Excel@FIT 2015
http://excel.fit.vutbr.cz

Speech recognition system in the A-PiMod project

Karel Beneš*

Abstract
This paper describes the construction of a speech recognition system for usage in the aeronautics cockpit. The primary goal of the recognizer is to allow pilots to control parts of the cockpit by voice. Speech recognition in the cockpit is challenging, because of changing context, variable noise and the possibility of off-talk. The recognizer is based on the Kaldi speech recognition toolkit and several project-specific components are implemented in C++. Additionally, a specific way of creating language model for coping with noises is presented. In general, we describe how to use a research-oriented speech recognition toolkit in a real-world application.

Keywords: Grammar-based language model — Push-to-talk button — Automatic Speech Recognition

Supplementary Material: *N/A*

*xbenes20@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

To make the air traffic as safe as possible, various approaches to achieve safety have been proposed [1][2]. One way of increasing safety by improved human-machine interaction is studied in the European Commission project A-PiMod – Applying Pilot Models for Safer Aircraft [3]. Within this project, a speech recognition engine is expected to allow pilots to control the cockpit by voice. It is hypothesized, that the stress level is reduced this way, as the pilot can keep his hands free for whatever other purpose. Additionally, this speech recognizer should eventually monitor overall voice activity of the pilot, so that the stress level can be estimated and unexpected events can be captured, thus allowing the whole system to react to them properly.

In order to create a satisfactory speech recognizer, several requirements must be met: As the cockpit-pilot interface can be in several distinct states (viewing a map, selecting waypoint or altitude etc.), respective *Interaction Contexts* have been defined. The actual Interaction Context defines a set of valid commands. Therefore the recognizer needs to keep notion of this Interaction Context. As this context can change at an arbitrary time, the recognizer must properly react by changing the language model. Also, the recognition will take place in a noisy environment, the microphone may not be close-talking, and the pilot may happen to speak a non-command utterance. Therefore, means of suppressing these effects must be developed. Additionally, two parameters should always be optimized: the recognizer should decode the utterance with a latency as small as possible, and the recognition should be as accurate as possible.

As some the above-described constraints are quite project-specific, no out-of-the-box solution exists that can be immediately applied. On the other hand, there are numerous speech recognition engines available. The Kaldi speech recognition toolkit [4] was chosen as

a basis for the application for the following reasons: It provides implementation of most of the state-of-the-art speech recognition techniques, it is designed as a set of very simple tools and it is open-source. Last but not least, the Speech@FIT group is one of the founders of the project, thus there is a wide understanding of the toolkit in the group.

As the project is in the middle of its lifespan, different parts of it are at different stages of development. I have implemented the communication of the recognition system with other parts of the A-PiMod system, so the recognizer is fully integrated. The acoustic modeling is kept as simple as possible, so that it can be easily extended with respect to more data which are yet to be collected. As a quick solution to implement a model for non-grammatical speech (off-talk), I implemented a kind of garbage collection model in a way, that is known from keyword spotting. This approach is suitable for future extension towards the integration of a large vocabulary continuous speech recognition system. This model of out-of-grammar speech is referred to as *background model* in the rest of this paper.

The organization of this paper is the following: In Section 2, the overall system design is described. Also, components specific to this project are presented and their role in the system is defined. Section 3 deals with the language modeling in the A-PiMod project and presents the background model as well. Finally, Section 4 describes the acoustic model in the system and gives an overview of the recognizer performance.

2. Basic components of the system

The whole system is implemented as a client-server application. The objective of the client is to capture audio from the microphone and send it to the server, which is in turn responsible for the decoding itself.

Additional steps (Subsections 2.1,2.2 and 2.3) are added into the decoding pipeline, so that the push-to-talk (PTT) and time information is handled correctly. As the core speech recognition blocks are computationally demanding, these additional blocks also provide nonblocking function calls, that allow the system to respond to commands from other parts of the system, while partially processing the audio input.

To understand the role of these additional components, the basic organization of the common speech processing pipeline has to be outlined, for details refer to Section 4. At first, speech is segmented into overlapping time frames, and a set front-end features is computed for each frame. Different normalizations, such as cepstral mean removal, and transformations, such as adding derivatives, are then applied to improve the discriminativity of the features while reducing the sensitivity to differences between speakers, microphones etc. The final features are then captured by a *Decodable*, which is the component able to compute the likelihood of the feature vector given an acoustic unit. On top of this Decodable component, the *decoder* operates. The task of the decoder is to use these likelihoods to find the sentence that matches the utterance best, constrained by the language model.

