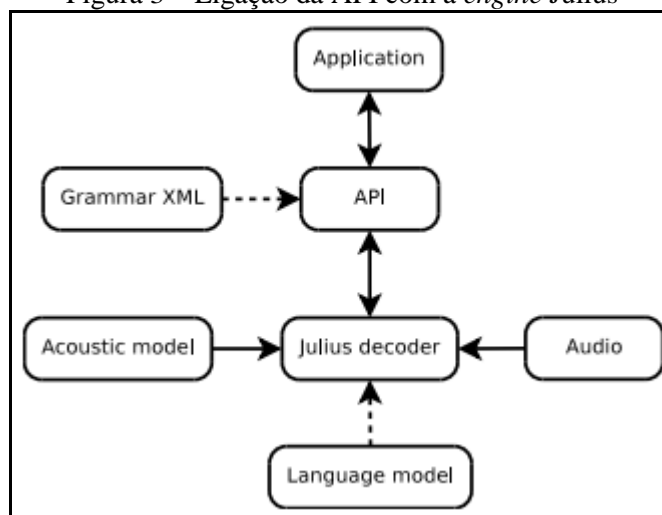
Assim, conforme especificado, algumas saudações possíveis são `olá Mundo`, `oi Mundo`, `olá Lua`, e assim por diante.

2.2.3 LapsAPI

Para facilitar a programação, o LaPS construiu uma *Application Programming Interface* (API) que encapsula a *engine* Julius. Ela foi desenvolvida em C++ com a especificação *Common Language Runtime* (CLR), para o funcionamento em todas as linguagens prevista para a plataforma .NET. A proposta da API é controlar a *engine* Julius em tempo real, bem como a interface de entrada de áudio do sistema (KLAUTAU et al., 2012).

Na Figura 3 observa-se a ligação da API com a interface do Julius. Comparando a Figura 3 com a Figura 1, verifica-se que a API fica entre o Julius e a aplicação. Além disso, a API pode usar uma gramática (`Grammar.XML`) especificada com *eXtensible Markup Language* (XML).

Figura 3 – Ligação da API com a *engine* Julius



Fonte: Klautau et al. (2012).

Segundo Louzada (2010, p. 25), usar XML para especificar o modelo de linguagem facilita o desenvolvimento da gramática. O modelo é especificado com o padrão *Speech Recognition Grammar Specification* (SRGS) definido pela *World Wide Web Consortium* (W3C), que é composto por *tags* de controle. O Quadro 4 traz um exemplo desse padrão.

Quadro 4 – Exemplo de gramática especificada com SRGS

```

1. <GRAMMAR>
2.   <RULE>
3.     <LIST>
4.       <P>UM</P>
5.       <P>DOIS</P>
6.       <P>TRES</P>
7.     </LIST>
8.   </RULE>
9. </GRAMMAR>
  
```

Fonte: Louzada (2010, p. 25).

A API faz a ponte com a *engine* Julius através do construtor, carregando todos os arquivos de configuração necessários. Para tanto, deve-se criar um objeto da classe `SREngine` (Quadro 5 - linha 1) e dar início ao processo de reconhecimento através da chamada ao método `startRecognition` (Quadro 5 – linha 2).

Quadro 5 – Métodos para uso da LapsAPI

```
1. engine = new SREngine(@".\LaPSAM\LaPSAM1.5.jconf");
2. engine.startRecognition();
```

2.3 ROBÔ LEGO NXT

O robô Lego MindStorm também chamado de NXT é um produto da empresa Lego Group (LEGO GROUP, 2013). O *kit* Lego NXT é composto por 619 peças, sendo três motores para movimentação, dois sensores de toque, um sensor de luz e cor, um sensor ultrassônico para detectar presença e distância e um bloco de Unidade Central de Processamento (UCP) de 32 bits. A UCP controla todos os sensores, distribui energia para cada sensor através de pilhas, possui quatro portas de sensores, três portas de servo motor e um *display* (Figura 4).

Figura 4 – Bloco da UCP com todos os motores e sensores conectados



Fonte: Lego Group (2013).

Na Figura 5 é possível visualizar os tipos de sensores: de toque, de luz e cor, ultrassônico (nessa ordem).

Figura 5 – Sensores do Lego NXT



Fonte: Lego Group (2013).

Segundo Pasqualin (2009, p. 12), o sensor de toque detecta quando é pressionado e quando é liberado; o sensor de luz consegue reconhecer entre preto e branco, derivando nos tons de cinza; o sensor ultrassônico emite uma onda sonora, que ao bater em um objeto numa linha reta de até 255 centímetros, capacidade máxima do sensor, devolve o valor em centímetros da distância desse objeto. Todos os sensores devolvem algum valor para a UCP para ser utilizado nos programas.

Além dos sensores, o *kit* possui o servo-motor. Segundo Pasqualin (2009, p. 12), o servo-motor tem como principal função a locomoção do Lego NXT, mas pode ser usado para diversos fins, como por exemplo, um braço robótico. Na Figura 6 verifica-se o servo-motor.

Figura 6 – Servo-motor



Fonte: Lego Group (2013).

Os programas criados para o Lego NXT podem ser enviados do computador para o robô via porta *Universal Serial Bus* (USB) ou por *bluetooth*. Depois de serem gravados na memória do robô, podem ser executados.

O Lego NXT pode ser programado de várias maneiras, sendo uma delas com a linguagem de programação visual NXT-G (PESTANA, 2008, p. 14) que acompanha o *kit* Lego Mindstorms NXT. Outra maneira de programar o Lego NXT é utilizando a API Lego Java *Operating System* (LeJOS). Segundo Pestana (2008, p. 17), LeJOS é tanto é uma API para a programação do Lego NXT na linguagem de programação Java, quanto um *firmware* substituto para o *firmware* original que acompanha a UCP do Lego NXT.

Souza e Tavares (2013, p. 350) afirmam que é possível baixar um pacote do LeJOS, encontrado para as plataformas Windows, Linux e MacOS, contendo:

- a) o *firmware* para substituir o *firmware* original presente no bloco do Lego NXT;
- b) uma biblioteca para auxiliar no desenvolvimento de aplicativos para computador e para o bloco do Lego NXT;
- c) um tradutor para transformar o código Java em uma linguagem compreendida pelo Lego NXT;
- d) vários exemplos de programas e códigos.

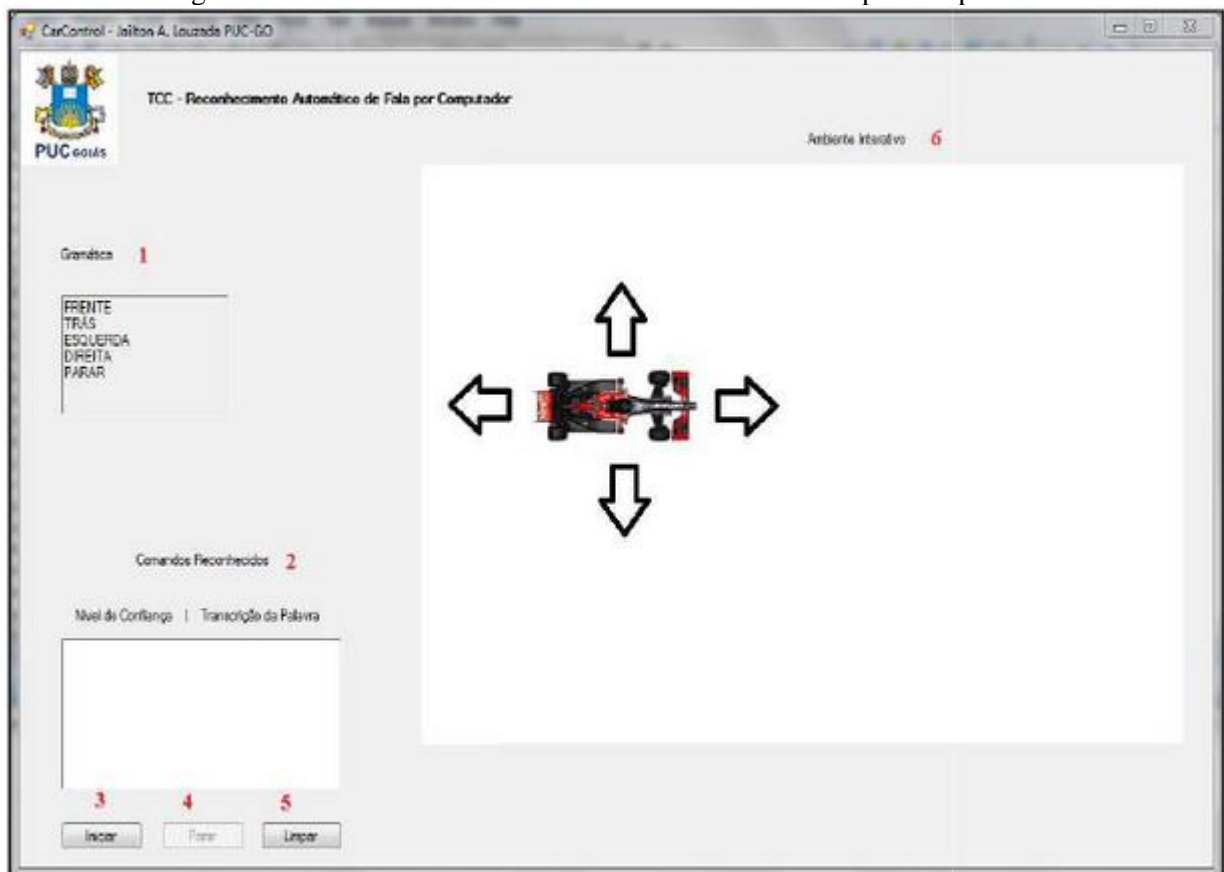
2.4 TRABALHOS CORRELATOS

Nesta seção são descritos os trabalhos de Louzada (2010), Barcelos et al. (2008) e Oliveira (2002), correlatos ao trabalho proposto.

2.4.1 Reconhecimento automático de fala por computador

Louzada (2010) apresenta uma aplicação baseada na *engine* Julius. Utilizando um sinal acústico de um microfone, a aplicação analisa e interpreta o sinal, transcrevendo-o em comandos para controlar a locomoção de um objeto gráfico (um carro). Na Figura 7 é possível observar a aplicação descrita.

Figura 7 – Interface do reconhecimento automático de fala por computador



Fonte: Louzada (2010, p. 32).

A aplicação tem um vocabulário restrito, descrito na gramática do Quadro 6.

Quadro 6 – Gramática da linguagem reconhecida

```

1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <GRAMMAR>
3.    <RULE>
4.      <LIST>
5.        <P>FRENTE</P>
6.        <P>TRAS</P>
7.        <P>DIREITA</P>
8.        <P>ESQUERDA</P>
9.        <P>PARAR</P>
10.     </LIST>
11.   </RULE>
12. </GRAMMAR>

```

Fonte: Louzada (2010, p. 25).

A aplicação é independente de locutor e reconhece somente palavras isoladas e pertencentes à gramática. Um índice de acerto de cerca de 92% foi obtido, tanto para locutores do sexo feminino como para do sexo masculino, mostrando que a *engine* Julius produz resultados satisfatórios. O autor observa que o índice de acerto é essencial para o funcionamento de sistemas RAV. Ainda assim podem ocorrer erros no reconhecimento que influenciam na transcrição (saída do programa) da palavra reconhecida.

2.4.2 Reconhecimento de voz para aplicação em cadeira de rodas

O reconhecimento de voz para aplicação em cadeira de rodas (BARCELOS et al., 2008) é um aplicativo que utiliza o programa IBM ViaVoice para o reconhecimento de fala. Ele acessa funções do IBM ViaVoice para interpretar os comandos de voz imputados pelo usuário. Para a aplicação foi criado um dicionário para delimitar as palavras reconhecidas e tornar a aplicação mais segura para o utilizador. Segundo Barcelos et al. (2008), a delimitação do dicionário ajuda a evitar erros no reconhecimento, o qual prejudicaria o desempenho do sistema.

