VMS: Speech To Text

Sidney Loyola de Sá[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)

sidney_loyola@id.uff.br

1. Introdução

A *Virtual Programmable IoT Multimedia Sensor* (V-PRISM) é uma arquitetura proposta em [Battisti et al. 2019] que cria um ambiente virtualizado executando na borda. Os componentes e sensores físicos acessam os dispositivos virtuais que enviam o fluxo para os sensores multimídia (VMS). Estes sensores podem realizar vários tipos de processamento, por exemplo, transformar o formato de um arquivo de áudio.

O V-PRIM propicia o desenvolvimento de aplicações virtuais sem a preocupação em como será realizado o *deploy*, além disso, a principal vantagem da sua utilização é reduzir o *delay* da comunicação entre os sensores, ideal para a implantação do paradigma de Internet das coisas (IoT do inglês *Internet of Things*).

Os VMS, conforme a taxonomia apresentada em [Battisti et al. 2019], podem ser categorizados em :

- ***Thing***. Os sensores virtuais desta categoria encaminham os dados realçando atributos do dispositivo físico.
- ***Process***. Nesta categoria os fluxos de dados recebidos são processados e entregues em outro formato, mas da mesma natureza, por exemplo, transformação de arquivos WAV em MP3.
- ***Service***. Os sensores virtuais desta categoria processam o fluxo de dados transformando a natureza do fluxo que será entregue. Por exemplo, a entrada de áudio transformada em texto na saída.

O objetivo deste trabalho é desenvolver um sensor de multimídia virtual, pertencente a categoria *Service*, que recebe uma entrada de áudio e o transforma em texto, enviando-o para as demais aplicações, através do protocolo MQTT[1] (MQ *Telemetry Transport*), que foi projetado para transmissão de mensagens leves entre sensores e pequenos dispositivos móveis.

2. Reconhecimento de Voz

Processamento de Linguagem Natural (NLP do inglês *Natural Language Processing*) é um subcampo da Inteligência Artificial que investiga métodos e técnicas, através das quais os agentes computacionais seriam capazes de se comunicar com os seres humanos. Além disso, essas técnicas podem ser utilizadas para extração de informação a partir dos diversos modelos de linguagem [Russel and I.Norvig 2013].

Diferentemente das linguagens formais, a comunicação humana permite ambiguidades que não deixa "As linguagens naturais serem caracterizadas como um conjunto de

[1]www.mqtt.org

sentenças definitivas" [Russel and I.Norvig 2013]. Portanto, busca-se a probabilidade de uma sentença pertencer ou não a determinada linguagem. Ou seja, utilizando a palavra "maçã" como exemplo e, escolhendo uma sentença aleatória do texto, desejamos saber qual a probabilidade P dessa sentença ser "maçã" [Russel and I.Norvig 2013].

Segundo [Russel and I.Norvig 2013] a distribuição de probabilidade sobre sequência de caracteres pode ser escrita $P(c_{1:n})$, por exemplo $P(a) = 0,27$ e $P(zgq) = 0,00000002$. Chamamos esses conjuntos de distribuição de probabilidades de modelos n-grama. Esses modelos são definidos como uma **cadeia de Markov** de ordem $n-1$, nessas cadeias a probabilidade do caractere c_i depende dos caracteres imediatamente anteriores. Na equação 1 temos um modelo de trigrama(3-grama) [Russel and I.Norvig 2013].

$$P(c_i|c_{1:i-1}) = P(c_i|c_{i-2:i-1}) \tag{1}$$

Esses conceitos formam os modelos de linguagem, eles são aplicados no processamento de textos e formam a base da transcrição de aúdio para texto. Os sons possuem diversas características e a sua captura não é exata, havendo perda de informação, assim aplicam-se modelos discriminativos que calculam a probabilidade dos atributos ocultos em função dos demais componentes. Por exemplo, em um texto (ou áudio) $e_{1:n}$, o modelo encontra a sequência de estado oculto $X_{1:n}$ que maximiza $P(X_{1:n}|e_{1:n})$ [Russel and I.Norvig 2013].

O processo de reconhecimento da voz, conforme as informações apresentadas na descrição da API CMUSphinx[Shmyrev 2020], necessita de três modelos: o modelo acústico, o dicionário fonético (mapeia as palavras para os fonemas) e o modelo da linguagem que possui as distribuições de probabilidade auxiliando na identificação das partes ocultas.