2.1 Processing push-to-talk input

It is usual in the cockpit, that microphone input is transmitted only when a designated push-to-talk button is pushed. This concept is exploited in the A-PiMod project, so the beginning and end of every utterance is determined by pushing and releasing this button.

Most of the PTT-related work is achieved by a component called *PTT buffer*. This buffer is inserted after the first block segmenting speech into frames – in the current setup the Mel-frequency Cepstral Coefficients (MFCCs) computation – and also accepts information about the state of the PTT button from a separate source. The role of the PTT buffer is illustrated in the Figure 1.

Further in the processing pipeline, there is no notion of the PTT buffer, therefore it does not limit the transformations applied to the features in any way.

To make the PTT control more comfortable, a 515 ms segment of audio input preceding the PTT push is decoded as well. This is achieved by keeping a ring-buffer of size 50 frames inside the PTT buffer.

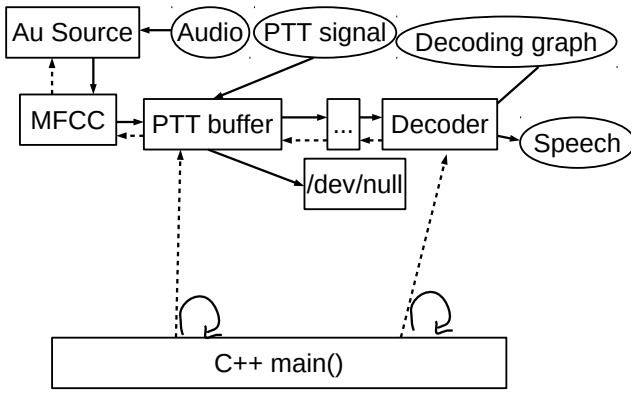


Figure 1. Use of the push-to-talk controlled buffer in the feature processing pipeline. Solid lines represent data-flow, while dashed lines stand for invocations. The main loop of the recognizer can operate either on the decoder, or on the PTT buffer in case the PTT button is released. In that case, the oldest feature vectors are discarded.

2.2 Response to incoming commands

The recognition engine is expected to respond to control commands, mainly the requirement to change the Interaction Context and PTT events. These commands are delivered via *the Datapool*[1], a many-to-many communication platform, in the form of discrete messages. As these commands come in asynchronously to the flow of the recognition, a message processing scheme was designed to present them to the recognizer. The overview of the schema is illustrated in Figure 2.

Except for the decoding itself, the recognition system regularly checks the Datapool for more messages after processing every batch of audio into frames. A latency in retrieving the messages from Datapool is introduced, while the recognizer is actually decoding, but it does not matter, because during decoding itself, no decisions are based on the information from the Datapool.

Every incoming message is passed to the respective state holding object, such as the object responsible for Interaction Context information. This component parses the message and retrieves any relevant information from it. When a decision based on information from the Datapool has to be made, this information is provided by the respective state-holding object. This way, the input stream of messages is effectively encapsulated in such a way, that allows the rest of the system to inspect the information at any time desired.

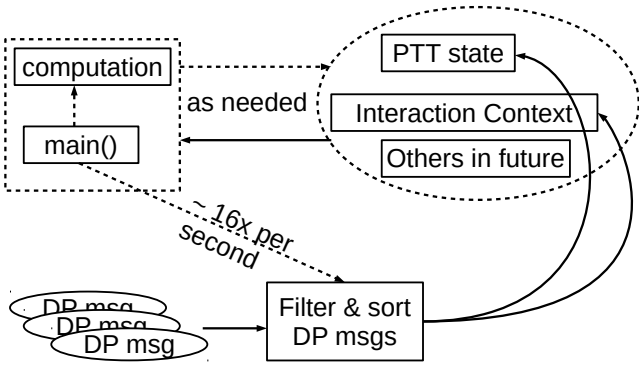


Figure 2. Schema of the reaction of the recognizer reaction to the commands, "DP" stands for Datapool. The main loop checks the Datapool frequently for new messages, their content is used to update the corresponding objects. Their state is then checked whenever needed, asynchronously to to recieving the messages.