Para demonstrar a funcionalidade do sistema foi utilizado um *notebook*, que através de uma porta serial comunica-se com o *Peripheral Interface Controller* (PIC) da interface da cadeira de rodas. O *notebook* através de um microfone capta o sinal de voz e processa o comando enviando-o pela porta serial para o micro controlador PIC, que executa o comando fazendo com que a cadeira se movimente.

O sistema tem duas interfaces: uma para entrada do sinal de voz e outra na cadeira de rodas (BARCELOS et al., 2008). A interface para entrada do sinal de voz (Figura 8) foi desenvolvida utilizando a linguagem Visual Basic. Ao processar o sinal de voz, o comando que foi falado é identificado e mostrado na interface.

Já a interface na cadeira de rodas possui um *display* (Figura 9), que apresenta o comando que será executado.

Figura 8 – Interface de reconhecimento de voz em cadeira de rodas



Fonte: Barcelos et al. (2008).

Figura 9 – Interface de execução do comando na cadeira de rodas



Fonte: Barcelos et al. (2008).

Barcelos et al. (2008) cita a utilização do IBM ViaVoice como a melhor alternativa para o processamento de linguagem natural utilizando o português do Brasil, tanto pelo percentual de acerto como pela facilidade de implementação e integração que se obteve utilizando a linguagem de programação Visual Basic.

2.4.3 Reconhecimento de voz através de reconhecimento de padrões

Oliveira (2002) apresenta um sistema RAV, desenvolvido em C++. Baseado em reconhecimento de padrões independente de locutor e em um pequeno vocabulário, o sistema reconhece somente palavras isoladas.

O sistema tem duas fases (OLIVEIRA, 2002, p. 18): uma de treinamento e outra de reconhecimento. Na primeira fase, o locutor treina o sistema com as palavras do vocabulário, que são Word, Access, Excel, VB, C++, Internet e Sair. Na fase seguinte, conforme cada uma destas palavras é reconhecida, o sistema apresenta uma caixa de texto perguntando se o usuário deseja abrir a aplicação correspondente. Caso a resposta seja positiva, a aplicação

correspondente é aberta e o sistema de reconhecimento é encerrado. Caso contrário, o sistema continua em execução. Outra forma de encerrar a execução do sistema é através do reconhecimento da palavra *Sair*.

Oliveira (2002, p. 45) encontrou dificuldades em torno do desenvolvimento do sistema RAV. Entre elas, pode-se citar:

- a) as máquinas utilizadas tinham pouca capacidade de processamento;
- b) nos teste realizados, após a fase de treinamento, os locutores mudavam a maneira de pronunciar a palavra gerando problemas no reconhecimento;
- c) os erros de reconhecimento eram mais numerosos a noite, pois, uma vez que os locutores já haviam utilizado a voz durante todo o dia, estavam com as cordas vocais cansadas, o que alterava um pouco a frequência da voz, diferenciando-a daquela utilizada durante a fase de treinamento.

3 DESENVOLVIMENTO

O capítulo está dividido em seis seções:

- a) mostra os requisitos principais do sistema na primeira seção;
- b) passa pela montagem, especificação e implementação do robô Lego NXT na segunda seção;
- c) apresenta a especificação da linguagem utilizada pelo robô Lego NXT e a sua compilação na seção 3.3;
- d) mostra a especificação e a implementação da aplicação de reconhecimento de voz;
- e) descreve na seção 3.5 a implementação, as ferramentas utilizadas e a operacionalidade da ferramenta;
- f) apresenta, por fim, os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO SISTEMA

O sistema de controle de voz deverá:

- a) possuir uma interface para a captação de voz (Requisito Funcional - RF);
- b) possuir uma gramática para reconhecer os comandos (RF);
- c) gerar comando de controle para o robô Lego NXT (RF);
- d) utilizar a *engine* Julius juntamente com a LapsAPI (Requisito Não Funcional - RNF);
- e) usar um robô Lego NXT para a execução dos comandos (RNF);
- f) transmitir via *bluetooth* os comandos para o robô Lego NXT (RNF);
- g) ter a inteligência do robô Lego NXT implementada na linguagem de programação Java com a *Integrated Development Environment* (IDE) Eclipse (RNF);
- h) ter a interface de captação de voz implementada na linguagem de programação C# com a IDE Visual Studio (RNF).

3.2 ROBÔ LEGO NXT

Esta seção apresenta a montagem do robô Lego NXT, a especificação do mesmo e a sua implementação.

3.2.1 Montagem do robô

O robô Lego NXT foi montado com peças disponibilizadas pela Universidade Regional de Blumenau (FURB), através do Laboratório de Robótica do Departamento de Sistemas Computação.

Para a construção do robô foi utilizado um modelo de robô chamado JohnNXT (BENEDETTELLI, 2008, p. 378-527). Segundo Benedettelli (2008, p. 378), a montagem do JohnNXT leva muito tempo e mais de mil peças são utilizadas para a sua conclusão, precisando de mais do que dois *kits* do Lego NXT. Observa-se que algumas peças não estão contidas no *kit* padrão, como a *black technic link tread* (Figura 10). Como o Laboratório de Robótica também não as tinha em quantidade suficiente, foram adquiridas as peças necessárias para a montagem da esteira que é responsável pela movimentação do robô. Além disso, muitas das peças necessárias para a construção desse modelo não estavam disponíveis no Laboratório de Robótica, sendo necessário adaptar o modelo utilizando outras peças e, dessa forma, descaracterizando o modelo proposto por Benedettelli.

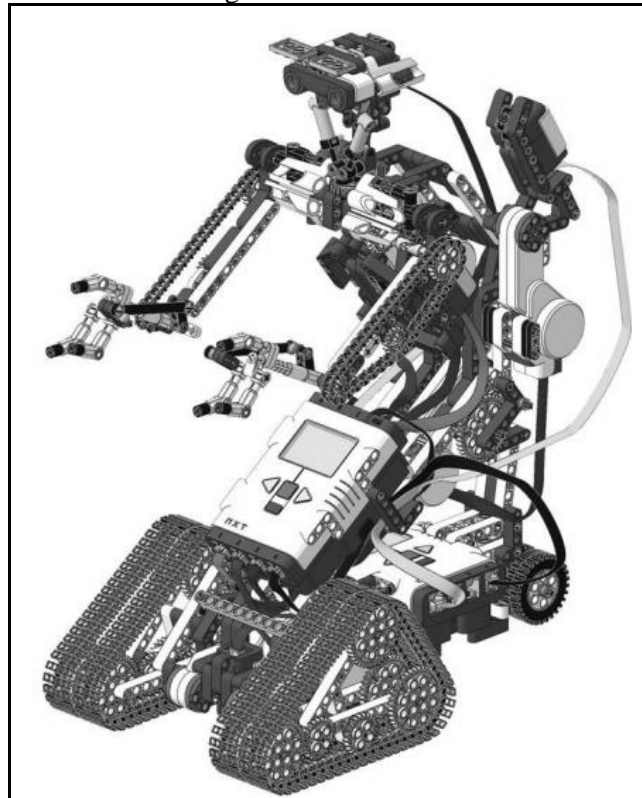
Figura 10 – *Black technic, link tread*



Fonte: Lego Group (2013).

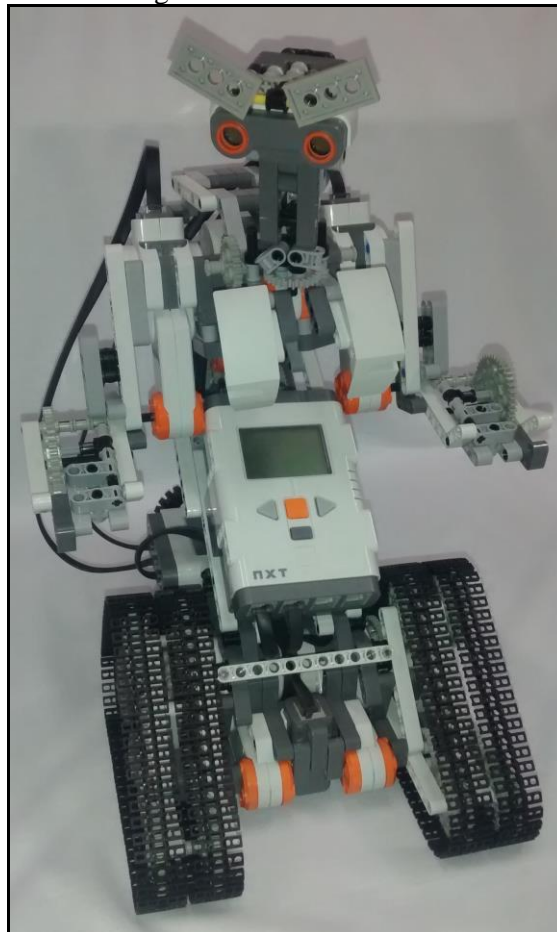
Nas Figuras 11 e 12 tem-se o JohnNXT e o modelo montado. Pode-se reparar, comparando as figuras, que a cabeça, os braços, o tronco e a base do tronco possuem diferenças entre os modelos. No Apêndice A tem-se um quadro com todas as peças necessárias para a montagem do robô proposto.

Figura 11 – JohnNXT



Fonte: Benedettelli (2008, p. 516).

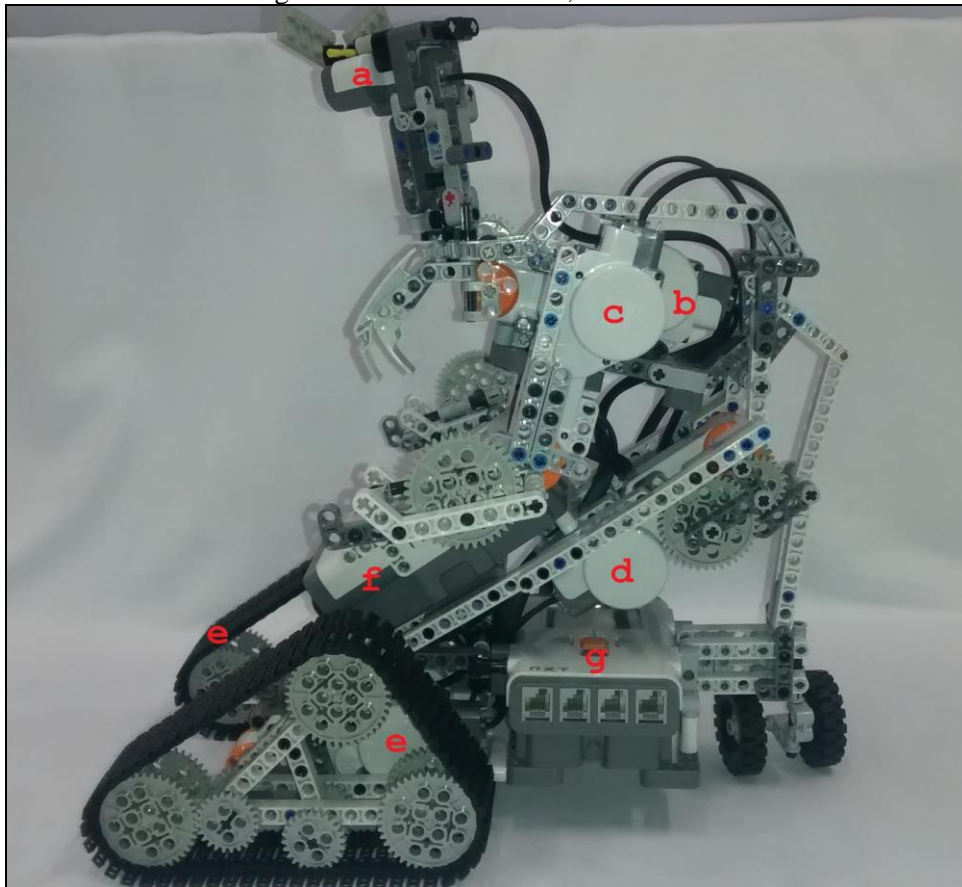
Figura 12 – Robô montado



A Figura 13 traz uma visão lateral do robô, dividindo as partes do corpo, onde observa-se:

- a) a cabeça do robô, na qual foi utilizado o sensor ultrassônico para obter uma expressão mais humanoide;
- b) o servo-motor que movimenta a cabeça;
- c) o braço e o servo-motor que o movimenta;
- d) o servo-motor que movimenta o tronco para cima e para baixo;
- e) as esteiras de locomoção e os servos-motores que as movimentam;
- f) o bloco UCP NXT I;
- g) o bloco UCP NXT II.