3. Desenvolvimento do VMS

O VMS proposto foi desenvolvido na linguagem Python, utiliza a biblioteca *SpeechRecognition* e a API *Google Speech Recognition* e a API*CMUSphinx*, através da biblioteca *Pocketsphinx* desenvolvida na linguagem C.

O ambiente de desenvolvimento possuía a seguinte configuração: Sistema Operacional Windows 10, processador Intel Core i7, 16 GB de RAM, Software VirtualBox 6.0, Sistema Operacional Ubuntu 18.04 instalado na Máquina Virtual, IDE PyCharm 2020.1.3 (Community Edition) e interpretador Python 3.6.

O VMS atende aos seguintes requisitos:

- Recebe fluxo de áudio vida UDP.
- Transcreve o áudio em texto.
- Possibilidade de utilização online e offline.
- envia o texto reconhecido para um *Broker* MQTT.

As restrições dessa versão Beta são:

- O áudio precisa estar no formato pré-determinado[2] (16 bits, 2 canais, 44.100 Hz).

[2]Este é o formato utilizado por padrão na maior parte dos gravadores dos computadores e dispositivos móveis

- A performance do modelo offline está abaixo do esperado, necessitando de calibração.
- Reconhece apenas o idioma Inglês.

Desenvolvimentos Futuros:

- Parametrização para utilização de diversos idiomas
- Inclusão de modelo para utilização do idioma Português offline
- Parametrização para permitir a calibração do VMS de acordo com o usuário, aumentando a acurácia offline.
- Possibilidade de envio de áudio em diferentes formatos.

3.1. Instalação

O VMS está disponível em `https://github.com/loyoladesa/modulo_vprism` e para instalá-lo é necessário possuir o V-PRISM configurado. O arquivo Dockerfile, que está na pasta *speech_to_text* precisa ser compilado com o comando:

```
sudo docker build . -t alfa/vms/voice_recognizer
```

Após isso, criar um Novo Tipo de VMS na interface web do V-PRISM com as seguintes informações:

```
Name : VMS = S2T
Docker Image: alfa/vms/voice_recognizer
Startup Parameters: TOPIC IP PORT MODE | Example: ABC 172.17.0.1 1883 ON
Description:Perform Speech Recognition by transforming audio into text
```

Os parâmetros necessários para a inicialização são:

1. Tópico MQTT - informa em qual tópico o texto deverá ser publicado.
2. IP do *Broker* MQTT.
3. Porta do *Broker* MQTT.
4. Modo de utilização do VMS: ON ele irá utilizar a API do Google que fica online; OFF - ele irá utilizar a API Sphinx que funciona offline.

Para inicializar o VMS podemos utilizar os parâmetros conforme o exemplo acima, no *Port Forward* colocamos o seguinte valor :

```
10001:5000
```

Após esses passos deve-se obter a tela, conforme a Figura 1.

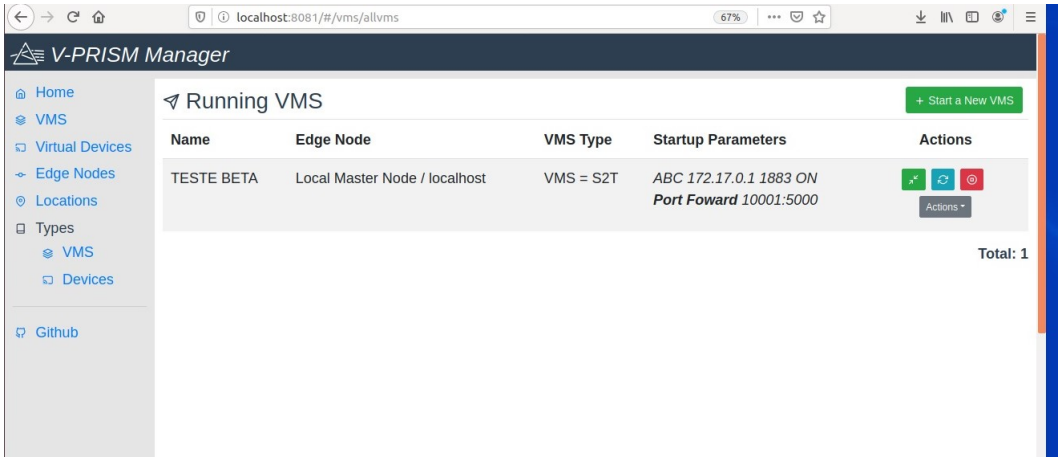

Figura 1. Lista de VMS inicializados

Para se verificar a correção da inicialização acessamos a página de *log* na interface web obtendo a visualização da Figura 2.