[1]Datapool is communication platform developed by Deutsches Zentrum für Luft- und Raumfahrt, Braunschweig. There is no publication describing it, except for private API description.

2.3 Adding time labels to recognized words

In order to further process the recognition result in the A-PiMod system, time labels are required to mark the beginning and ending time of each word. The Kaldi toolkit provides only information about frame alignment and a portion of the frames is not even presented to the decoder (refer to the Figure 1), so additional effort is needed to assign every frame a correct timestamp. As explained in this section, doing so involves both the server and the client.

The best moment to get the information about timing is the moment we capture the audio, because anytime later, more and more nondeterministic delay is introduced (network delay, process switching, etc.). Therefore it is the client's task. The information about audio timing is then transferred together with the audio, one timestamp per every 1024 audio samples. As the sampling frequency in A-PiMod is 16 kHz, we have nearly 16 updates of timing information per second, which ensures this information is up to date. The times of the other samples can be simply computed by exploiting the knowledge of the sampling frequency.

On the side of the server, the timing information is first assigned to the frames. This was achieved by enhancing the component computing the MFCCs. The PTT buffer operating on top of this component is then responsible for splitting the timing information from the features. The timing information is saved in a dedicated component and can later be requested by the output formatter.

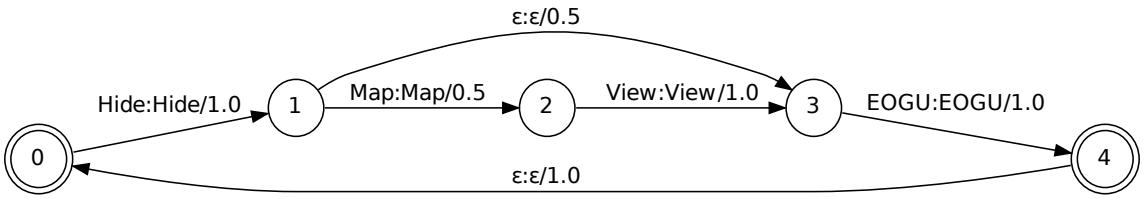
This time-information-holding-object exploits the information about frame shift[2], thus it does not need to save timestamp of every frame. The overall scheme of processing time labels is illustrated in the Figure 3.

3. Language and background modeling

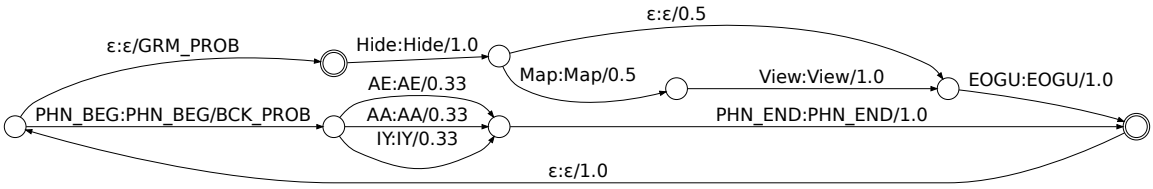
The core of the language model is given by the control grammar, which is in turn given by the Interaction Context. For the use in the Kaldi system, the grammar is represented as a weighted finite state transducer (WFST) [5]. To compile a grammar into an FST, the Grammator tool [6] is used. An example of grammar of the simplest Interaction Context *Map View* is shown in Figure 4a.

A language model based solely on such grammar exhibits undesired behavior, when the captured utterance is not a clean example from the grammar. As the set of hypotheses is limited by the language model, the decoder always tries to find the best match from this set. Thus any hesitations or similar sounds get mapped

[2]Frame shift is the amount of time between the start of two consecutive frames. A typical value is 10 ms.



(a) Original Map View grammar



(b) Grammar enhanced by phone loop. For brevity, only phonemes "ae", "aa" and "iy" are considered.

Figure 4. Illustration of enhancing a grammar by the phone loop for background modeling. Dummy words "EOGU", "PHN_BEG" and "PHN_END" mark the end of a grammar utterance and the beginning and the end of background model, respectively. They are used for straightforward post processing of the recognition result. When the background model is added (Subfigure b), a pair of tied parameters is introduced: GRM_PROB and BCK_PROB, where GMR_PROB + BCK_PROB = 1. These parameters model the probability of an utterance being in-grammar or they can be alternatively interpreted as an *insertion penalty*.