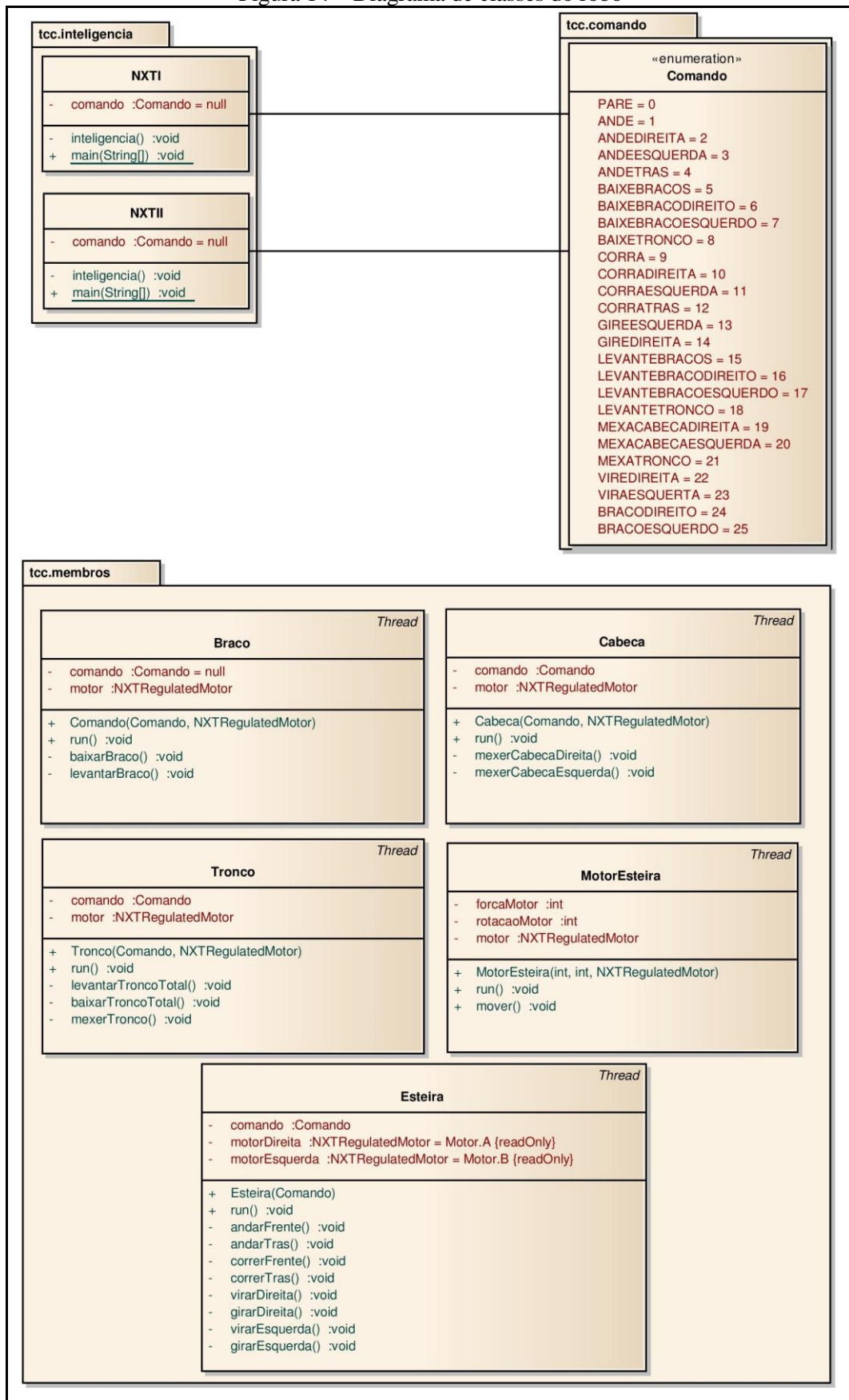
Figura 13 – Robô montado, visão lateral



3.2.2 Especificação da inteligência do robô

Nessa seção são descritas as classes que constituem a inteligência do robô. Pode-se observar pelo diagrama de classes (Figura 14), que a especificação foi dividida em 3 pacotes: `tcc.comando`, `tcc.membros` e `tcc.inteligencia`.

Figura 14 – Diagrama de classes do robô



O pacote `tcc.comando` padroniza os comandos recebidos e executados pelo robô. São 26 comandos representados pela enumeração `Comando`. Já o pacote `tcc.membros` possui as classes que representam os servos-motores, que agem sobre cada membro do robô, tendo uma classe para representar a cabeça, os braços, o tronco e as esteiras de locomoção. A classe `MotorEsteira` é criada a partir da classe `Esteira` e cuida da locomoção do robô, controlando os seus motores. Cada uma das classes possui métodos para movimentar o membro correspondente, sendo que a classe `Braco` tem os comandos para controlar tanto o braço esquerdo como o braço direito. Todas as classes desse pacote possuem uma generalização para a classe `Thread`. Por serem independentes entre si, podem realizar operações em paralelo.

O pacote `tcc.inteligencia` faz a separação física e lógica dos blocos de UCP do robô Lego NXT. Cada classe é responsável por receber um comando de um canal de comunicação e processá-lo, agindo assim nos respectivos membros. No Quadro 7 são apresentados os membros do robô e a classe responsável pelo mesmo.

Quadro 7 – Membros x classes do pacote `tcc.inteligencia`

membro	classe
braço	NXTII
cabeça	NXTI
esteira	NXTI
tronco	NXTII

3.2.3 Implementação da inteligência do robô

Na implementação da inteligência do robô foi utilizada a IDE Eclipse e a API LeJOS.

Inicialmente o robô espera um comando de voz que é transmitido por um canal de comunicação, sendo que neste projeto a comunicação é via *bluetooth*. Pode-se verificar no Quadro 8 (na linha 1) o comando que espera pela conexão *bluetooth*. A requisição só pode ser feita por dispositivos previamente pareados. Ainda no Quadro 8 (na linha 2) é determinado o tipo de conexão (por pacote). Isso significa que o robô espera um pacote contendo o tamanho da mensagem e o outro pacote com o comando propriamente dito. No Quadro 8 (na linha 3) o canal de comunicação é aberto através do método `openDataInputStream`.

Quadro 8 – Conexão e abertura do canal *bluetooth*

```
1. NXTConnection conexao = Bluetooth.waitForConnection();
2. conexao.setIOMode(NXTConnection.PACKET);
3. DataInputStream input = conexao.openDataInputStream();
```

Após a conexão *bluetooth* ser estabelecida e o `DataInputStream` ser criado, o robô aguarda por um comando de voz. As classes `NXTI` e `NXTII` são a inteligência do robô. Cada

uma delas é responsável por uma parte do robô (conforme mostrado no Quadro 7) e faz com que o membro correspondente execute o comando de voz recebido.

O comando é transmitido como um *array* de *bytes*. Verifica-se na linha 3 do Quadro 9 a leitura desse *array*. Sabe-se que o *array* tem nove posições e que a posição que contém o comando é a quinta. Os *bytes* recebidos são percorridos um a um, convertidos para inteiro e armazenados na variável `comandoRecebido`. Na linha 6 tem-se a atribuição à variável `comando` do comando correspondente ao comando recebido, a partir da enumeração `Comando`. Busca-se com isso aumentar a legibilidade do código, associando números aos comandos que representam os movimentos que o robô deve executar. Por fim, na linha 1 do Quadro 9 tem-se a condição de parada. Quando o comando `PARE` for recebido, o robô irá encerrar a execução nos dois blocos de UCP.

Quadro 9 – Processamento do comando de voz recebido

```

1. while (comando != Comando.PARE) {
2.     try {
3.         for (int i = 0; i < 9; i++) {
4.             comandoRecebido = (int) (input.readByte());
5.             if ((i == 4)) {
6.                 comando = Comando.values()[comandoRecebido];
7.             }
8.         }
9.     }
10. }

```

Uma vez definido o `comando` que o robô deve executar (Quadro 9, linha 6), é selecionada a ação correspondente (Quadro 10, linha 1 a 7). Após o `start` da ação, a variável `comando` fica disponível para receber o próximo comando de voz.

Quadro 10 – Seleção de ações do robô

```

1. switch (comando) {
2.     case ANDE:
3.         new Esteira(Comando.ANDE).start();
4.         break;
5.     ...
6.     default: break;
7. }

```

Para a movimentação do robô são utilizados os servos-motores. Basicamente para mover um servo-motor utilizam-se os comandos `backward` e `forward`. Mas para um controle mais eficiente das rotações do robô, optou-se pela utilização do comando `rotate`, além do controle de velocidade dos servos-motores com o comando `setSpeed`. No Quadro 11 verifica-se a utilização desses dois comandos nas linhas 3 e 4. Na linha 2 utiliza-se o comando `waitComplete()`. Esse método espera o servo-motor executar a ação atual e após libera o programa para seguir adiante.

Quadro 11 – Movimentação de um servo-motor

```

1. NXTRegulatedMotor motor = Motor.A;
2. motor.waitComplete();
3. motor.setSpeed(this.forcaMotor);
4. motor.rotate(this.rotacaoMotor);

```

O mesmo princípio é adotado para os outros motores, o que diferencia a movimentação de cada um é a velocidade com que a ação é executada e a quantidade de ângulos que o motor gira.

3.3 LINGUAGEM

Nesta seção é vista a especificação da linguagem usando a notação BNF e sua compilação para ser utilizada pela *engine* Julius.

3.3.1 Especificação da linguagem

A linguagem proposta tem o intuito de fazer o robô locomover-se para quatro direções (direita, esquerda, frente, trás) e em duas velocidades, andando e correndo, além de permitir que ele vire a cabeça para direita ou para esquerda, levante e abaixe os braços, mexa o tronco e execute o comando de parada. O robô realiza suas ações com base nos 26 comandos pré-definidos. Sendo assim, foi especificado um modelo de linguagem, a ser processado pela *engine* Julius, usando uma notação semelhante à BNF. A especificação se dá através dos arquivos `.grammar` (Quadro 12) `.voca` (Quadro 13)

Quadro 12 – Arquivo `.grammar` da linguagem proposta

```

1. INICIO: silencio_inicio COMANDO silencio_fim
2. COMANDO: locomover
3. COMANDO: locomover para direcao
4.
5. COMANDO: mover os membros
6. COMANDO: mover o membro lado
7. COMANDO: mover o tronco
8.
9. COMANDO: membro lado
10. COMANDO: cabeça para ONDE
11.
12. COMANDO: mexer a cabeça para ONDE
13. COMANDO: mexer o tronco
14.
15. COMANDO: virar para ONDE
16.
17. ONDE: direita
18. ONDE: esquerda
19.
20. COMANDO: parar

```

Observa-se no Quadro 12 que o nó não terminal `INICIO` é constituído pela regra `silencio_inicio COMANDO silencio_fim`. As palavras reservadas `silencio_inicio` e `silencio_fim` indicam à *engine* Julius que o `COMANDO` a ser reconhecido deve ter um período sem som, que indica o começo e o final de uma sentença. Os nós não terminais são

representados por palavras com todas as letras maiúsculas e os nós terminais (palavras reservadas e outros *tokens*) com todas as letras minúsculas. Cada nó terminal deve ser representado por uma categoria no arquivo `.voca` (Quadro 13). Uma categoria é representada iniciando-se pelo caractere `%` juntamente com o nome do seu respectivo nó terminal. Cada categoria pode ter mais de uma palavra que pode ser reconhecida.

