ática2.38

Universidade do Minho

Projeto
Interpretador de comandos de voz para controlo de veículos motorizados

Paulo Alexandre Azevedo Ferreira
(A74816)

Vítor Hugo Gonçalves Dias
(A76531)

Orientadores: José João Almeida e Vitor Fonte

5 de Julho de 2017

Conteúdo

0.1 Agradecimentos

A elaboração deste projeto não teria sido possível sem a ajuda, estímulo e motivação de diversas pessoas. Gostaríamos, por isso, de expressar a nossa gratidão e apreço por todos aqueles, que de forma direta ou indireta, contribuíram para a concretização deste trabalho.
Primeiramente, queríamos endereçar o nosso sincero agradecimento aos professores José João Almeida e Vitor Fonte, orientadores do projeto, por todo o acompanhamento desenvolvido ao longo destes meses de trabalho. Para além do acompanhamento, salientamos também a partilha de conhecimento/ideias por parte destes, facto este que nos ajudou a seguir um bom caminho na realização deste projeto.
Por fim, queremos deixar uma palavra de agradecimento ao nosso colega Ándre Santos, que apesar de não fazer parte dos orientadores iniciais do projeto, mostrou-se sempre disponível e interessado em ajudar-nos (desde o primeiro momento em que foi abordado) e foi por isso imprescindível para a concretização de um dos objetivos a que nos propusemos, controlar um objeto físico (neste caso um robot) com comandos de voz.

Capítulo 1

Introdução

O processo de reconhecimento de voz tem sofrido grandes desenvolvimentos ao longo das últimas décadas. Se antes este tópico era apenas visto como mera ficção científica, com os avanços tecnológicos e investigação desenvolvida nesta área, atualmente é uma realidade que está presente no nosso dia-a-dia e com relativa facilidade de acesso.

O presente relatório é referente ao projeto desenvolvido no âmbito da unidade curricular Projeto. O projeto consiste numa aplicação capaz de interpretar comandos de voz para controlar vários tipos de objetos sejam eles físicos ou virtuais, desde drones, carros, etc. O relatório contém uma introdução teórica ao tema de reconhecimento de voz, explicação do funcionamento das ferramentas utilizadas, explicação da comunicação entre o reconhecedor de voz e a simulação/robot.

Capítulo 2

Reconhecimento de Voz

2.1 Conceitos básicos

Uma abordagem ingénua poderá levar-nos a pensar que o reconhecimento de voz consiste palavras e que cada palavra, por sua vez, consiste num fonema. Porém, a realidade é bastante diferente, uma vez que o reconhecimento de voz é um processo dinâmico no qual não é de todo possível distinguir as suas componentes de uma forma sistemática. As abordagens mais recentes, remetem-nos para uma ideia de probabilidade associada a este conceito pois não existem separações certas entre unidades ou palavras, o que leva a certas dificuldades por parte dos desenvolvedores de software, uma vez que estes estão diariamente a trabalhar com sistemas determinísticos, o que não é o caso do reconhecimento por voz.

2.2 Estrutura

O reconhecimento de voz é visto como uma "stream"continua de áudio, onde existem elementos estáticos e dinâmicos, destes elementos um pode definir uma classe de som um pouco similar, porém as propriedades acústicas de um fonema podem variar consoante vários fatores, como por exemplo, o contexto do fonema, pessoa que o pronuncia, estilo do discurso, entre outros. Por causa destes fatores que afetam o reconhecimento, a co-articulação faz com que os fonemas soem de maneira muito diferente da sua forma "canónica". Uma vez que as transições entre palavras são mais informativas do que as regiões estáticas, é comum falar-se de difonemas - partes de fonemas entre dois fonemas consecutivos ou trifonemas.

2.3 Ferramentas utilizadas

Para a realização do projeto foi utilizada a biblioteca *Pocketsphinx* da *CMU Sphinx* e *Python* para a manipulação da mesma. Antes da escolha recair para

as ferramentas mencionadas foram levadas em consideração outras ferramentas, para o reconhecimento de voz consideraram-se as seguintes:

- Cloud Speech API (Google)
- Bing Speech API (Microsoft)
- *Kaldi*
- *CMU Sphinx*

As primeiras hipóteses a serem retiradas foram a utilização da *Cloud Speech API* e *Bing Speech API* pelo facto de estas serem pagas e para além disso online. Depois, entre *Kaldi* e *CMU Sphinx* a escolha recaiu sobre a *CMU Sphinx* devido ao facto de esta ter interface de utilização em várias linguagens e por isso o seu uso ser mais facilitado. Agora, dentro da *CMU Sphinx* temos duas bibliotecas a *Pocketsphinx* e a *Sphinx4*, no site é recomendado que se use a *Pocketsphinx* caso o objetivo seja a rapidez do reconhecimento e a portabilidade, como tal, acabamos por escolher a *Pocketsphinx*.

