Recursos para Desenvolvimento de Aplicativos com Suporte a Reconhecimento de Voz para Desktop e Sistemas Embarcados

Rafael Oliveira, Pedro Batista, Nelson Neto e Aldebaro Klautau

[1]Laboratório de Processamento de Sinais – LaPS
Universidade Federal do Pará – UFPA
Rua Augusto Correa, 1 – 660750-110 – Belém, PA, Brasil
http://www.laps.ufpa.br/falabrasil

{rafaelso,pedro,nelsonneto,aldebaro}@ufpa.br

Resumo. *Um sistema de reconhecimento automático de voz é composto por módulos dependentes da língua. Enquanto existem muitos recursos públicos para algumas línguas, como Inglês e Japonês, os recursos para o Português Brasileiro (PB) ainda são escassos. Este trabalho descreve o desenvolvimento de recursos e ferramentas livres para reconhecimento de voz em PB, incluindo uma interface de programação para o sistema Coruja e um modelo acústico para o pacote CMUSphinx.*

Abstract. *An automatic speech recognition system has modules that depend on the language and, while there are many public resources for some languages (e.g., English and Japanese), the resources for Brazilian Portuguese (BP) are still limited. This work describes the development of resources and free tools for BP speech recognition, consisting of an application programming interface for the Coruja system and an acoustic model for the CMUSphinx toolkit.*

1. Introdução

Nos últimos anos, o desempenho dos computadores pessoais tem evoluído com o advento de processadores cada vez mais velozes, fato que viabiliza o uso das tecnologias de voz por meio desses. Existem várias tecnologias de voz, entre elas síntese de voz (TTS, de "text-to-speech") e reconhecimento automático de voz (ASR) são as mais proeminentes. Um sistema TTS [Taylor 2009] é constituído por módulos que convertem textos em linguagem natural em voz sintetizada. ASR [Huang et al. 2001] pode ser visto como o processo inverso ao TTS, onde o sinal de voz digitalizado é convertido em texto.

Apesar da reconhecida importância, as atividades em processamento de voz no Brasil, tanto na academia quanto na indústria ainda não alcançaram a dimensão necessária para que as mesmas tragam benefícios significativos à sociedade. Diante da carência de recursos específicos ao Português Brasileiro (PB), este trabalho objetiva a implementação e disponibilização de recursos para o desenvolvimento de aplicativos com ASR em PB para *desktop* e sistemas embarcados (e.g. *smartphones*, *tablets*, entre outros).

Primeiramente, uma interface de programação seguindo a especificação Java Speech API (JSAPI) [JSAPI 2011] foi implementada para facilitar o uso do sistema de reconhecimento de voz proposto em [Silva et al. 2010], chamado Coruja. Já visando o desenvolvimento de aplicativos baseados em voz para dispositivos móveis, recursos específicos para ASR com o pacote CMUSphinx também foram construídos.

Este trabalho está organizado da seguinte maneira. Na Seção 2 são descritos os principais recursos livres e comerciais para ASR encontrados no mercado. Já a Seção 3 descreve os recursos desenvolvidos pelos autores para o PB. A Seção 4 apresenta os primeiros resultados experimentais para a execução em sistemas embarcados. Finalmente, a Seção 5 conclui o trabalho e sugere pesquisas futuras.

2. Recursos para Desenvolvimento de Aplicativos de Voz

Para o desenvolvimento de aplicativos baseados em voz, é imprescindível a presença de um "engine" de voz (reconhecedores, sintetizadores, ou ambos). Tendo um *engine* disponível para a língua alvo, uma API ("application programming interface") facilita o trabalho do programador durante o processo de desenvolvimento. Existem várias soluções comerciais ofertadas pela Microsoft, Nuance, e outras companhias, contudo, poucos são os recursos (APIs e *engines*) livres disponíveis. A situação é ainda mais complicada para as línguas menos expressivas comercialmente, como o PB. Por exemplo, *softwares* comerciais como o Dragon e IBM Via Voice não oferecem ASR em PB. A seguir, são descritos os principais recursos livres e comerciais encontrados no mercado.

2.1. APIs de Voz

Uma API de voz especifica uma interface que suporta aplicações em TTS e ASR, essa última tanto em aplicações com gramática para comando-e-controle, como para ditado. Dessa forma, as APIs não contêm apenas as funcionalidades de TTS e ASR, mas também métodos e eventos que permitem ao programador abstrair requisitos de baixo nível do *engine*. Os *engines* possuem sua própria API, mas existem pelo menos duas especificações desenvolvidas para uso geral: a Speech API (SAPI) [SAPI 2011] e a Java Speech API.