Quadro 13 – Arquivo `.voca` da linguagem proposta

1.	<code>%silencio_inicio</code>	
2.	<code><s></code>	<code>sil</code>
3.	<code>%silencio_fim</code>	
4.	<code></s></code>	<code>sil</code>
5.	<code>%locomover</code>	
6.	<code>ande</code>	<code>a~ dZ i</code>
7.	<code>corra</code>	<code>k o R a</code>
8.	<code>%mover</code>	
9.	<code>baixe</code>	<code>b a j S i</code>
10.	<code>levante</code>	<code>l e v a~ tS i</code>
11.	<code>%virar</code>	
12.	<code>gire</code>	<code>Z i r i</code>
13.	<code>vire</code>	<code>v i r i</code>
14.	<code>%mexer</code>	
15.	<code>mexa</code>	<code>m e S a</code>
16.	<code>%parar</code>	
17.	<code>pare</code>	<code>p a r i</code>
18.	<code>%cabeca</code>	
19.	<code>cabeça</code>	<code>k a b e s a</code>
20.	<code>%tronco</code>	
21.	<code>tronco</code>	<code>t r o~ k u</code>
22.	<code>%membro</code>	
23.	<code>braço</code>	<code>b r a s u</code>
24.	<code>%membros</code>	
25.	<code>braços</code>	<code>b r a s u s</code>
26.	<code>%direita</code>	
27.	<code>direita</code>	<code>dZ i r e j t a</code>
28.	<code>%esquerda</code>	
29.	<code>esquerda</code>	<code>e s k e R d a</code>
30.	<code>%direcao</code>	
31.	<code>direita</code>	<code>dZ i r e j t a</code>
32.	<code>esquerda</code>	<code>e s k e R d a</code>
33.	<code>trás</code>	<code>t r a j s</code>
34.	<code>frente</code>	<code>f r e~ tS i</code>
35.	<code>%lado</code>	
36.	<code>direito</code>	<code>dZ i r e j t u</code>
37.	<code>esquerdo</code>	<code>e s k e R d u</code>
38.	<code>%a</code>	
39.	<code>a</code>	<code>a</code>
40.	<code>%o</code>	
41.	<code>o</code>	<code>u</code>
42.	<code>%os</code>	
43.	<code>os</code>	<code>u s</code>
44.	<code>%para</code>	
45.	<code>para</code>	<code>p a r a</code>

Observa-se na linha 30 do Quadro 13 a definição do nó terminal `%direcao`. Nessa categoria estão especificadas as palavras que indicam as possíveis direções: `direita`, `esquerda`, `trás` e `frente`. Cada palavra que pode ser reconhecida é acompanhada do seu fonema. A partir da definição do modelo de linguagem, podem ser dados os seguintes comandos para o robô: `mexa a cabeça para direita` ou `cabeça para direita`. Embora

sintaticamente as construções sejam diferentes, a ação do robô será a mesma, mover a cabeça para a direita (MEXACABECADIREITA). As construções do modelo de linguagem definem uma linguagem mais restrita, porém o ganho de performance com o reconhecimento de palavras é maior. Isto é, quanto mais restrito ou elaborado for esse modelo, melhor será o reconhecimento.

3.3.2 Compilação do modelo de linguagem

Com os arquivos `.grammar` e `.voca.` especificados, é necessário compilar o modelo de linguagem para um formato que a *engine* Julius processe durante o reconhecimento dos comandos por voz. Para isso utiliza-se um *script* em Prolog chamado `mkdfa.pl` que está localizado na pasta `bin` da *engine* Julius. O *script* gera no total três arquivos: o `.dfa` é a representação do autômato finito que reconhece as palavras associadas aos nós terminais (do Quadro 13), o `.dict` é um dicionário com as palavras e seus respectivos fonemas e o `.term` são os termos utilizados para a representação dos nós terminais. Na implementação desse trabalho são usados os arquivos `.dict` e `.dfa`.

Na Figura 15 visualiza-se o resultado da compilação dos arquivos `comandos.voca` e `comandos.grammar`. É possível verificar que o modelo de linguagem especificado tem 14 regras gramaticais (14 *rules*) no arquivo `comandos.grammar` e 26 palavras (26 *words*) distribuídas em 19 categorias (19 *categories*) especificadas no arquivo `comandos.voca`. Foram gerados os arquivos `comandos.dfa` (Apêndice B), `comandos.term` (Apêndice C) e `comandos.dict` (Apêndice D).

Figura 15 – Compilação dos arquivos `.grammar` e `.voca`

```
C:\Users\Deivid\Desktop\TCC\Julius\julius-4.3.1-win32bin\bin>mkdfa.pl comandos
comandos.grammar has 14 rules
comandos.voca has 19 categories and 26 words
---
Warning: dfa_minimize not found in the same place as mkdfa.pl
Warning: no minimization performed
Now parsing grammar file
Now modifying grammar to minimize states[1]
Now parsing vocabulary file
Now making nondeterministic finite automaton[17/17]
Now making deterministic finite automaton[17/17]
Now making triplet list[17/17]
---
generated: comandos.dfa comandos.term comandos.dict
```

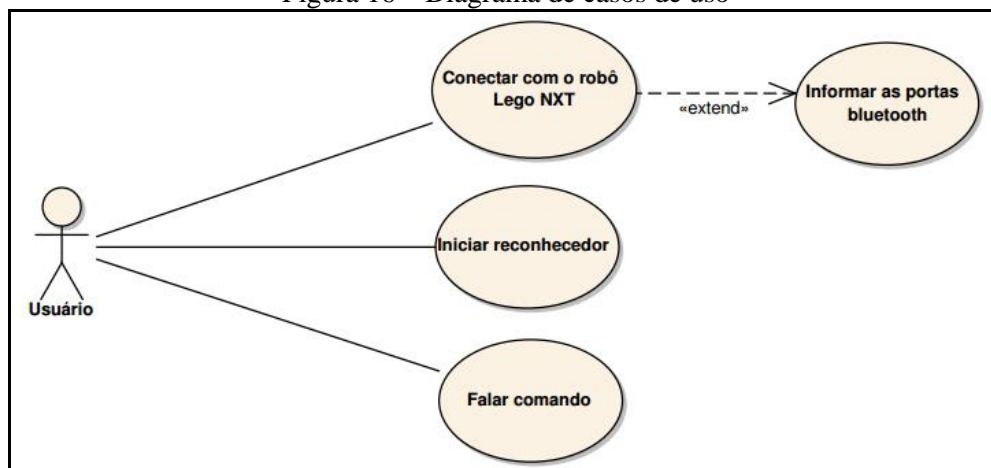
3.4 RECONHECIMENTO DE VOZ

Esta seção traz a especificação da aplicação de reconhecimento de voz, mostrando os casos de uso, os diagramas de classe e de sequência, além da implementação.

3.4.1 Especificação do reconhecimento de voz

A aplicação que faz o reconhecimento de voz possui somente quatro funcionalidades que podem ser executadas pelo usuário, representadas no diagrama de casos de uso da Figura 16.

Figura 16 – Diagrama de casos de uso



O usuário pode: informar as portas para a comunicação *bluetooth* entre o computador e o robô Lego NXT; conectar-se com o robô após informar as portas para comunicação (Quadro 14); iniciar o reconhecimento de voz (Quadro 15); falar comandos que serão processados e enviados para o robô (Quadro 16).

Quadro 14 – Caso de uso: conectar com o robô Lego NXT

UC001 – Conectar com o robô Lego NXT	
Descrição	Conecta-se a um robô Lego NXT nas portas informadas.
Pré-condição	
Atores	Usuário
Fluxo principal	<ol style="list-style-type: none"> 1. O usuário informa as portas de cada UCP do robô Lego NXT. 2. O usuário clica nos botões para efetuar a conexão. 3. O sistema faz a conexão com cada UCP.
Fluxo de exceção	Caso o sistema não consiga efetuar a conexão, apresenta a mensagem “UCP não conectada. Informe a porta correta.”.
Pós-condição	Conexão realizada com sucesso.

Quadro 15 – Caso de uso: iniciar reconhecimento

UC002 – Iniciar reconhecimento.	
Descrição	Inicia o reconhecimento de voz.
Pré-condição	O sistema deve estar conectado com o robô Lego NXT (nas duas UCPs).
Atores	Usuário
Fluxo principal	<ol style="list-style-type: none"> 1. O usuário clica no botão para iniciar o reconhecimento de voz. 2. O sistema inicia o reconhecimento de voz.
Fluxo de exceção	Caso o sistema não esteja conectado com as duas UCPs, apresenta a mensagem “Você deverá conectar com o robô Lego NXT primeiro.”.
Pós-condição	Reconhecimento de voz é iniciado com sucesso. Sistema disponível para entrada de comandos.

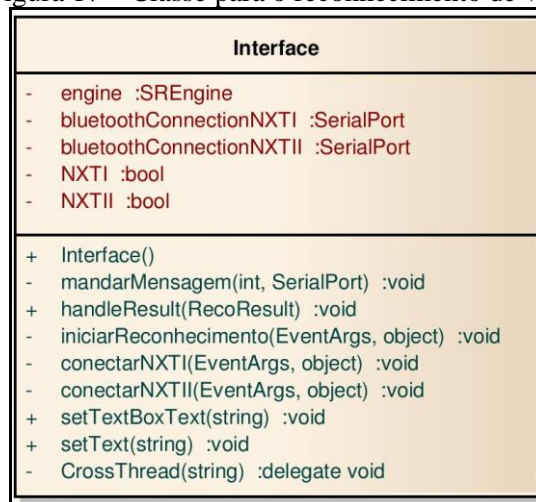
Quadro 16 – Caso de uso: falar comando

UC003 – Conectar com o robô Lego NXT.	
Descrição	Reconhece um comando a ser transmitido para o robô Lego NXT.
Pré-condição	O reconhecimento de voz deve estar iniciado.
Atores	Usuário
Fluxo principal	<ol style="list-style-type: none"> 1. O usuário fala um comando. 2. O sistema reconhece o comando. 3. Se o grau de confiança no reconhecimento do comando for maior que 80% e for um comando reconhecido, o sistema apresenta o comando reconhecido e o seu grau de confiança e envia o comando para o robô Lego NXT.
Fluxo de exceção	Caso o grau de confiança no reconhecimento do comando seja inferior a 80%, o sistema não reconhece o comando falado e apresenta a mensagem “Comando não reconhecido.”.
Pós-condição	Comando reconhecido enviado para o robô. Sistema disponível para reconhecimento do próximo comando.