Escolhida a ferramenta de reconhecimento de voz falta agora escolher a linguagem de programação a ser utilizada, como referido anteriormente a biblioteca escolhida tem suporte em várias linguagens de programação. A nossa escolha inicial recaiu sobre *Java* devido à maior familiaridade com a linguagem, mas depois devido à extensão do código decidimos utilizar *Python* por causa da praticidade da linguagem.

2.4 Funcionamento do algoritmo de reconhecimento de voz

As palavras são constituídas por fonemas, porém estes nunca são iguais pois variam de acordo com a voz, pronuncia e a articulação entre fonemas faz com que um fonema seja diferente da sua representação canónica, etc. Na onda sonora de um fonema, encontram-se facilmente três partes que permitem identificar tal fonema, essas três partes são chamadas de trifonemas, onde a 1ª depende do fonema precedente, a 2ª parte é estável e a 3ª parte depende do fonema subsequente.

Por motivos computacionais é mais útil detetar partes dos trifonemas do que detetar os trifonemas em si, por exemplo, criar um detetor de som para o inicio de um trifonema e utilizar esse mesmo detetor noutro trifonema, esse detetor de som é chamado de *senone*.

O algoritmo utiliza a onda de som formada pela fala e divide a mesma em *utterances* [1], de seguida tenta-se reconhecer o que é dito em cada *Utterance*. Para tal são verificadas todas as combinações possíveis de palavras até que se encontre a palavra mais idêntica.

[1] *Utterances* é uma unidade de som que começa e termina com intervalos de silêncio bem claros, isto é, palavras ou *fillers*, sons que não constituem uma palavra, como por exemplo, tossir, espirrar, etc.

O processo de se encontrar as palavras mais idênticas ao som é feito através de números, primeiro, divide-se a fala em *frames* com cerca de 10 milissegundos e em cada frame é formado o *feature vector* [2], a forma como estes números são obtidos é um caso de estudo muito complexo, como tal não falaremos dele aqui, porém, de uma forma resumida são obtidos através do espetro sonoro.

2.5 Elementos necessários para o reconhecimento

Para o reconhecimento de voz ser bem sucedido são necessários três elementos fundamentais:

- Modelo acústico - contém todos os fonemas possíveis e o *feature vector* mais provável para cada fonema.
- Dicionário - Contém as palavras que devem ser traduzidas e cada palavra tem uma sequência de fonemas associada, todas as palavras aqui presentes devem estar também presentes no modelo de linguagem, caso contrário esta não é reconhecida.

```
AND     AH N D
AND(2)  AE N D
FORWARD F AO R W ER D
LEFT    L EH F T
LISTENING       L IH S AH N IH NG
LISTENING(2)    L IH S N IH NG
MOVE    M UW V
```

Figura 2.1: Estrutura de um dicionário

- Modelo de linguagem - Dentro do modelo de linguagem existem essencialmente três tipos diferentes: *Keyword list*, modelo de linguagem estatístico e gramática. Cada uma deles tem vantagens e desvantagens quando comparados com outros, pelo que deve ser escolhido aquele que mais se ajuste à funcionalidade necessária.
  - Modelo de linguagem estatístico : Serve para restringir as palavras a ser reconhecidas, define as palavras mais prováveis a aparecer depois de já ter reconhecido alguma palavra e retira as palavras menos prováveis de aparecer o que melhora significativamente os testes, principalmente em modelos maiores.
  - *Keyphrase lists*: É um substituto do Modelo de linguagem e não pode ser utilizado em simultâneo com este, tal como o anterior, serve também para limitar o número de palavras a serem reconhecidas, com a vantagem de apenas reconhecer as frases presentes enquanto que no Modelo acústico mesmo que uma palavra que não esteja presente seja dita, a palavra mais próxima será reconhecida mesmo que esta

