Implementation of the Wake Word for Smart Home Automation System

Igor Stefanović, Eleonora Nan
Faculty of Technical Sciences
University of Novi Sad
Serbia

Boris Radin
Faculty of Technical Sciences
University of Novi Sad
Serbia

***Abstract*—As part of the voice command system for the existing smart home automation solution, we want to build the wake word module. This module should continuously listen and process the sounds from the environment, in order to detect the pre-defined wake word. Once the wake word is pronounced by the user, wake word module should trigger the actual voice command processing, which will result in the action within the home automation system. In this paper, we evaluate the possibility of using some of the existing offline speech recognition engines for this purpose. We analyze their accuracy and performance, and set guidelines for the future work.**

Keywords—wake word; home automation; voice command

I. Introduction

Voice control is becoming one of the most important and highly demanded features of home automation (HA) systems [1]-[2].
Voice control module should convert the spoken command to text first, and then map it to the appropriate action in the HA system. Developing the speech-to-text engine is a demanding task, but there are already a number of available speech-to-text engines, which can be used as a building block of the custom voice control module for HA systems [3]-[6]. However, the implementation of the invocation mechanism remains a challenge. Namely, the goal is to minimize the number of cases in which the action in the HA system is triggered without the actual user intent. This is impossible if the voice command module continuously listens and processes the sounds from the environment. In order to achieve better control, we can require users to push a button on a dedicated device, before issuing a command. However, this requires that the user approaches the actual device. The more comfortable way for a user to address the HA voice control is to pronounce a pre-defined wake word, as a signal that the actual voice command is to follow.

In this paper, we will focus on the challenges of implementing the wake word detection mechanism as part of the HA voice control module. We explore the possibility of using some of the available speech recognition engines for this purpose [3]-[6]. We focus on the engines that are known to work well on embedded devices, as we want to add voice command processing to the existing smart home gateway [7]. Julius is a high-performance, continuous speech recognition software, based on word N-gram and context-dependent hidden Markov models [3]. However, it requires the strong DSP processor, and cannot perform well on the targeted embedded device. Kaldi requires a cluster of Linux machines to achieve its full performance, and running it on a single machine reduces its speed and accuracy [4]. On the other hand, Pocketsphinx is a voice recognition engine designed for embedded devices [5], and is available as a C library. Also, Snowboy [6] is a highly customizable wake word detection engine that is compatible with embedded devices with low memory.

Based on the investigation, we decided to implement and test the wake word module using both Pocketsphinx and Snowboy, and compare these two implementations in terms of accuracy and performance. In the following section, we will briefly describe the architecture of the voice control system for HA. Then, we will focus on the implementation of the wake word module using both [5] and [6], and compare these two approaches.

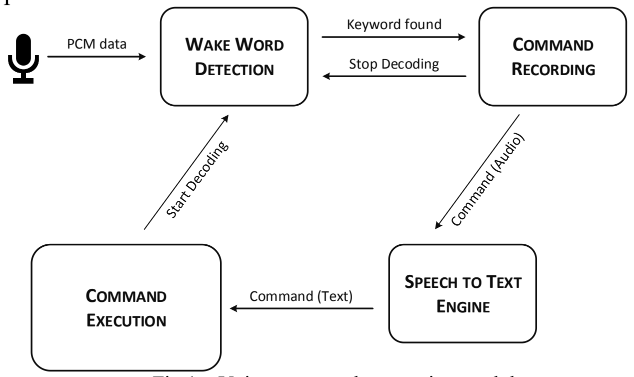

Fig 1. Voice command processing module

II. Voice Control System Architecture

In this section, we will briefly describe the voice control module architecture.

The voice control module is represented in Fig. 1. The wake word detection component continuously records and processes the sounds from the environment. Once the pre-defined wake word is pronounced by the user, it triggers the actual command recording. The recorded command is then processed by the speech-to-text engine. Textual output of the

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant number TR32034.

engine is finally converted into MQTT protocol [8] commands for the HA system by command execution module.

III. Implementation of Wake Word Detection

In this section, we will focus on the actual implementation of the wake word detection module.

Continuous recording of user's voice is implemented through the background thread. The recorded audio from environment is stored in the PCM buffer, and a window of ~2s of duration is analyzed by the wake word recognition engine. When the wake word is detected within the analyzed data, the background thread triggers further command processing. At the same time, this background thread stops processing the recorded PCM buffer data, and waits for command processing to end. Once the command is executed, the background wake word detection thread is started again, and PCM buffer data is analyzed.