A aplicação tem apenas uma classe, mostrada na Figura 17, que faz a ponte entre a captação de voz, o processamento de voz e a saída para o robô. Pode-se destacar que:

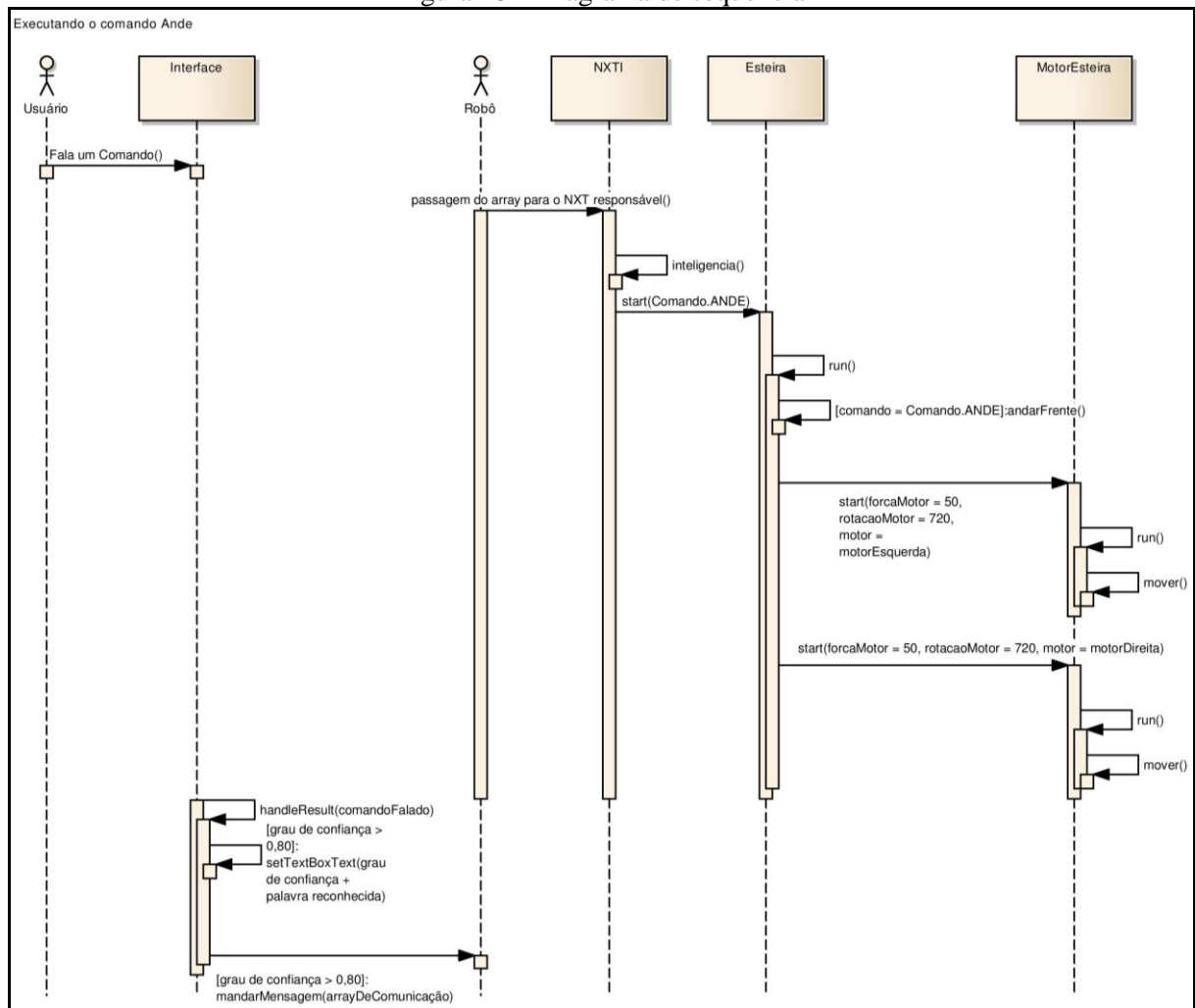
- a) `engine` é a instância da LapsAPI que, por sua vez, encapsula a `engine Julius`;
- b) `bluetoothConnectionsNXTI` e `bluetoothConnectionsNXTII` representam as conexões *bluetooth*;
- c) `NXTI` e `NXTII` são variáveis booleanas que mostram se as UCPs estão conectadas;
- d) o método `mandarMensagem` encaminha uma mensagem via *bluetooth* para o robô Lego NXT com o comando reconhecido pelo método `handleResult`;
- e) o método `handleResult` reconhece o comando falado e faz a classificação da mensagem que deve enviar para o robô Lego NXT;
- f) o método `iniciarReconhecimento` inicializa a `engine Julius`;
- g) os métodos `conectarNXTI` e `conectarNXTII` são responsáveis por estabelecer a conexão com o robô;
- h) o método `setTextBoxText` possui duas funções, fazer conexão entre a *thread* iniciada com a `engine Julius` e os objetos da interface (através do `CrossThread`), exibir o comando reconhecido junto com o seu grau de confiança (através do `setText`).

Figura 17 – Classe para o reconhecimento de voz



Para exemplificar o reconhecimento e o processamento dos comandos, na Figura 18 é apresentado o diagrama de sequência da execução do comando ANDE.

Figura 18 – Diagrama de sequência



O usuário fala um comando no microfone. A interface capta o som e processa-o através do método `handleResult`. Se o grau de confiança no reconhecimento for maior que

0,80, é chamado o `setTextBox`, que apresenta em tela o grau de confiança e a palavra reconhecida. Logo em seguida é disparada uma mensagem através do método `mandarMensagem` que envia para o bloco UCP NXT I do robô o comando reconhecido. O robô recebe o *array* de *bytes* com a mensagem e executa o método `inteligência`, que identifica o comando recebido e processa a ação correspondente, iniciando, no caso, o `Comando.ANDE`. Uma instância de `Esteira` é acionada para processar a ação, criando duas instâncias de `MotorEsteira`, uma para o motor da esquerda e outra para o motor da direita. Essas, por sua vez, processam o comando recebido e fazem com que o robô se mova para frente.

3.4.2 Implementação do reconhecimento de voz

Para a implementação do reconhecimento de voz, foi utilizado como base o código fornecido pelo LaPS (KLAUTAU et al., 2012). Eles fornecem o código do Coruja 0.2, programa que reconhece palavras isoladas e pode ser facilmente adaptado para fala contínua.

Alguns pontos importantes da implementação devem ser destacados. O Quadro 17 traz a inicialização da *engine* Julius, onde: na linha 1 é feita a declaração de `engine`, responsável por receber os parâmetros para iniciar o reconhecimento; na linha 3 tem-se a inicialização propriamente dita com o arquivo de configuração da *engine* Julius, que contém as chamadas para o processamento do modelo acústico e do modelo de linguagem; na linha 4 é chamado o método que faz o reconhecimento de voz.

Quadro 17 – Código de inicialização da *engine* Julius

```
1. private SREngine engine = null;
2. ...
3. engine = new SREngine(@"\LaPSAM\LaPSAM1.5.jconf");
4. engine.startRecognition();
```

A parte mais importante da implementação é a seleção dos códigos dos comandos que serão passados para o robô via *bluetooth*. Pode-se verificar no Quadro 18 na linha 1 que somente são processados os comandos que tenham um grau de confiança no reconhecimento superior a 80%. O comando reconhecido e o grau de confiança são apresentados em tela. Em seguida (linhas 4 a 9), são selecionadas as ações que devem ser enviadas ao robô. Caso o comando reconhecido seja " <s> pare </s>"¹ (Quadro 18, linha 5), são enviadas duas mensagens, ambas com o código 0, que indicam que o robô deve executar o `Comando.PARE`

¹ Todos os comandos reconhecidos possuem as *tags* de início de silêncio e final de silêncio, conforme especificado no modelo de linguagem.

nas duas UCPs, ou seja, que o próximo comando a ser executado é a parada total do robô. O reconhecimento dos outros comandos obedece a mesma lógica.

Quadro 18 – Seleção do comando a ser enviado para o robô

```

1.  if (result.getConfidence() > 0.80) {
2.      setTextBoxText(result.getConfidence()+" | "+result.getUtterance()+"\n");
3.
4.      switch (result.getUtterance()) {
5.          case @" <s> pare </s>": mandarMensagem(blueoothConnectionNXTI, 0);
6.                                  mandarMensagem(blueoothConnectionNXTII, 0);
7.                                  break;
8.          ...
9.          default: break;
10.     }
11. }
```

3.5 IMPLEMENTAÇÃO

Nesta seção são vistas as técnicas utilizadas para a implementação e a operacionalidade da aplicação.

3.5.1 Técnicas e ferramentas utilizadas

As técnicas e ferramentas utilizadas no desenvolvimento desse trabalho foram:

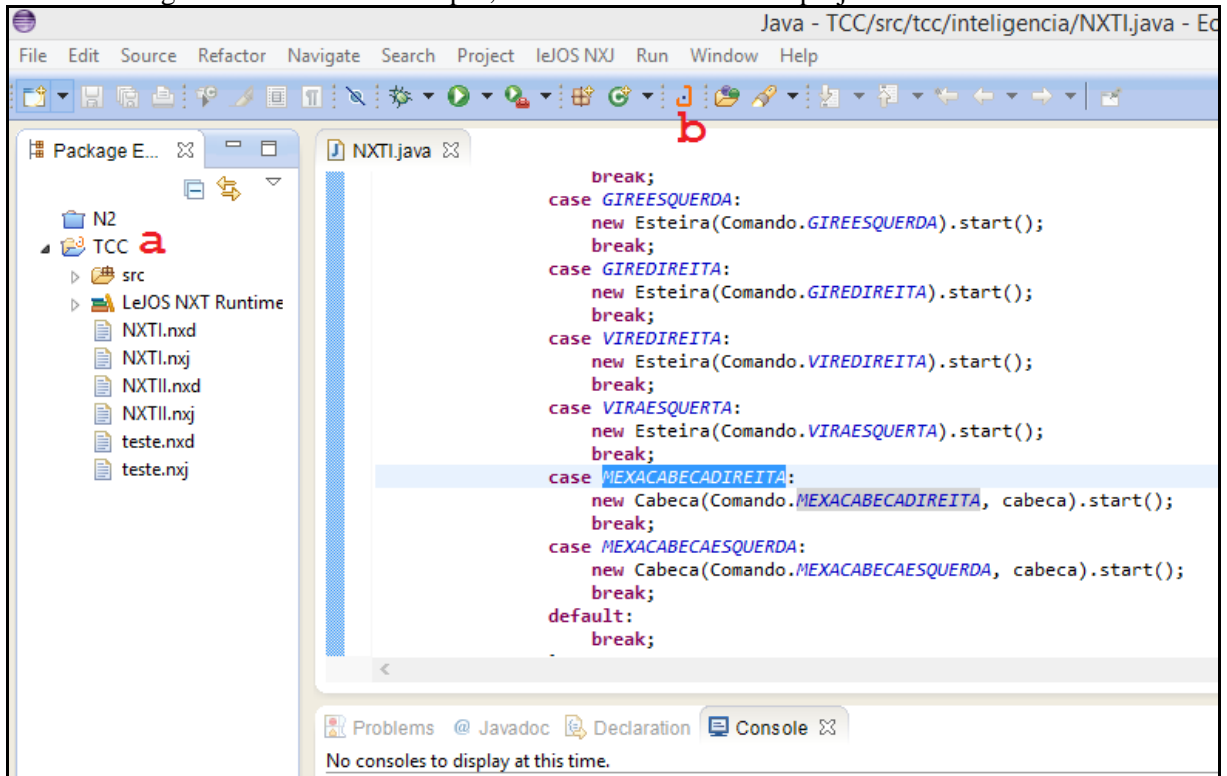
- a) no desenvolvimento dos programas para o Lego NXT foi utilizado o Eclipse juntamente com o *plugin* para o LeJOS;
- b) no desenvolvimento da aplicação de reconhecimento de voz foi utilizado o Visual Studio;
- c) para o reconhecimento de fala contínua foi usada a LapsAPI juntamente com o modelo acústico do idioma LaPSAM 1.5, sendo que para a configuração da *engine* Julius foi utilizado um arquivo fornecido pelo LaPS (LaPSAM1.5.jconf) que pode ser conferido na sua íntegra no Anexo A.