[2] Sequência dos números que representam a fala

```
\data\
ngram 1=9
ngram 2=11
ngram 3=8

\1-grams:
-0.9031 </s> -0.3010
-0.9031 <s> -0.2430
-1.5051 FORWARD -0.2430
-1.5051 LEFT -0.2430
-1.5051 MOVE -0.2872
-1.5051 MOVING -0.2430
-1.5051 RIGHT -0.2430
-1.2041 ROTATE -0.2730
-1.5051 STOP -0.2872

\2-grams:
-0.9031 <s> MOVE 0.0000
-0.6021 <s> ROTATE 0.0000
-0.9031 <s> STOP 0.0000
-0.3010 FORWARD </s> -0.3010
-0.3010 LEFT </s> -0.3010
-0.3010 MOVE FORWARD 0.0000
-0.3010 MOVING </s> -0.3010
-0.3010 RIGHT </s> -0.3010
-0.6021 ROTATE LEFT 0.0000
-0.6021 ROTATE RIGHT 0.0000
-0.3010 STOP MOVING 0.0000

\3-grams:
-0.3010 <s> MOVE FORWARD
-0.6021 <s> ROTATE LEFT
-0.6021 <s> ROTATE RIGHT
-0.3010 <s> STOP MOVING
-0.3010 MOVE FORWARD </s>
-0.3010 ROTATE LEFT </s>
-0.3010 ROTATE RIGHT </s>
-0.3010 STOP MOVING </s>
```

Figura 2.2: Estrutura de um *Statistical Language Model*

não tenha nada a ver com a palavra dita. As *Keyphrase lists* contêm as frases, ou palavras que devem ser reconhecidas e cada uma destas tem um *threshold* associado, *threshold* é um valor no formato /1e-x/ com $x \in [0, 50]$, sendo que quanto maior for o número de sílabas da frase, maior terá de ser o x, é recomendado que as frases ou palavras tenham um mínimo de 3 sílabas e um máximo de 10 sílabas com vista a tornar o reconhecimento mais eficaz.

– Gramática : Geralmente representam um tipo simples de linguagem e contrariamente a outros tipos de modelos, estas não contém qualquer tipo de probabilidade associada. As gramáticas permitem definir *inputs* com precisão, por exemplo, se uma palavra tiver se der repetida duas ou três vezes. Porém, esta capacidade pode ser problemática caso o utilizador se esqueça de alguma palavra que a gramática necessite, pois deste modo o reconhecimento irá falhar.

```
#JSGF V1.0;

grammar hello;
public <greet> = (good morning | hello) ( bhiksha | evandro | rita | will );
```

Figura 2.3: Estrutura de uma gramática

Estes elementos podem ser gerados através da ferramenta *LMTool*, para isso, basta criar um ficheiro com o nome "corpus.txt"e esse ficheiro deve conter todas as frases que que devem ser reconhecidas.

2.6 Conceitos relacionados à temática

- *Lattice* - É um grafo orientado que representa variantes do reconhecimento. Por vezes, obter o melhor *match* não é o mais prático, nesse caso as *Lattices* são bons formatos para apresentar o resultado do reconhecimento.
- *N-Best-Lists* - São idênticas às *lattices* porém não apresentam tanta informação como as mesmas.
- Velocidade - Supondo que um ficheiro de áudio tem 2 horas e a descodificação do mesmo demora 6 horas, então a velocidade é 3 x RT(*Recording time*).
- Curvas Roc - É um gráfico que representa a relação entre o número de tentativas e o número de falsos alarmes e tenta encontrar o ponto ótimo, ou seja, o ponto onde o número de falsos alarmes é mínimo.

2.7 Idioma do reconhecedor

O idioma padrão da ferramenta é o Inglês, foi considerada a hipótese de reconhecer em Português porém, por motivos mais à frente mencionados, decidimos não implementar a língua Portuguesa.

2.8 Suporte para um novo idioma

Tal como referido anteriormente foi considerada a hipótese de adicionar o português ao nosso modelo, tal é possível com os seguintes passos:

- Recolha de dados áudio, em português, com algumas horas de duração, o recomendado seria "5 hour of recordings of 200 speakers for command and control for many speakers"

- Treino do modelo, é recomendado que se treine o modelo caso se tenha mais do que 1 mês para o fazer, pois trata-se de um processo muito demorado e extenso.
- Testes

Pelo facto de se tratar de um processo muito demorado decidimos utilizar apenas a língua Inglesa pois a mesma já se encontra preparada para ser utilizada.