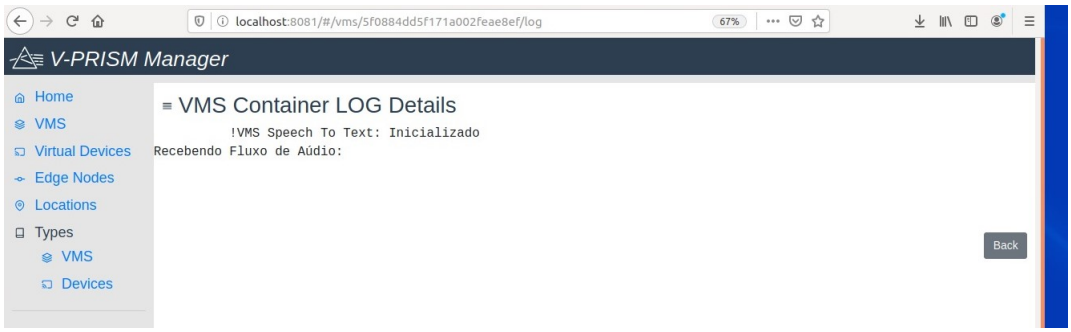

Figura 2. Visualização do Log

Para verificar o funcionamento correto, deve-se enviar arquivos de áudio via UDP para a porta 10001. Executamos a seguinte linha de comando no Ubuntu:

```
1   gst-launch-1.0 filesrc location=turn_on.wav ! wavparse !
      audioconvert ! udpsink host=localhost port=10001
```

O VMS irá converter o áudio e veremos a saída da Figura 3.

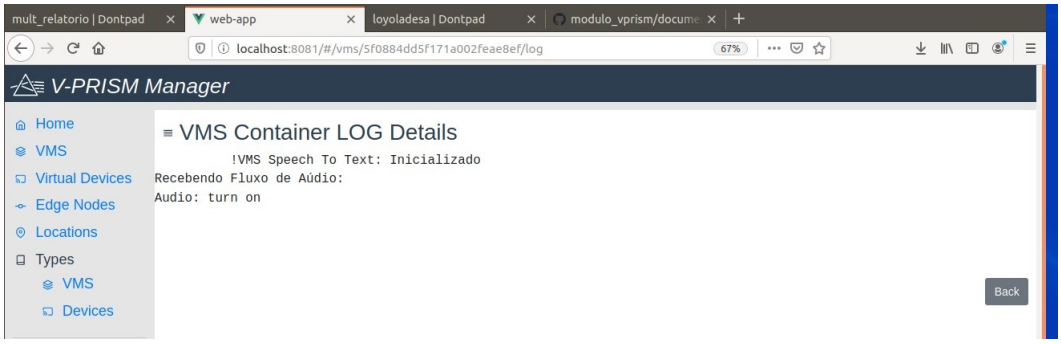

Figura 3. Áudio Transcrito

Executamos o envio de diversos áudios e o resultado está apresentado na Figura 4.

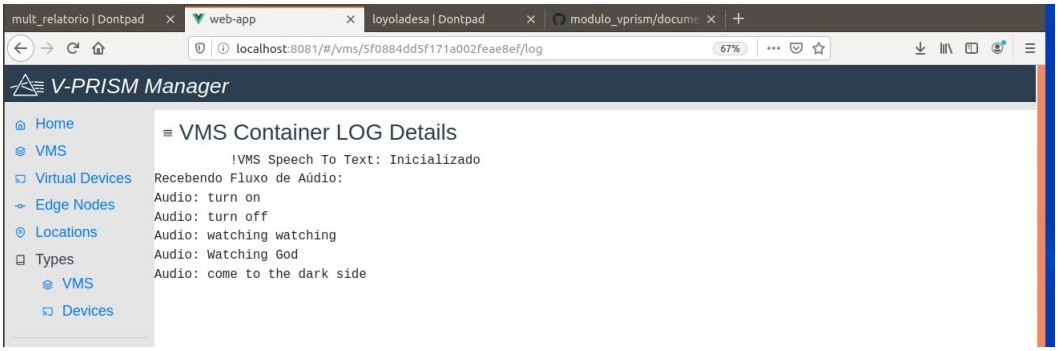

Figura 4. Lista de Áudios Convertidos

4. Código Fonte

A versão beta do VMS desenvolvido apresenta o código mais simples possível, pois o objetivo principal era verificar a possibilidade de execução e integração de software de reconhecimento de voz com a plataforma V-PRISM. O código fonte é dividido em quatro funções principais executadas por *threads* distintas. A primeira função é a *main*, responsável por executar o código e inicializar as demais funções: *udpStream, transcribe, send*.