Quanto ao uso do Eclipse juntamente com o *plugin* para o LeJOS, verifica-se, na Figura 19, que o Eclipse tem novos ícones, quais sejam:

- a) a pasta de projeto recebe um “J” no canto direito superior;
- b) a barra de ferramentas tem um botão para o *download* do *firmware* para o Lego NXT.

Observa-se também que LeJOS facilita a programação do Lego NXT, tendo uma classe para cada sensor e algumas classes para o servo-motor. Dessa forma, é possível dar movimento a um servo-motor executando o seguinte comando `Motor.A.rotate(360)` que faz o servo-motor girar 360° no eixo da sua parte móvel.

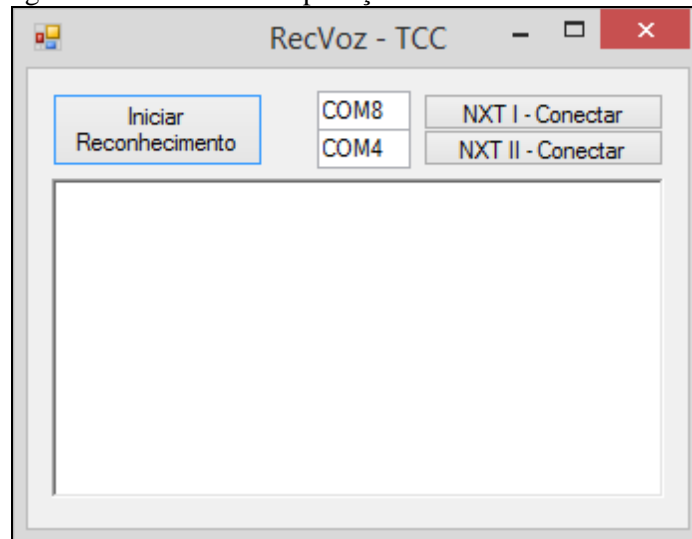
Figura 19 – Ambiente Eclipse, destacando o símbolo de projeto e o botão LeJOS



3.5.2 Operacionalidade da ferramenta

A aplicação desenvolvida possui uma interface simples, apresentada na Figura 20.

Figura 20 – Interface da aplicação de reconhecimento de voz



Após parear as UCPs do Lego NXT com o computador que está rodando a aplicação, o usuário deve informar as portas COMs para cada uma das UCPs do robô Lego NXT nos campos ao lado dos botões **NXT I - Conectar** e **NXT II - Conectar** e pressionar nos botões para que a conexão seja estabelecida. Quando os botões mudarem a descrição para **Conectado**, a conexão foi estabelecida com sucesso. O usuário deve então clicar no botão

Iniciar Reconhecimento. Ao ser apresentada a mensagem Reconhecendo no campo de texto abaixo desse botão, o usuário pode dizer os comandos que quer que o robô realize. Conforme o usuário for falando, a aplicação indica se o comando foi reconhecido ou não. Os comandos reconhecidos são apresentados juntamente com o grau de confiabilidade, enquanto que para os comandos não reconhecidos é apresentada a mensagem Comando não reconhecido. Quando é reconhecido um comando, é gerada uma mensagem com o respectivo comando, que é transferida via *bluetooth* para que o robô, ao recebê-la, execute a ação correspondente.

3.6 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou a especificação de uma linguagem para o controle por voz de um robô Lego NXT, mostrando a capacidade de reconhecimento de palavras e sequências de palavras da *engine* Julius. A aplicação se mostrou funcional, conseguindo captar e processar os sinais acústicos, enfatizando que a *engine* Julius tem um bom funcionamento para o reconhecimento de fala. Para que o reconhecimento fosse mais eficiente, foi necessária a definição de uma linguagem mais restrita, onde os comandos são frases imperativas com no máximo cinco palavras. O ganho no grau de confiança no reconhecimento de uma sequência de palavras foi evidente, embora no trabalho de Louzada (2010) o grau de confiança tenha sido mais alto do que os 80% utilizados nesse trabalho. Isso porque aquele autor reconhecia uma linguagem com palavras isoladas.

Para a obtenção dos 80% utilizados como grau de confiança, foi analisada uma população com dez locutores, sendo composta por seis mulheres entre 6 a 43 anos de idade e quatro homens entre 24 e 57 anos de idade. As vozes foram captadas durante a noite utilizando um microfone externo, que se mostrou mais eficiente que o microfone do *notebook*, calibrado para um volume de 80 pontos através da área de configuração de gravação no sistema operacional Windows. O locutor falava cada comando e o sistema apresentava o grau de confiança. Os resultados foram tabelados para estudo (Tabela 1).

Tabela 1 – Análise de reconhecimento de comandos

comando / locutor	M6	M21	M23	M24	M30	M43	H24	H31-1	H31-2	H57	média	desvio padrão
ande	0,38	1,00	1,00	0,98	0,03	0,88	0,99	1,00	1,00	1,00	0,84	0,34
ande para frente	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,32
ande para direita	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,00
ande para esquerda	1,00	0,83	1,00	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,00
ande para trás	0,97	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00
baixe os braços	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,01
baixe o braço direito	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00
baixe o braço esquerdo	0,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,00	1,00	0,71	0,48
baixe o tronco	0,70	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,42
corra	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,09
corra para frente	0,83	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00
corra para direita	0,00	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,00
corra para esquerda	0,94	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,26
corra para trás	0,82	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,02
gire para esquerda	0,74	0,83	0,83	0,71	0,66	0,83	0,83	0,00	0,83	0,83	0,67	0,26
gire para direita	1,00	0,82	0,83	0,80	0,74	0,68	0,73	0,00	0,00	0,83	0,54	0,33
levante os braços	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00
levante o braço direito	0,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	1,00	1,00	0,86	0,42
levante o braço esquerdo	0,00	0,99	1,00	1,00	1,00	0,00	1,00	0,00	1,00	1,00	0,71	0,48
levante o tronco	0,00	0,98	0,98	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,31
braço direito	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,32
braço esquerdo	0,00	1,00	1,00	1,00	1,00	1,00	1,00	0,99	1,00	1,00	1,00	0,32
mexa a cabeça para direita	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,00
mexa a cabeça para esquerda	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,81	0,90	0,90	0,89	0,03
mexa o tronco	0,98	1,00	1,00	1,00	1,00	0,89	0,93	1,00	1,00	1,00	0,97	0,04
cabeça para direita	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,83	0,67	0,83	0,81	0,05
cabeça para esquerda	0,00	0,83	0,83	0,83	0,83	0,83	0,83	0,67	0,67	0,83	0,79	0,26
pare	1,00	1,00	0,99	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00
vire para direita	0,00	0,83	0,83	0,83	0,83	0,83	0,83	0,72	0,75	0,83	0,80	0,26
vire para esquerda	0,00	0,83	0,83	0,83	0,83	0,83	0,83	0,81	0,83	0,83	0,83	0,26
total média / desvio padrão											0,89	0,13

A Tabela 1 mostra o grau de confiança de cada comando captado para cada locutor. Pode-se verificar que foram obtidos valores fechados em 100% para comandos que foram melhor pronunciados pelos locutores, tais como `ande para direita`, `ande para esquerda`, `ande para trás`, `baixe o braço direito`, `corra para frente`, `corra para direita`, `levante os braços`, `mexa a cabeça para direita` e `pare`. Verifica-se também um grau de confiança muito baixo para os comandos para executar giros (`gire para esquerda`, `gire para direita`), sendo que as médias do grau de confiança chegam no máximo a 67%. Para decidir sobre a utilização de 80% como um bom número para o filtro do grau de confiança, calculou-se a média de todos os comandos que não alcançaram a média de 100% e obteve-se o valor de 80% como média.

O maior agravante no reconhecimento de voz foi o ruído gerado, tanto pelo microfone como pelo ambiente. Como a *engine* Julius não consegue filtrar todo o ruído gerado, foram reconhecidas várias sentenças com grau de confiança abaixo dos 80% filtrados pela aplicação ou diferentes do que foi de fato falado. No último caso, muitas vezes resultava na execução do comando `PARE`, ou seja, na parada total do robô. Além disso, ao iniciar o reconhecimento de voz, percebeu-se que, independente do que seja falado, o primeiro comando reconhecido sempre é `ANDE`. Essa situação ainda não foi solucionada.

A montagem do robô foi complicada, principalmente por não estarem disponíveis no Laboratório de Robótica muitas das peças necessárias, dificultando e atrasando a finalização do mesmo. Esteticamente, constatou-se que o robô pende levemente para a esquerda, mas isso não parece influenciar nos movimentos realizados. Uma dificuldade evidente é a locomoção do robô. As peças de plástico facilitam o deslizamento do mesmo em diversas superfícies, fazendo com que o robô execute os comandos de locomoção de forma a não parecerem completos. Observou-se que quanto mais larga a esteira utilizada, melhor se torna a locomoção.

Quanto aos trabalhos correlatos, o Quadro 19 traz uma comparação entre eles e o trabalho desenvolvido.

Quadro 19 – Quadro comparativo entre os trabalhos correlatos e trabalho desenvolvido

Característica/ferramenta	LOUZADA (2010)	BARCELOS et al. (2008)	OLIVEIRA (2002)	trabalho desenvolvido
reconhecimento usando <i>engine</i> Julius	X			X
reconhecimento usando IBM ViaVoice		X		
reconhecimento nativo			X	
necessita de treinamento			X	
independente de locutor	X		X	X
interage com meio físico		X		X
linguagem restrita	X	X	X	X
reconhecimento de fala contínua				X

Verificou-se que:

- a) o trabalho de Louzada (2010) usa a *engine* Julius para reconhecer palavras isoladas, portanto é independente de locutor, mas processa uma linguagem composta apenas por quatro palavras;
- b) o trabalho de Barcelos et al. (2008) utiliza no reconhecimento de palavras o IBM ViaVoice. Reconhece sentenças compostas por duas palavras, sendo seu vocabulário pequeno e é restrito, evitando assim erros no reconhecimento. Tem como propósito a movimentação de uma cadeira de rodas. Portanto, é único dos três trabalhos correlatos a interagir com o meio físico como o trabalho desenvolvido;
- c) diferente dos outros trabalhos, Oliveira (2002) foi o único que implementou o reconhecimento nativo, pois não utilizou nenhuma *engine*. Nesse caso, necessitou de uma fase de treinamento e reconhece somente palavras isoladas em um vocabulário restrito. Os problemas identificados por Oliveira (2002) não são vistos nos trabalhos que utilizaram a *engine* Julius, já que esses não possuem uma fase de treinamento para efetuar o reconhecimento de voz.

4 CONCLUSÕES

Cada vez mais as pessoas realizam atividades utilizando aparelhos eletrônicos. Assim, são necessárias maneiras alternativas para manipulá-los, sendo o controle por voz uma solução viável e bastante desejada atualmente.

Neste sentido, este trabalho apresentou o desenvolvimento de uma aplicação para efetuar controle por voz com fala contínua de um robô Lego NXT. Para tanto, foi utilizada a *engine* Julius, que dá suporte para sistemas RAV, facilitando a implementação de soluções de reconhecimento e controle por voz. A gramática elaborada através do modelo de linguagem facilitou o reconhecimento. O modelo acústico utilizado foi o corpora criado pelo LaPS (KLAUTAU et al., 2012), que se mostrou adequado para a aplicação desenvolvida, não sendo necessária uma fase de treinamento. Foi definido um vocabulário independente de locutor e com o reconhecimento de sequências de palavras e não apenas palavras isoladas, permitindo alcançar os objetivos propostos.