Capítulo 3

Desenvolvimento do programa

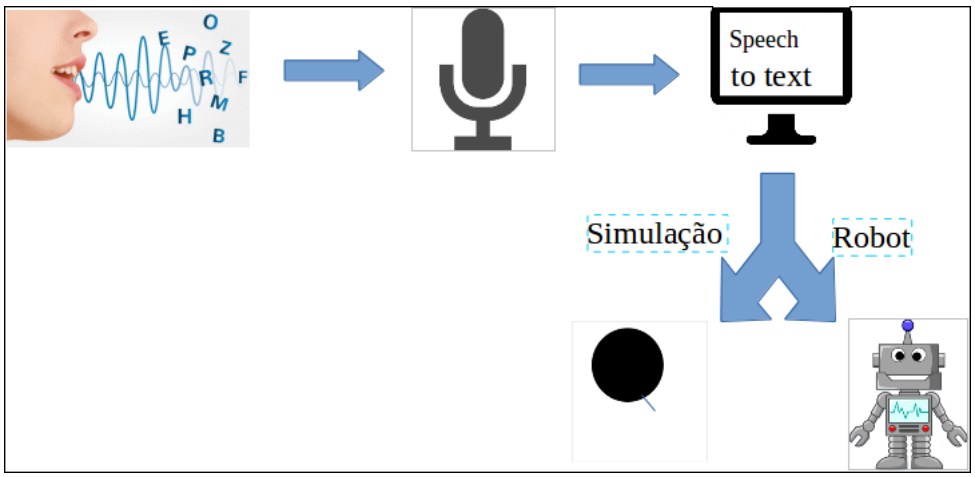

Figura 3.1: Esquema geral do sistema

Este projeto é baseado numa arquitetura cliente-servidor, o cliente é responsável pela transformação do som em texto e enviar, através de um socket, as instruções para o Robot ou Simulação(figura 3.1) .

3.1 Cliente

Ao desenvolver o cliente, inicialmente, optamos por utilizar Java por ser uma linguagem mais prática em relação ao C na manipulação de strings, pois o objetivo era transformar o áudio em texto que mais tarde seria manipulado,

mas depois de alguma pesquisa reparamos que a biblioteca do *Pocketsphinx* tinha, também, uma interface de utilização na linguagem *Python*, que é ainda mais prática na manipulação de strings e mais sucinta relativamente ao C e ao Java, logo, decidimos utilizar *Python*.

O próximo passo seria transformar a voz em texto, fizemos várias tentativas até chegar ao resultado pretendido.

Inicialmente, começamos por utilizar o *Pocketsphinx* com o dicionário e o modelo de linguagem padrão e apesar de traduzir a voz para texto, nem sempre a tradução era feita da melhor forma, como tal precisávamos de melhorar a eficácia com que essa tradução era feita. Como o nosso modelo contém poucas palavras, decidimos fazer um dicionário e um modelo de linguagem, utilizando a ferramenta *lmtool*, adaptado ao nosso modelo contendo apenas as frases necessárias, com esta adição ao projeto a melhoria na eficácia do reconhecimento foi notória, uma vez que as palavras possíveis seriam apenas 4 ou 5, mas outro problema surgiu, quando uma palavra que não estava presente nem no dicionário nem no modelo era dita, o *Pocketsphinx* tentava encontrar a palavra, presente no modelo, mais parecida com a referida palavra e tal comportamento não era desejável. Por último, com vista a resolver o caso da rejeição de palavras fora do modelo, decidimos utilizar *Keyword lists*, tal como mencionado anteriormente é adequado para restringir as palavras a serem reconhecidas e tem a vantagem de conseguir recusar as palavras que não estão presentes. O formato das *Keyword lists* é o seguinte:

```
frase /1e-10/
exemplo /1e-15/
outra frase /1e-30/
```

Para calcular os *thresholds* mais adequados, foi utilizado o seguinte comando:

```
pocketsphinx_continuous -infile file.wav
-keyphrase keyphrase -kws_threshold threshold -time yes
```

O file.wav é um ficheiro no formato 16 bit 16k Hz, com cerca de 1 hora de duração, contendo algumas ocorrências da palavra desejada. A *keyphrase* será cada palavra ou frase, individualmente presentes na *Keyword list* e o *threshold* é o *threshold* associado. Este comando, sempre que detetar o *keyphrase* no áudio, irá imprimir a frase e o tempo do áudio que encontrou a frase, por isso, devem ser tentados vários *thresholds* até chegar aquele que apresenta menos falhas.