2.2. Microsoft Speech SDK

A Microsoft desponta como uma das empresas que mais investe em tecnologias de voz. A prova disso são os recursos que a mesma disponibiliza para seu sistema operacional desde a versão Windows 2000, como o Microsoft Speech SDK, um *kit* de desenvolvimento constituído por *engines* de ASR e TTS, e pela SAPI. A Microsoft oferece suporte em ASR a 26 línguas, incluindo o PB. Contudo, todos esses recursos são pagos e protegidos por licenças, fato que dificulta sua utilização no meio acadêmico. Nas seções seguintes, são descritas algumas opções de *software* livre disponíveis.

2.3. CMUSphinx

O CMUSphinx é um pacote de ferramentas para desenvolvimento de aplicativos com suporte a ASR. Dentre as ferramentas disponibilizadas destacam-se: o SphinxTrain [SphinxTrain 2011], conjunto de *scripts* para treinamento de modelos acústicos; o Sphinx-4 [Walker et al. 2004], *engine* em Java que implementa parte da especificação JSAPI (ASR); e o PocketSphinx [Huggins-Daines 2010], implementado na linguagem C e adequado para aplicações em dispositivos móveis. Contudo, o pacote CMUSphinx não disponibiliza recursos de voz para o PB.

2.4. Julius

O Julius [Lee et al. 2001] é um *engine* para ASR de alta performance para grandes vocabulários. Seu pacote de distribuição traz uma API que pode ser acessada via código em

linguagem C/C++. Dentre as dificuldades em se usar essa API, está o fato da mesma não seguir uma especificação, portanto, o código da aplicação que controla o Julius não pode ser reaproveitado para manipular outro *engine*.

2.5. Coruja

Em [Silva et al. 2010], os autores apresentam um sistema ASR para o PB distribuído sob a licença *Berkeley Software Distribution* (BSD). O Coruja, como é chamado, oferece modelos acústicos e de linguagem, além de uma API própria construída para facilitar a tarefa de controlar o *Julius engine*. Essa API visa flexibilidade quanto à linguagem de programação, por isso, foi implementada em C++ sendo compatível com a especificação *Common Language Runtime*, o que permite que a mesma seja utilizada por linguagens suportadas pela plataforma .NET 3.5 ou superior. Entretanto, essa API também não segue uma especificação consagrada como a SAPI ou JSAPI.

3. Os Recursos Desenvolvidos

Este trabalho oferece uma API compatível com a especificação JSAPI para o sistema Coruja, tendo como objetivo tornar possível que o mesmo possa ser controlado a partir da linguagem Java. A segunda parte desta pesquisa foca na construção de um modelo acústico específico para o pacote CMUSphinx, visando sua utilização em sistemas embarcados com o decodificador PocketSphinx. Ambos encontram-se publicamente disponíveis em [FalaBrasil 2011] e são detalhados a seguir.

3.1. JLaPSAPI: Uma API de Voz em Java

A JLaPSAPI opera sobre a API do Coruja (LaPSAPI) para controlar o *engine Julius*, evitando a re-implementação de funcionalidades básicas já implementadas na atual versão do Coruja. A comunicação entre o código em Java e o código C++ é provida pela Java Native Interface (JNI) [Liang 1999].

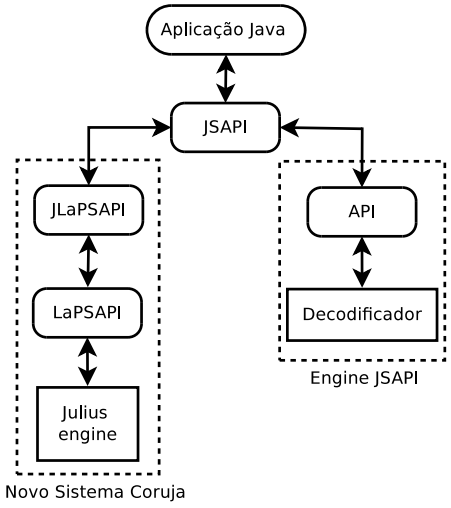

Figura 1. Nova arquitetura do sistema Coruja.