A utilização da *engine* Julius juntamente com a API LapSAPI foram de grande valia para a conclusão desse trabalho, mostrando que a especificação e o reconhecimento por voz de uma linguagem, mesmo que restrita, é feita de forma simples. Ainda, as demais ferramentas utilizadas ajudaram para a finalização desse trabalho, sendo o Lego Digital Designer (LEGO GROUP, 2013) usado na especificação do esquema de montagem do robô, uma adaptação do JohnNXT proposto por Benedittelli (2008), e a biblioteca do LeJOS usada para a programação do Lego NXT.

A análise de dados se mostrou importante, uma para justificar o grau de confiança de 80% utilizado como filtro e outra para mostrar que a fonte de entrada de voz é muito importante, visto que nos primeiros testes desse trabalho se usou um microfone interno (embutido no notebook) e depois se optou por uma versão externa e um pouco mais robusta, aumentando assim o grau de confiança das sentenças.

4.1 EXTENSÕES

Existem pontos que podem ser melhorados ou incrementados, originando trabalhos futuros, quais sejam:

- a) aumentar o vocabulário reconhecido pelo robô, permitindo a passagem de parâmetros para representar velocidade, total de passos ou distância pretendidos, como por exemplo “ande na velocidade 3 por 150 centímetros”;
- b) utilizar o sensor ultrassônico do Lego NXT como um sensor de aproximação, sendo possível o robô detectar obstáculos à frente e parar o movimento dos

- motores de locomoção antes que uma colisão ocorra;
- c) utilizar os sensores de cor e luz para propor desafios para o robô, fazendo com que ele procure em um determinado ambiente um sinal ou comando;
 - d) verificar a possibilidade de utilizar o microfone do *kit* Lego NXT para a entrada de voz e utilizar a LapsAPI embutida na inteligência do robô, não sendo necessário ter uma aplicação *desktop* para a transmissão dos comandos via *bluetooth*;
 - e) utilizar a LapsAPI juntamente com um robô feito com o *kit* de desenvolvimento Arduino (ARDUINO, 2014), possibilitando assim a utilização de uma gama maior de sensores;
 - f) criar uma linguagem com comandos para a realização de diversos movimentos de locomoção e outros movimentos como o das articulações do corpo, não somente para robôs com esteiras, mas para outros tipos de robôs, como bípedes e rastejadores;
 - g) utilizar reconhecimento de voz para a criação de um ambiente para o aprendizado de programação de robôs para crianças, utilizando o lado lúdico do brincar juntamente com a facilidade da fala.

REFERÊNCIAS

ARDUINO. **What Arduino can do**. [S.l.], 2014. Disponível em: <<http://arduino.cc/>>. Acesso em: 22 jun. 2014.

BARCELOS, Arlei et al. Reconhecimento de voz para aplicação em cadeira de rodas. In: SIMPÓSIO DE EXCELÊNCIA EM GESTÃO E TECNOLOGIA, 5., 2008, Resende. **Anais eletrônicos...** Resende: AEDB, 2008. Não paginado. Disponível em: <http://www.aedb.br/seget/artigos08/445_Reconhecimento%20de%20Voz%20aplicado%20na%20cadeira%20de%20rodas.pdf>. Acesso em: 29 mar. 2013.

BENEDETTELLI, Daniele. **Creating cool Mindstorms NXT robots**. Berkeley: Apress, 2008.

DAVIS, K. H.; BIDULPH, R.; BALASHEK, S. Automatic recognition of spoken digits. **The Journal of the Acoustical Society of America**, [S.l.], v. 24, n. 6. p. 637-642, Nov. 1952. Disponível em: <<http://www.idemployee.id.tue.nl/g.w.m.rauterberg/presentations/bell-labs.pdf>>. Acesso em: 05 maio 2014.

KLATT, Denis H. Review of the ARPA speech understanding project. In: WAIBEL, Alexander; LEE, Kai-Fu (Ed.). **Readings in speech recognition**. San Mateo: Morgan Kaufmann Publishers, 1990. p. 554-575. Disponível em: <http://books.google.com.br/books?id=iDHgboYRzmgC&pg=PA554&lpg=PA554&dq=ARPA+Speech+Understanding+Project&source=bl&ots=jaeSCVTlgK&sig=d2JaO5Ksg9z0x6GmCBW0gimf76U&hl=pt-BR&sa=X&ei=6_VrU6LAArC-sQS-ooD4CQ&ved=0CFMQ6AEwBA#v=onepage&q=ARPA%20Speech%20Understanding%20Project&f=false>. Acesso em: 08 maio 2014.

KLAUTAU, Aldebaro et al. **Reconhecimento de voz para o português brasileiro**. [S.l.], 2012. Disponível em: <<http://www.laps.ufpa.br/falabrasil/downloads.php>>. Acesso em: 04 abr. 2013.

LEE, Akinobu; KAWAHARA, Tatsuya. Recent development of open-source speech recognition engine Julius. In: ASIA-PACIFIC SIGNAL AND INFORMATION PROCESSING ASSOCIATION ANNUAL SUMMIT AND CONFERENCES, 1., 2009, Hokkaido. **Proceedings...** Hokkaido: APSIPA, 2009. p. 131-137. Disponível em: <<http://www.ar.media.kyoto-u.ac.jp/EN/bib/intl/LEE-APSIPA09.pdf>>. Acesso em: 29 mar. 2013.

LEE, Akinobu. **The Julius book**. [S.l.], 2009. Disponível em: <<http://osdn.dl.sourceforge.jp/julius/37581/Juliusbook-part-4.1.2-en.pdf>>. Acesso em: 14 abr. 2013.

LEGO GROUP. **Lego Mindstorms NXT 2.0**. [S.l.], 2013. Disponível em: <<http://shop.lego.com/en-US/LEGO-MINDSTORMS-NXT-2-0-8547>>. Acesso em: 20 abr. 2013.

LOUZADA, Jailton A. **Reconhecimento automático de fala por computador**. 2010. 52 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Computação, Pontifícia Universidade Católica de Goiás, Goiás. Disponível em: <http://www.jailtonalkiminlouzada.com/downloads/ASR_Jailton_Alkimin_Louzada.pdf>. Acesso em: 29 mar. 2013.

NUANCE COMMUNICATION. **Dragon NaturallySpeaking 12**: stop typing and start talking. [S.l.], 2013. Disponível em: <<http://www.nuance.com/dragon/whats-new-version-12/index.htm>>. Acesso em: 20 abr. 2013.

OLIVEIRA, Karla M. **Reconhecimento de voz através de reconhecimento de padrões**. 2002. 93 f. Projeto Final de Curso (Bacharelado em Informática) – Departamento de Informática, Universidade Católica do Salvador, Salvador. Disponível em: <info.ucsal.br/banmon/Arquivos/Mono_140707.doc>. Acesso em: 01 maio 2013.

PASQUALIN, Douglas P. **Tratamento de falhas em robô Lego Mindstorms com arquitetura reativa**. 2009. 48 f. Trabalho Final de Graduação (Bacharelado em Sistemas de Informação) – Centro Universitário Franciscano, Santa Maria. Disponível em: <<http://www-usr.inf.ufsm.br/~dpasqualin/artigos/TFG.vFinal.revisado.pdf>>. Acesso em: 21 maio 2014.

PESTANA, Hélder G. **DROIDE M.L.P**: NXT software development kit. 2008. 67 f. Dissertação (Mestrado em Engenharia Informática) – Departamento de Matemática e Engenharias, Universidade da Madeira, Funchal. Disponível em: <http://dme.uma.pt/projects/droide/portal/DROIDE_MLP_NXT_Software_Development_Kit.pdf>. Acesso em: 20 maio 2014.

SILVA, Carlos P. A. **Um software de reconhecimento de voz para o português brasileiro**. 2010. 74 f. Dissertação (Mestrado em Engenharia Elétrica) – Instituto de Tecnologia, Universidade Federal do Pará, Belém. Disponível em: <http://www.repositorio.ufpa.br/jspui/bitstream/2011/2074/1/Dissertacao_SoftwareReconhecimentoVoz.pdf>. Acesso em: 29 mar. 2014.

SOUZA, Nilton M.; TAVARES, Dalton M. Estudo de caso usando o framework LeJOS. In: ENCONTRO ANUAL DE COMPUTAÇÃO, 10., 2013, Catalão. **Anais...** Catalão: UFG, 2013. p. 349-356. Disponível em: <<http://www.aptor.com.br/enacomp2013/pdf/48.pdf>>. Acesso em: 26 maio 2014.

VIEIRA, Renata; LIMA, Vera L. S. **Linguística computacional**: princípios e aplicações. [S.l.], 2001. Disponível em: <<http://www.inf.unioeste.br/~jorge/MESTRADOS/LETRAS%20-%20MECANISMOS%20DO%20FUNCIONAMENTO%20DA%20LINGUAGEM%20-%20PROCESSAMENTO%20DA%20LINGUAGEM%20NATURAL/ARTIGOS%20INTERESSANTES/lingu%EDstica%20computacional.pdf>>. Acesso em: 28 maio 2013.

YNOGUTI, Carlos A. **Reconhecimento de fala contínua usando modelos ocultos de Markov**. 1999. 138 f. Tese (Doutorado em Engenharia Elétrica) – Curso de Pós-Graduação em Engenharia Elétrica, Universidade Estadual de Campinas, Campinas. Disponível em: <<http://www.bibliotecadigital.unicamp.br/document/?code=vtls000188821>>. Acesso em: 29 mar. 2013.






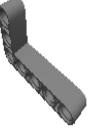




APÊNDICE A – Quadro das peças para a montagem do robô

O Quadro 20 relaciona as peças necessárias para a montagem do robô. Para cada peça, tem-se o código, o nome, a imagem e a quantidade necessária. São necessárias um total de 841 peças. Observa-se que, diferente do modelo proposto por Benedettelli (2008), os braços foram simplificados com um servo-motor para cada braço, não sendo preciso colocar correntes para movimentar os mesmos, fazendo com que a quantidade de peças necessárias seja menor.

Quadro 20 – Peças

Brick	Name	Picture	Quantity
4133542	PLATE 2X4		2
255526	PLATE 1X1 W. UP RIGHT HOLDER		2
6047885	CHAINS M PLATE M 5 ø3, 2 HOLE		250
348224	SPOKED HUG D17MM		2
363426	TRACTOR TYRE Ø17/Ø43		2
4546542	COLOUR SENSOR, ASSEMBLED		1
4297174	Ultrasound sensor		1

4297008	Tacho Motor		6
4296825	NXT		2
4177444	TECHNIC 2M BEAM		2
4211862	TECHNIC 2M BEAM		1
4210751	TECHNIC 3M BEAM		23
4210686	TECHNIC 5M BEAM		10
4211651	TECHNIC 5M BEAM		18
4297199	TECHNIC 7M BEAM		11
4210667	TECHNIC ANG. BEAM 4X2 90 DEG		22
4297202	TECHNIC 9M BEAM		13

4611705	TECHNIC 11M BEAM		1
4297200	TECHNIC 11M BEAM		3
3252501	TECHNIC 11M BEAM		1
4297203	TECHNIC 13M BEAM		3
4542578	TECHNIC 15M BEAM		6
4210753	TECHNIC ANG. BEAM 3X5 90 DEG.		10
4210656	TECHNIC ANGULAR BEAM 4X4		9
4211668	TECHNIC ANGULAR BEAM 4X4		4
4537417	TECHNIC ANGULAR BEAM 3X7		6
4210668	DOUBLE ANGULAR BEAM 3X7 45°		2

4234240	DOUBLE ANGULAR BEAM 3X7 45°		2
4169492	TECHNIC LEVER 3M		2
290526	TRIANGLE		1
4211573	1/2 BUSH		15
4142865	2M CROSS AXLE W. GROOVE		4
4121715	CONNECTOR PEG W. FRICTION		129
4211815	CROSS AXLE 3M		6
4666579	CONNECTOR PEG/CROSS AXLE		4
4225927	CONNECTOR PEG/CROSS AXLE		29
4211622	BUSH FOR CROSS AXLE		40