Neste momento o programa já é capaz de transformar som em texto, a próxima fase passa por processar o texto e enviar os comandos respetivos. Por sugestão de um dos orientadores decidimos adicionar um ficheiro de configuração de forma a tornar o programa mais genérico e não apenas voltado para um robot em específico, como tal, o ficheiro de configuração tem a seguinte estrutura:

```
[Files]
Dictionary: path
Kws: path
```

```
Logfn: path

[Commands]
comando = mensagem a enviar
```

O programa precisa de 3 ficheiros para funcionar: o dicionário, a *keyword list* e o *logfn* que é o ficheiro onde as informações do reconhecimento são guardadas, como as definições do micro, definições do reconhecedor, etc. O comando deve ser a frase a ser reconhecida com as palavras que a constituem presente no dicionário e com uma mensagem a enviar, ao Robot ou à Simulação, atribuída.

O ficheiro de configuração deverá ter o nome "config.txt".

3.1.1 Algoritmo utilizado para o envio de mensagens

Começamos por ler os comandos e as suas respetivas mensagens do ficheiro de configuração, guardando-as num dicionário, depois quando existe algum de tipo de mensagem sonora, verifica-se se o texto traduzido está no dicionário e de seguida envia-se a mensagem associada ao comando através de um *socket*.

3.2 Servidor

O servidor é o objeto que irá ser controlado, neste projeto temos um Robot físico e uma simulação feita em *Processing/Java*.

Depois de feita a conexão entre o cliente e o servidor, este espera por uma mensagem com o conteúdo *talker* vinda do cliente, de seguida, o servidor está pronto a receber as mensagens

3.2.1 Simulação

A simulação é uma bola que recebe vários comandos.

3.2.1.1 Comandos

- Move Forward - Esta mensagem ordena que a bola se mova em frente e envia a mensagem "f 0.01"
- Rotate Right - Ordena que a bola rode sobre si no sentido negativo, a mensagem é "r -0.7"
- Rotate Left - Ordena que a bola rode sobre si no sentido positivo, a mensagem é "r 0.7"
- Stop Moving - Ao receber esta mensagem a bola para completamente qualquer tipo de movimento, a mensagem é "0"
- Save Point - Este comando guarda o local atual da bola, a mensagem é "save point"

- Go To Location - É o comando que ordena a bola a ir ao encontro do ponto guardado com o comando "Save Point", a mensagem é "go to point".

```
[Files]
Dictionary: /home/paulo/PycharmProjects/VoiceRecognition/Data/Simulacao/Simulacao.dict
Kws: /home/paulo/PycharmProjects/VoiceRecognition/Data/Simulacao/Simulacao_Keyphrase.list
Logfn: /home/paulo/PycharmProjects/VoiceRecognition/Data/Simulacao/logfn.log

[Commands]
ROTATE RIGHT = r -0.01
ROTATE LEFT = r 0.01
MOVE FORWARD = f 0.7
STOP MOVING = 0
SAVE POINT = save point
GO TO LOCATION = go to point
```

Figura 3.2: Ficheiro de configuração da simulação

```
ROTATE RIGHT/1e-40/
ROTATE LEFT/1e-40/
MOVE FORWARD/1e-50/
STOP MOVING/1e-10/
STOP LISTENING/1e-20/
SAVE POINT/1e-20/
GO TO LOCATION/1e-25/
```

Figura 3.3: *Keyphrase List* da simulação

```
FORWARD F AO R W ER D
GO  G OW
LEFT    L EH F T
LISTENING   L IH S AH N IH NG
LISTENING(2)    L IH S N IH NG
LOCATION    L OW K EY SH AH N
MOVE    M UW V
MOVING  M UW V IH NG
POINT   P OY N T
RIGHT   R AY T
ROTATE  R OW T EY T
SAVE    S EY V
STOP    S T AA P
TO  T UW
TO(2)   T IH
TO(3)   T AH
```

Figura 3.4: Dicionário da simulação

3.2.1.2 Algoritmo

Como referido anteriormente, a simulação foi desenvolvida utilizando *Processing*.

Foram desenvolvidas 3 classes, uma classe "Ball", que contém todas as informações da bola, como velocidade linear, velocidade angular, coordenadas, etc. Nesta classe é utilizado um Lock de forma a poder fazer controlo de concorrência entre as outras classes, uma vez que uma escreve e a outra lê.

A outra classe, "Servidor", é uma *Thread* que recebe as mensagens através do *Socket* e, sempre que recebe uma mensagem, de acordo com o conteúdo da mesma coloca as informações na classe "Ball".

Por último a classe principal, "Simulacao", é a classe responsável, por ler os dados necessários e desenhar a bola na sua posição correta, esta classe segue as normas do Processing, onde deve existir uma função "setup"e uma "draw", a função "setup"corre apenas uma vez no inicio do programa e a "draw"corre em loop.