Nessa nova arquitetura, o acesso ao Coruja é feito através de código especificado pela JSAPI. Como mostrado na Figura 1, o programador tem, agora, a possibilidade de

alternar entre o Coruja e qualquer outro *engine* que siga a especificação JSAPI, como o Sphinx-4, sem a necessidade de alteração no código da sua aplicação Java. Atualmente, a JLaPSAPI conta com um conjunto reduzido de métodos e eventos (veja Tabela 1). Entretanto, esses recursos são suficientes para a construção de aplicativos simples com suporte a ASR em PB.

Tabela 1. Métodos e eventos suportados pela JLaPSAPI.

Métodos e Eventos	Descrição Básica
createRecognizer	Cria uma instância do *engine*
allocate	Aloca os recursos do *engine*
deallocate	Desaloca os recursos do *engine*
resume	Inicia o reconhecimento
pause	Pausa o reconhecimento
resultAccepted	Recebe o resultado do reconhecimento

3.2. Modelo Acústico para o CMUSphinx em PB

Um típico sistema ASR adota uma abordagem estatística baseada em modelos ocultos de Markov (HMMs) [Huang et al. 2001] e é composto por cinco blocos principais: *front-end*, dicionário fonético, modelo acústico, modelo de linguagem e decodificador ou reconhecedor, conforme ilustrado na Figura 2.

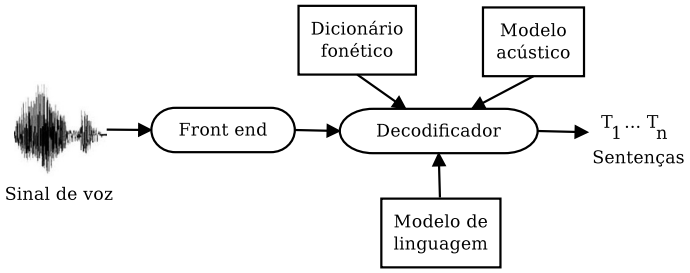

Figura 2. Os principais blocos que compõem um sistema ASR.

Estimar um bom modelo acústico é considerado o maior desafio dentro do projeto de um sistema ASR. Para treinar um modelo acústico é necessária uma base de dados com arquivos de áudio e suas respectivas transcrições ortográficas. A seguir são descritos alguns aspectos do modelo acústico aqui construído para o pacote CMUSphinx em PB.

A base de dados usada para o treinamento do modelo acústico foi composta pelos corpora: LapsStory [FalaBrasil 2011] e West Point [LDC 2011], que somam juntos 21,65 horas de áudio, com taxa de amostragem de 16.000 Hz (mono, 16 bits). O amplamente utilizado *front-end* MFCC (*Mel-Frequency Cepstral Coefficients*) [Huang et al. 2001] foi adotado para parametrizar os arquivos de áudio. Por fim, os parâmetros MFCC foram normalizados através da média cepstral [Huang et al. 2001].

O modelo acústico foi construído de acordo com o tutorial descrito em [SphinxTrain 2011]. A abordagem *flat-start* foi adotada, iniciando com modelos baseados em monofones e com uma Gaussiana por mistura. Em seguida, as HMMs foram expandidas de forma a compor modelos com múltiplas Gaussianas por mistura e

utilizando modelos trifones. Durante todo o processo de treino, o algoritmo de Baum-Welch [Welch 2003] foi utilizado para re-estimar os modelos.

Foram utilizadas inicialmente 39 HMMs (38 monofones + modelo de silêncio), usando como base o dicionário fonético UFPAdic [FalaBrasil 2011] com 65.000 palavras. Cada HMM possui 3 estados na topologia *left-to-right* com *self-loops* e *skip transitions* [Huang et al. 2001]. Então, modelos trifones dependentes de contexto foram criados a partir dos monofones.

Em seguida, os estados dos trifones foram vinculados (*tied-state*) através de uma árvore de decisão gerada automaticamente por um algoritmo de clusterização provido pelo *script* de treinamento. O número de *tied-states*, que define a poda da árvore de decisão, foi setado em 3.000. Após o vínculo dos estados, finalizou-se o processo de treino do modelo com o incremento do número de Gaussianas até 22-Gaussianas por mistura.

A base de dados usada no treinamento do modelo acústico é totalmente livre de ruído, fato que pode prejudicar o desempenho do decodificador. Visando diminuir o efeito do descasamento acústico entre os ambientes de treino e teste, o modelo acústico foi adaptado com a base de dados Spoltech [LDC 2011], gravada em ambientes não controlados (ruidosos) e re-amostrada de 44,1 KHz para 16.000 Hz (mono, 16 bits), utilizando a técnica de adaptação *maximum likelihood linear regression* [Silva et al. 2009].