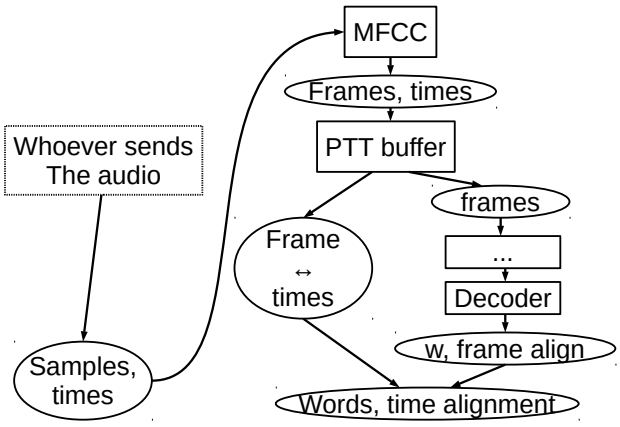


Figure 3. Scheme of processing timestamps. The timing information is captured by the client and sent with the audio samples. The server then assigns the information to frames and stores the information separately. This information is eventually picked up to construct output messages with proper time alignment.

to the closest grammar utterance. To overcome this issue, either specific acoustic models can be trained for hesitations (this is typically done in LVCSR systems) or a more general background model can be employed.

Taking inspiration from the acoustic keyword spotting [7], a phone loop is added, as illustrated in the Figure 4b. Although this approach is not fully consistent, in the sense that it puts phones to the level of words, it works surprisingly well as discussed in Section 4. The useful side-effect of creating the background model on the level of language modeling is the possibility to be replaced by- or combined with- more advanced models, such as general a English unigram or bigram language model.

4. Experiments and obtained results

The features used for acoustic modeling are mel-frequency cepstral coefficients [8] and first and second order derivatives ($\Delta + \Delta\Delta$). As a basic robustness improvement, cepstral mean normalization [9] is applied. The mean of the cepstrum is estimated on the fly, from up to 600 previous frames. Triphones are used as the unit of acoustic modeling. The schema describing this feature computation pipeline is illustrated in the Figure 5.

These features are classified by a Gaussian Mixture Model, trained on data from the publicly available database Voxforge [10].

This set of features[3] is not too complex, which allows for fast computation and gives more relative importance to the language model, which is specific to the project.

As described in Section 2, this pipeline is enhanced by blocks handling push-to-talk (PTT) and timing information, but these do not directly effect the recognition.

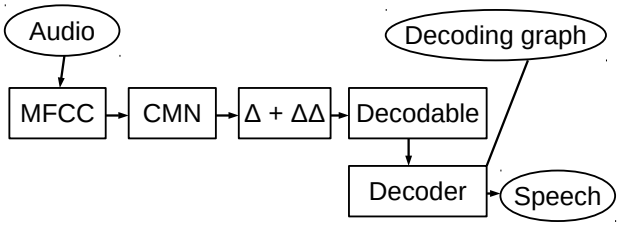


Figure 5. Pipeline for computing feature vectors. The audio input is read from a socket and the MFCC component first segments it into overlapping frames. Then the MFCCs are computed for each frame, the cepstral mean is subtracted and the feature vector is extended by first- and second-order derivatives. The likelihood of final features is used for scoring the decoding hypotheses. The CMN component estimates the cepstral mean from up to 6 preceding seconds.

At the level of A-PiMod project, there are several validations planned. The first validation, which took place in November 2014 in Braunscheig, Germany, included a session for testing the speech recognizer separately from other parts of the A-PiMod system. Each pilot read a list of 60 commands from the largest Interaction Context *Main Screen*. In case the system did not correctly recognize the command, the pilots were supposed to read the command once more, which has been agreed as a reasonable procedure.

The evaluation was conducted in a clean environment, because the simulator cockpit able to produce flight-like noise was unavailable for the validation. However, informal testing earlier in 2014 suggested, that a steady level of noise does not effect the recognition accuracy significantly, because the speakers in such environment tend to speak more precisely and louder.