3.2.2 Robot

Para controlar o Robot, gentilmente cedido pelo nosso colega André Santos, é utilizado um servidor já desenvolvido pelo mesmo, esse servidor lê comandos e de seguida envia os comandos ao Robot de forma a que este se mova, o Robot é controlado através de ROS *Robot Operating System*. O servidor irá receber mensagens do tipo "vel_linear vel_angular", sendo que vel_linear tem um máximo de $0.25m/s$ e a vel_angular um máximo de $1.25r/s$.

3.2.2.1 Comandos

- Move Forward - Esta mensagem ordena que o robot se mova em frente e envia a mensagem "0.1 0"
- Rotate Right - Ordena que o robot rode sobre si no sentido negativo, a mensagem é "0 -0.7"
- Rotate Left - Ordena que o robot rode sobre si no sentido positivo, a mensagem é "0 0.7"
- Stop Moving - Ao receber esta mensagem o Robot para completamente qualquer tipo de movimento, a mensagem é "0 0"
- Walk Left - Este comando faz com que o robot se mova em frente e rode sobre si ao mesmo tempo, a mensagem é "0.1 0.7".
- Walk Right - Tal como o comando anterior, este faz o mesmo, porém o robot roda sobre si noutro sentido, logo a mensagem é "0.1 -0.7".

```
[Files]
Dictionary: /home/paulo/PycharmProjects/VoiceRecognition/Data/Robot/Robot.dic
Kws: /home/paulo/PycharmProjects/VoiceRecognition/Data/Robot/Robot_Keyphrase.list
Logfn: /home/paulo/PycharmProjects/VoiceRecognition/Data/Robot/logfn.log

[Commands]
ROTATE RIGHT = 0 -0.7
ROTATE LEFT = 0 0.7
MOVE FORWARD = 0.1 0
STOP MOVING = 0 0
WALK LEFT = 0.1 0.7
WALK RIGHT = 0.1 -0.7
```

Figura 3.5: Ficheiro de configuração do Robot

```
ROTATE RIGHT/1e-40/
ROTATE LEFT/1e-40/
MOVE FORWARD/1e-50/
STOP MOVING/1e-10/
STOP LISTENING/1e-20/
WALK LEFT/1e-15/
WALK RIGHT/1e-15/
```

Figura 3.6: *Keyphrase List* do Robot

```
AND AH N D
AND(2)  AE N D
FORWARD F AO R W ER D
LEFT    L EH F T
LISTENING   L IH S AH N IH NG
LISTENING(2)    L IH S N IH NG
MOVE    M UW V
MOVING  M UW V IH NG
RIGHT   R AY T
ROTATE  R OW T EY T
STOP    S T AA P
WALK    W AO K
WALK(2) W AA K
```

Figura 3.7: Dicionário do Robot

Capítulo 4

Conclusão

Com a realização deste projeto foi possível ficar a conhecer, ainda que de forma muito simples, aquilo que é hoje uma realidade cada vez mais desenvolvida, que é o reconhecimento por voz, quer seja para controlar um veículo (caso deste projeto) ou simplemesnte para enviar um email. Desta forma ficamos a conhecer as principais ferramentas utilizadas neste tipo de projetos, bem como certos elementos comuns a todas elas, tais como os dicionários, modelos acústicos, entre outros.

Os objetivos principais do projeto foram atingidos de forma satisfatória, porém ao longo destes meses de trabalho, o grupo deparou-se com algumas dificuldades que teve de ultrapassar, entre a quais destacamos a dificuldade com a implementação correta da *keywordlist*, essencial para que apenas certas palavras posssam ser reconhecidas pela ferramenta.

Futuramente, pensamos que projetos envolvendo esta temática serão construtivos e de valor em termos de aprendizagem, uma vez que esta é, como já foi dito, uma área em larga expansão.

Bibliografia

[1] CMU Sphinx website
https://cmusphinx.github.io/wiki/tutorial/

[2] Documentação *pocketsphinx* em Python
https://pypi.python.org/pypi/pocketsphinx

[3] Fórum *CMU Sphinx*
https://sourceforge.net/p/cmusphinx/discussion/

[4] https://student.dei.uc.pt/ guilhoto/downloads/voz.pdf

[5] StackOverflow
https://stackoverflow.com/

[6] http://searchcrm.techtarget.com/definition/voice-recognition

[7] http://www.explainthatstuff.com/voicerecognition.html