A função *udpStream* lida com a captura do fluxo UDP. A função *send* envia o arquivo trancrito para o *Broker* MQTT. A função *transcribe* executa a função principal do VMS, para isso ela transforma o fluxo em um arquivo de áudio temporário passado para a biblioteca *SpeechRecognition*:

```
1    audio = voice_recognizer.record(source)
```

A biblioteca *SpeechRecognition* acessa a API de forma transparente, ou seja, o código da aplicação não precisa ser alterado, apenas editamos o nome do método de chamada. A transcrição acontece nas seguintes linhas de código:

```
frase = voice_recognizer.recognize_sphinx(audio)

frase = voice_recognizer.recognize_google(audio)
```

No código acima pode-se passar um segundo parâmetro além do áudio, que determinaria o idioma a ser utilizado. A API do Google possui um modelo para o português, já a API Sphinx permite que se produza e integre um modelo, caso se deseje utilizar o português [Naslausky 2018].

5. Resultados

O reconhecimento de voz e a sua transcrição para textos é uma realidade nos dias atuais, mas ainda não é uma tarefa trivial, nem um problema completamente resolvido. Por exemplo, nenhum modelo consegue reconhecer uma palavra completamente nova. O modelo que atinge a melhor performance utilizando a métrica WER [3], é a API do Google com 9%, já a API Sphinx alcançou 37% no experimento realizado em [Këpuska and Bohouta 2017].

A API Sphinx pode alcançar resultados muito melhores do que o realizado neste trabalho, basta produzirmos um modelo e calibrarmos o reconhecedor de voz. A maioria dos passos está descrito em [Naslausky 2018]. Esta API foi escolhida como a alternativa offline por ser *Open Source*, permitindo a construção de novos modelos e evolução da acurácia com o desenvolvimento do projeto.

A API do Google obtém melhores resultados, em grande parte, devido a utilização de dados do mundo inteiro e por estar em constante treinamento. Além de utilizarem modelos baseados em Redes Neurais que demandam grande poder computacional.

6. Conclusão

Este trabalho apresentou o desenvolvimento de um VMS integrado ao V-PRISM que realiza o reconhecimento de voz transformando-o em texto. As limitações existentes na versão Beta, principalmente, em relação a utilização da API offline estão relacionadas ao desenvolvimento de modelos de linguagem.

Enquanto a API do Google utiliza uma base de dados para treinamento gigantesca e constantemente atualizada, as aplicações offline precisam de modelos específicos e calibrados com um modelo acústico, o mais próximo possível da utilização, atingindo resultados profissionais. Não foi encontrada nenhuma limitação técnica em relação ao V-PRISM para o desenvolvimento desses modelos.

Em relação a trabalhos futuros existe a necessidade de desenvolvimento de um modelo que proporcione maior acurácia na utilização da API Sphinx, proporcionando a possibilidade dos usuários calibrarem o VMS de acordo com seu uso especifico. Além disso, a parametrização do VMS para que se utilizassem os diversos idiomas já disponíveis nas duas API seria muito útil. Por último, a evolução do software pode permitir uma maior confiabilidade na captura do aúdio UDP, talvez realizando *checksum* em nível de aplicação.

[3] https://pt.qwe.wiki/wiki/Word_error_rate

Referências

Battisti, A. L. , Muchaluat-Saade, D. C., and Delicato, F. C. (2019). V-PRISM: An edge-based IoT architecture to virtualize multimedia sensors. *WMC19 - Technical Session 3 - Multimedia and Edge Computing*, page 6.

Këpuska, V. and Bohouta, G. (2017). Comparing speech recognition systems (microsoft api, google api and cmu sphinx). *Int. J. Eng. Res. Appl*, 7(03):20–24.

Naslausky, E. (2018). *TRANSCRIÇÃO DE ÁUDIO UTILIZANDO BIBLIOTECAS OFF-LINE*. PhD thesis, Universidade Federal do Rio de Janeiro.

Russel, S. J. and I.Norvig, P. (2013). *Inteligencia Artificial*. Elsevier.

Shmyrev, N. (2020). Basic concepts of speech recognition. `http://cmusphinx.github.io/wiki/tutorialconcepts/`Acesso em: 10 de julho de 2020.