The word error rate (WER) is used as the evaluation metric. WER is computed by aligning the decoder output with the reference transcription and then calculated as:

$$\mathrm{WER} = \frac{\#\mathrm{insertions} + \#\mathrm{deletions} + \#\mathrm{substitutions}}{\#\mathrm{words}}$$

[3] equal to the Kaldi recipe `tri_2a` system

The results of the Validation Cycle 1 speech recognition session are summarized in Table 1. The resulting WER of 57 % does not seem very satisfactory, but note, that the error rate is dominated by insertions. Insertions occur every time the pilot needed two attempts to get the command recognized correctly. I did personally inspect the recordings, and I have observed that the pilots are used to a slightly different form of the commands, which confused them sometimes.

Thus it is relevant to consider error rate without the insertions. In this case, the average WER is approximately 10 %. This is a satisfactory result, especially when we take the simplicity of the acoustic model into consideration.

Table 1. Word error rate for different pilots in Validation Cycle 1.

Pilot	WER	Insertions	Deletions	Substitutions
1	55 %	82	1	9
2	39 %	54	0	12
3	74 %	90	2	31
4	88 %	115	1	31
5	27 %	45	0	0

The pilots in the validation cycle 1 were also questioned about their feeling about the recognition latency. Results are captured in Table 2, the average result of 4.4/5 can be considered as a success.

Table 2. Satisfaction with recognizer latency, on the scale 1-5, where 5 is the best.

Pilot	Satisfaction
1	4
2	3
3	5
4	5
5	5

Considering the evaluation of the speech recognition system including the background model, only preliminary experiments were carried out. While the original system without background model added false-positive output to approximately 40 % of the utterances, this figure dropped to approximately 5 % when the background model was added.

These false-positive outputs are partly caused by input imperfections, mainly the microphone that captures loud breathing, and partly because of the on-line update of the cepstral mean removal – as the voice commands are separate in time, it may happen that an estimate from a previous utterance biases the current one.

5. Conclusions

In this paper, I have described the construction of a speech recognizer for the aeronautics cockpit using a research oriented speech recognition toolkit. Several issues are addressed, from push-to-talk voice activity marking through enhancing the result by wall-clock timestamps and response to external commands as well as a simple implementation of a background model.

The proposed system achieved a 90 % accuracy during a real experiment setup with actual pilots in a cockpit simulator. Given the insights learned from early stages of the project, the sensitivity to the most common noise was reduced from approximately 40 % false positives to only 5 % false positives.

Further work on the system will consist of training and plugging in a more advanced acoustic model and applying a more structured background model, which will allow for the recognition of out-of-grammar utterances as well.

Acknowledgements

This work was carried out within the Speech@FIT group. My thanks go to the common expertise shared by the group, to Honza Černocký for general guidance and to Mirko Hannemann for very valuable advice on language modeling and confidence estimation.

References

[1] Annual General Report 2013. Technical report, European Aviation Safety Agency, 2014. http://easa.europa.eu/system/files/dfu/TOAC14001ENN.pdf.

[2] The Commercial Aviation Safety Team. Cast Safety Enhancement Plan. http://www.skybrary.aero/index.php/Category:CAST_SE_Plan.

[3] A-PiMod: Applying Pilot Models for Safer Aircraft. http://www.apimod.eu/Default.aspx, 2013.

[4] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi Speech Recognition Toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[5] Mehryar Mohri and Fernando Pereira and Michael Riley. Speech Recognition with Weighted Finite-State Transducers. In *Springer Handbook on Speech Processing and Speech Communication*, chapter 28. Springer-Verlag New York, Inc., 2008.

[6] Karel Beneš. Finite State Grammars as Language Models for Automatic Speech Recognition. Bachelor's thesis, Faculty of Information Technology, Brno University of Technology, 2014.

[7] Igor Szöke, Petr Schwarz, and Martin Karafiát. Phoneme based acoustics keyword spotting in informal continuous speech. In *in Proc. RADIOELEKTRONIKA 2005*, 2005.

[8] Steven B. Davis and Paul Mermelstein. Readings in speech recognition. pages 65–74. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[9] Olli Viikki and Kari Laurila. Cepstral domain segmental feature vector normalization for noise robust speech recognition. *Speech Communication*, 25(1-3):133–147, August 1998.

[10] Ken MacLean. VoxForge. http://www.voxforge.org/home.