As already said, the analysis of audio from environment is performed by a third-party wake word recognition engine, i.e. Pocketsphinx or Snowboy. Pocketsphinx can perform general speech-to-text conversion, but it can also be configured to work in a wake word spotting mode. On the other hand, Snowboy is designed specifically for the scenario of wake word detection. It can be configured to work with different sensitivity levels, which affect its accuracy. Sensitivity ranges from 0 to 1. The more sensitive the engine is, more instances of the wake will be detected, but there will also be more cases of words mistakenly identified as the wake word.

IV. Testing results

In this section, we compare the two solutions, based on their accuracy and performance.

To evaluate the quality of wake word detection, we observe precision P – Eq. 1, recall R – Eq. 2, and overall accuracy A – Eq. 3. Here, N_{TP} represents the number of true positives, i.e. the number of times when the wake word was pronounced and detected correctly. Similarly, the value N_{TN} is the number of true negatives, i.e. the number of times when the word other than the wake word was pronounced, and it was correctly decided that this was not the wake word. The value N_{FP} is the number of false positives, i.e. the number of times when some other word was mistakenly identified as the wake word. The number of false negatives N_{FN} is the number of times the wake word was pronounced, but was not detected correctly.

$$P = \frac{N_{TP}}{N_{TP}+N_{FP}} \quad (1)$$

$$R = \frac{N_{TP}}{N_{TP}+N_{FN}} \quad (2)$$

$$A = \frac{N_{TP}+N_{TN}}{N_{TP}+N_{TN}+N_{FP}+N_{FN}} \quad (3)$$

Testing results are presented in Table 1. Tests were performed for Pocketsphinx configured to work in wake word spotting mode, and for Snowboy with two sensitivity levels: 0.4 and 0.5. Using higher sensitivity levels significantly decreased Snowboy's precision, while the lower sensitivity levels negatively affected the recall.

TABLE I. Accuracy of Wake Word Module

Voice Recognition Engine	Precision (*P*)	Recall (*R*)	Accuracy (*A*)
Snowboy with sensitivity 0.5	0.48	0.86	0.64
Snowboy with sensitivity 0.4	0.74	0.72	0.83
Pocketsphinx	1.00	0.74	0.87

To evaluate the performance, we tested both implementations on a PC, and on an embedded device (Raspberry Pi 2 – RPI 2, with a 900MHz quad-core ARM Cortex-A7 CPU and 1GB RAM). Both implementations performed well on the PC, without noticeable latency. On Raspberry Pi, the speed of Snowboy implementation remained satisfactory. On the other hand, the Pocketsphinx implementation suffered from the considerable amount of latency, which impedes its practical application.

V. Conclusion

In this paper, we analyzed two implementations of the wake word detection module, based on Pocketsphinx and Snowboy voice recognition engines. Experiments have shown that Pocketsphinx has better accuracy than Snowboy, but its performance is limited on RPI2. In the cases when the voice command processing module is performed on a more powerful device, Pocketsphinx implementation is preferred, due to the better accuracy. However, for the embedded device scenario, Snowboy is a better solution, due to its satisfactory performance.

References

[1] T. Giannakopoulos, N.A. Tatlas, T. Ganchev and I. Potamitis, "A Practical, Real-Time Speech-Driven Home Automation Front-end" *IEEE Transactions on Consumer Electronics*, Vol. 51 (2), pp. 514- 523, May 2005.

[2] I. V. McLoughlin and H. R. Sharifzadeh, "Speech recognition engine adaptions for smart home dialogues", *Proc. of 6th ICSP*, 2007.

[3] A. Lee, T. Kawahara, "Recent Development of Open-Source Speech Recognition Engine Julius", *APSIPA ASC*, 2009.

[4] D. Povey, A. Ghoshal, G. Boulianne, et al., "The Kaldi Speech Recognition Toolkit", *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, 2011.

[5] D. Huggins-Daines, M. Kumar, et al., "Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices", *Proc. of IEEE ICASSP*, 2006.

[6] Snowboy, [Online] Available: https://github.com/Kitt-AI/snowboy

[7] M. Bjelica, B. Mrazovac, V. Vojnovic, I. Papp, "Gateway device for energy-saving cloud-enabled smart homes", *Proc. of MIPRO*, 2012.

[8] A. Banks, R. Gupta, "MQTT Version 3.1.1." [Online] Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf, 2014.

[9] Powers, David M W, "Evaluation: From Precision, recall and F-Measure to ROC, Informedness, Markedness & Correlation", *Journal of Machine Learning Technologies*, December 2007.