370526	CROSS AXLE 4M		9
655826	CONNECTOR PEG W. FRICTION 3M		5
4514553	CONNECTOR PEG W. FRICTION 3M		33
4514554	3M CONNECTOR PEG		1
4107081	CATCH W. CROSS HOLE		5
4114740	CROSS AXLE 5M		1
4211639	CROSS AXLE 5M		17
4211775	CROSS BLOCK 90°		2
370626	CROSS AXLE 6M		10
4107742	2M FRIC. SNAP W/CROSS HOLE		2

4211805	CROSS AXLE 7M		6
4121667	DOUBLE CROSS BLOCK		5
370726	CROSS AXLE 8M		5
4107783	ANGLE ELEMENT, 180 DEGREES [2]		1
4211779	CROSS BLOCK 3M		16
4535768	CROSS AXLE 9M		3
373726	CROSS AXLE 10M		2
4211629	TECHNIC CROSS BLOCK 2X1		3
4211889	TECHNIC CROSS BLOCK/FORK 2X2		7
4119589	MODULE BUSH		7

4225033	BEAM 3 M. W/4 SNAPS		15
4296059	Angular beam 90degr. w.4 snaps		2
4211432	GEAR WHEEL T=8, M=1		7
471626	WORM		1
4211565	GEAR WHEEL Z24		12
4285634	GEAR WHEEL 40T		16
6015596	RIGHT SCREEN Ø 4.85 4X7X4		1
6015597	LEFT SCREEN Ø 4.85 4X7X4		1
4514192	OUTER CABLE 64MM, VIRTUEL		1
Total:			841

APÊNDICE B – Arquivo comandos.dfa

O arquivo `comandos.dfa` (Quadro 21) é o autômato finito gerado a partir da compilação dos arquivos `comandos.grammar` e `comandos.voca`.

Quadro 21 – Arquivo `comandos.dfa`

1.	0	1	1	0	1
2.	1	2	2	0	2
3.	1	6	2	0	2
4.	1	8	3	0	2
5.	1	9	4	0	2
6.	1	10	5	0	2
7.	1	11	6	0	6
8.	1	12	6	0	6
9.	1	13	7	0	2
10.	1	14	8	0	2
11.	2	0	9	2	0
12.	3	16	10	0	0
13.	4	16	11	0	0
14.	5	17	12	0	0
15.	6	18	13	4	0
16.	7	18	14	0	0
17.	8	9	15	0	0
18.	9	-1	-1	1	0
19.	10	3	2	0	0
20.	10	5	2	0	0
21.	11	3	2	0	0
22.	12	3	2	0	0
23.	13	4	2	0	0
24.	13	7	16	0	0
25.	14	2	2	0	0
26.	15	0	9	2	0
27.	15	16	17	2	0
28.	16	0	9	2	0
29.	16	15	18	2	0
30.	17	3	2	0	0
31.	18	5	2	0	0

APÊNDICE C – Arquivo `comandos.term`

O Quadro 22 apresenta o arquivo `comandos.term`, que contém as categorias reconhecidas pelo *script* `mkdfa.pl`.

Quadro 22 – Arquivo `comandos.term`

0	<code>silencio_inicio</code>
1	<code>silencio_fim</code>
2	<code>locomover</code>
3	<code>mover</code>
4	<code>virar</code>
5	<code>mexer</code>
6	<code>parar</code>
7	<code>cabeca</code>
8	<code>tronco</code>
9	<code>membro</code>
10	<code>membros</code>
11	<code>direita</code>
12	<code>esquerda</code>
13	<code>direcao</code>
14	<code>lado</code>
15	<code>a</code>
16	<code>o</code>
17	<code>os</code>
18	<code>para</code>

APÊNDICE D – Arquivo `comandos.dict`

O Quadro 23 apresenta o `comandos.dict`. Nesse arquivo cada linha contém, para cada palavra, a categoria, a palavra reconhecida e os fonemas da palavra.

Quadro 23 – Arquivo `comandos.dict`

0	[<s>]	sil
1	[</s>]	sil
2	[ande]	a~ dZ i
2	[corra]	k o R a
3	[baixe]	b a j S i
3	[levante]	l e v a~ tS i
4	[gire]	Z i r i
4	[vire]	v i r i
5	[mexa]	m e S a
6	[pare]	p a r i
7	[cabeça]	k a b e s a
8	[tronco]	t r o~ k u
9	[braço]	b r a s u
10	[braços]	b r a s u s
11	[direita]	dZ i r e j t a
12	[esquerda]	e s k e R d a
13	[direita]	dZ i r e j t a
13	[esquerda]	e s k e R d a
13	[trás]	t r a j s
13	[frente]	f r e~ tS i
14	[direito]	dZ i r e j t u
14	[esquerdo]	e s k e R d u
15	[a]	a
16	[o]	u
17	[os]	u s
18	[para]	p a r a

ANEXO A – Arquivo de configuração da *engine* Julius

No Quadro 24 visualiza-se o arquivo utilizado como configuração da *engine* Julius. As linhas 22 e 23 contém a definição do modelo acústico, que são disponibilizados pelo LaPS (KLAUTAU et al. , 2012), enquanto as linha 26 e 27 contém a definição do modelo de linguagem, isso é, os arquivos gerados pela compilação dos arquivos `comandos.grammar` e `comandos.voca`.

Quadro 24 – Arquivo de configuração da *engine* Julius

```

1. #####
2. # Julius Conf File
3. #####
4.
5.
6. #-lv 3000          # level threshold (0-32767)
7. #-zc 150           # zero-cross threshold (times in sec.)
8. #-headmargin 800  # head silence margin (msec)
9. #-tailmargin 900  # tail silence margin (msec)
10. #-rejectshort 50   # reject shorter input (msec)
11.
12. #-realtime         # force real-time processing
13. #-norealtime       # force non real-time processing
14.
15. #### Acoustic HMM file ####
16. ## support ascii hmmdefs or binary format (converted by "mkbinhmm")
17.
18. # Hmm model
19. # Caminho para o modelo acústico.
20. -h "./LaPSAM1.5.am.bin"
21. # triphone model needs HMMList that maps logical triphone to physical ones.
22. -hlist "./tiedlist1.5.txt"
23. -htkconf "./LaPSAM/edaz.conf"
24.
25. #Gramatica
26. -dfa ../Gramatica/comandos.dfa
27. -v ../Gramatica/comandos.dict
28.
29. #-multipath
30. #-spsegment
31.
32. # Specify short pause model name to be treated as special
33. -spmmodel "sp"          # HMM model name
34. #-iwspword
35.
36.
37. # Context-dependency handling will be enabled according to the model type.
38. # Try below if julius wrongly detect the type of hmmdefs
39. #
40. #-no_ccd              # disable context-dependency handling
41. -force_ccd            # enable context-dependency handling
42.
43. ##### binary LM
44. #-d LM/3gram.bigram
45. #-d /home/04080002801/LMTools/SRILM/lms/juliusLM5gram
46.
47.
48. # 1st LM
49. #-nlr /home/04080002801/lvcsr1.1/LM/HLMToolkit/lms/bigram_HLM dictionary.lm

```

```

50.
51. # 2nd LM
52. #-nrl
53. /home/04080002801/lvcsr1.1/LM/HLMToolkit/lms/bigram_HLM_dictionary.rev.lm
54.
55. # Dictionary
56. #-v ../dic.temp
57.
58. # LM weights and word insertion penalties
59. -lmp 15.0 10.0
60. -lmp2 15.0 10.0
61.
62. # Envelope beam width (number of hypothesis) - deixa o decoder bem mais lento
63. !!
64. -b 2000
65. -b2 200
66. -sb 300
67.
68. # The number of candidates Julius tries to find.
69. #-n 5
70.
71. ## If julius go wrong with checking parameter type, try below.
72. ##
73. #-notypecheck
74. #
75.
76. ## (PTM/triphone) switch computation method of IWCD on 1st pass
77. ##
78. -iwcd1 best 5 # assign average of N-best likelihood of the same context
79. #-iwcd1 max # assign maximum likelihood of the same context
80. #-iwcd1 avg # assign average likelihood of the same context (default)
81.
82. #-cvn
83.
84.
85. #### Gaussian Pruning ####
86. ## Number of mixtures to select in a mixture pdf.
87. ## This default value is optimized for IPA99's PTM,
88. ## with 64 Gaussians per codebook
89. #-tmix 1
90.
91. ## Select Gaussian pruning algorithm
92. ## defulat: beam (standard setting), safe (others)
93. -gprune safe # safe pruning, accurate but slow
94. #-gprune heuristic # heuristic pruning
95. #-gprune beam # beam pruning, fast but sensitive
96. #-gprune none # no pruning
97.
98.
99. #### Search Parameters ####
100.
101. #-b 400 # beam width on 1st pass (#nodes) for monophone
102. #-b 800 # beam width on 1st pass (#nodes) for triphone,PTM
103. #-b 1000 # beam width on 1st pass (#nodes) for
104. triphone,PTM,engine=v2.1
105. #-s 50 # Stack size
106. #-b2 200 # beam width on 2nd pass (#words)
107. #-sb 200.0 # score beam envelope threshold
108. #-s 500 # hypotheses stack size on 2nd pass (#hypo)
109. #-m 2000 # hypotheses overflow threshold (#hypo)
110. #-lookuprange 5 # lookup range for word expansion (#frame)
111. #-n 1 # num of sentences to find (#sentence)
112. #-n 10 # (default for 'standard' configuration)
113. #-output 1 # num of found sentences to output (#sentence)
114. #-looktrellis # search within only backtrellis words
115.
116.
117. ## For insertion of context-free short-term inter-word pauses between words

```

```

118. ## (multi-path version only)
119. ##
120. -iwsf # append a skippable sp model at all word ends
121. #-iwsfpenalty -5.0 # transition penalty for the appenede sp models
122.
123.
124. #####
125. #### Speech Input Source
126. #####
127. ## select one (default: mfcfile)
128. #-input mfcfile # MFCC file in HTK parameter file format
129. #-input rawfile # raw wavefile (auto-detect format)
130. # WAV(16bit) or
131. # RAW(16bit(signed short),mono,big-endian)
132. # AIFF,AU (with libsndfile extension)
133. # other than 16kHz, sampling rate should be specified
134. # by "-smpFreq" option
135. #-input mic # direct microphone input
136. # device name can be specified via env. val.
137. "AUDIODEV"
138. #-input netaudio -NA host:0 # direct input from DatLink(NetAudio) host
139. #-input adinnet -adport portnum # via adinnet network client
140. #-input stdin # from standard tty input (pipe)
141.
142. #-filelist filename # specify file list to be recognized in batch mode
143.
144. #-nostrip # switch OFF dropping of invalid input segment.
145. # (default: strip off invalid segment (0 sequence
146. etc.)
147. -zmean # enable DC offset removal (invalid for mfcfile
148. input)
149. -zmeanframe
150.
151. #####
152. #### Acoustic Analysis
153. #####
154. -smpFreq 22050 # sampling rate (Hz)
155. #-smpPeriod 227 # sampling period (ns) (= 10000000 / smpFreq)
156. #-fsize 400 # window size (samples)
157. #-fshift 160 # frame shift (samples)
158. #-delwin 2 # delta window (frames)
159. #-hifreq -1 # cut-off hi frequency (Hz) (-1: disable)
160. #-lofreq -1 # cut-off low frequency (Hz) (-1: disable)
161. #-cmnsave filename # save CMN param to file (update per input)
162. #-cmnload filename # load initial CMN param from file on startup

```