4. Resultados Experimentais

O modelo acústico descrito na Seção 3.2. foi testado usando o decodificador PocketSphinx, apesar do mesmo também ser compatível com o Sphinx-4. As medidas de desempenho utilizadas foram a taxa de erro por palavra (WER) e escala de tempo real média (RT). O fator RT é obtido dividindo-se o tempo que o sistema gasta para reconhecer uma frase, pela duração da mesma. As simulações foram realizadas em um computador Intel(R) Pentium Dual Core 1,8 GHz com 2 GB de memória RAM.

A base de dados usada no processo de avaliação do modelo acústico foi a Laps-Benchmark [FalaBrasil 2011], com 54 minutos de fala contínua (ditado). Nota-se que a base LapsBenchmark é totalmente desvinculada da base de dados usada na fase de treinamento (locutores e sentenças são distintos). O modelo de linguagem trigrama usado nos experimentos foi o LaPSLM v1.7 [FalaBrasil 2011]. O melhor resultado com 46,25% de WER e 0,81 de RT foi obtido com os parâmetros de decodificação descritos na Tabela 2.

Tabela 2. Parâmetros de decodificação do PocketSphinx.

Parâmetro	Valor
Word beam width	1e-40
Pruning beam width	1e-80
Word insertion penalty	0,2
Language model scale factor	6

A WER obtida é relativamente alta, se comparada a outros sistemas ASR para PB. Por exemplo, [Silva et al. 2010] obteve 29,37% de WER e 0,9 de RT, usando a mesma base de teste para ditado e o decodificador HDecode (parte do HTK). Contudo, o modelo acústico desenvolvido funciona bem em aplicações com gramáticas de comando e controle.

5. Conclusões e Trabalhos Futuros

Através dos recursos desenvolvidos e disponibilizados neste trabalho, programadores sem conhecimento técnico na área de processamento de voz podem inserir ASR em suas aplicações, tanto para *desktop* através da linguagem Java, quanto em plataformas móveis.

Os resultados preliminares demonstram um desempenho razoável em grandes vocabulários do modelo acústico desenvolvido, mas que pode ser incrementado com árvores de decisão baseadas em conhecimento linguístico, por exemplo. Os recursos aqui implementados já se encontram em uso pela comunidade. Por exemplo, em [Iphone 2011], um aplicativo com ASR foi implementado para o *iphone* usando o PocketSphinx e o modelo acústico desenvolvido.

Referências

FalaBrasil (Visited in March, 2011). www.laps.ufpa.br/falabrasil.

Huang, X., Acero, A., and Hon, H. (2001). *Spoken Language Processing*. Prentice-Hall.

Huggins-Daines, D. (2010). *PocketSphinx API Documentation*. Version 0.6.

Iphone (Visited in March, 2011). http://www.youtube.com/watch?v=golbklihcfg.

JSAPI (Visited in March, 2011). java.sun.com/products/java-media/speech/.

LDC (Visited in March, 2011). http://www.ldc.upenn.edu.

Lee, A., Kawahara, T., and Shikano, K. (2001). Julius - an open source real-time large vocabulary recognition engine. *Proc. European Conference on Speech Communication and Technology*, pages 1691–1694.

Liang, S. (1999). *The Java™Native Interface Programmer's Guide and Specification*. Addison-Wesley.

SAPI (Visited in March, 2011). www.microsoft.com/speech/.

Silva, P., Batista, P., Neto, N., and Klautau, A. (2010). An open-source speech recognizer for Brazilian Portuguese with a windows programming interface. *The International Conference on Computational Processing of Portuguese (PROPOR)*.

Silva, P., Neto, N., and Klautau, A. (2009). Novos recursos e utilização de adaptação de locutor no desenvolvimento de um sistema de reconhecimento de voz para o Português Brasileiro. *In XXVII Simpósio Brasileiro de Telecomunicações*.

SphinxTrain (Visited in March, 2011). http://cmusphinx.sourceforge.net/wiki/tutorialam/.

Taylor, P. (2009). *Text-To-Speech Synthesis*. Cambridge University Press.

Walker, W., Lamere, P., Kwok, P., Raj, B., Singh, R., Gouvea, E., Wolf, P., and Woelfel, J. (2004). Sphinx-4: A flexible open source framework for speech recognition. Technical report, Sun Microsystems Inc.

Welch, L. R. (2003). Hidden Markov models and the Baum-Welch algorithm. *IEEE Information Theory Society Newsletter*, 53:10–12.