Automatic processing pipeline for collecting and annotating air-traffic voice communication data

Martin Kocour1, Karel Veselý1, Igor Szöke1,2, Santosh Kesiraju1, Juan Zuluaga-Gomez3,4, Alexander Blatt5, Amrutha Prasad3, Iuliia Nigmatulina3, Petr Motlíček3, Dietrich Klakow4, Allan Tart6, Hicham Atassi7, Pavel Kolčárek7, Honza Černocký1, Claudia Cevenini8, Khalid Choukri9, Mickael Rigault9, Fabian Landis6, Saeed Sarfjoo3, Chloe Salamin3

Citation: Kocour, M.; Veselý, K.; Blatt, A.; Szöke, I.; Kesiraju, S.; Zuluaga-Gomez, J.; Prasad, A.; Nigmatulina, I.; Motlíček, P.; et al.; Automatic processing pipeline for collecting and annotating air-traffic voice communication data. Journal Not Specified 2021, 1, 0. https://doi.org/

Received: Accepted: Published:

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Submitted to Journal Not Specified for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The speech-processing tools for air-traffic data could work better if we had a large amount of reliably annotated data. However, the collection and manual annotation of air-traffic data is slow and costly. The recordings are often noisy, accented and the speech is very fast. Moreover, for the downstream tasks, we also need to label the transcripts with the air-traffic related entities. In our case those are call-signs, commands and its values (i.e. arguments of the command). Plus, other complication is that the annotators need to be experts that understand the domain.

This work presents how the whole annotation process can be efficiently accelerated by using already existing machine learning concepts. The main focus is to improve the quality of the automatic transcripts for the annotators to help them work faster.
The previous EU project focused on ATC speech processing was MALORCA. It produced Active Listening Assistant (AcListen) [1], which used voice input for faster update of a plan in the approach planning system of an airport. However, the used speech-to-text module was tailored only to a particular airport, because the training data was collected from two airports only. If we collect training recordings from many airports, we believe our system will become more airport agnostic, which was a supportive argument for collecting the data in our ATCO² project.

The purpose of this document is to describe the final set of tools that allows us to pre-process and automatically transcribe the audio data that we collect in ATCO² project. The tools are based on techniques from Signal Processing, Automatic Speech Recognition (ASR) and Natural Language Processing (NLP), which get applied to the air-traffic recordings. The purpose is to speed-up the process of building a large air-traffic dataset. And, the project goals are to obtain 1000 hours of recordings with automatic transcripts, from which 50 hours should be manually corrected by the community. The data are planned to become accessible both for research and commercial use.

The overview of the data processing pipeline is in Figure 1. It consists of a) speech pre-processing tools (segmentation, volume adjustment, discarding noisy recordings), b) diarization (split audio per speaker), c) speech-to-text recognition, d) English language detection, e) call-sign recognition, f) ATCO – pilot classification and g) labelling of commands and values.

2. Data pre-processing and diarization

At the very beginning, the radio signal is captured by a community of feeders from Open Sky Network by their antenna and recording device. Since the radio broadcast is most of the time passive, i.e. the channel is silent, we apply adaptive energy-based segmentation to divide the signal into segments with a voice activity. The I/Q radio signal is converted to a waveform audio signal by a software defined radio (csdr) in the demodulation block.

**Volume control:** the gain of the signal is increased in volume control in case of a weak signal from a distant airplane. We noticed, that the speaker turns are separated by spikes, which arise from push-to-talk control of the radio communication. Our method first finds the positions of the spikes and then adjusts the volume of each segment separated by spikes with a different scalar value. However, this method does not ensure that one segment contains speech from a single speaker, as some spikes may not be detected.

**Signal-to-noise ratio based filtering:** our intention is to get rid of noisy segments that are unintelligible. However, our aim is not to discard all noisy segments as moderate
noise levels in some of the training data will make the speech-to-text system more robust to noise. For estimating the Signal-to-noise ratio (SNR), we first separate the speech and non-speech parts by a Voice Activity Detection tool (VAD) described in [2]. For the use in SNR filtering, we adjusted its hyper-parameters to ensure that almost no non-speech parts are marked as speech. Since the true non-speech segments, are not recorded because of push-to-talk, we estimate SNR from the distribution of the speech signal only, which we do with the WADA-SNR (Waveform Amplitude Distribution Analysis) algorithm [3]. This method is based on the assumption that the distribution of samples from a clean speech signal is a Gamma function with a predefined shaping parameter, and the distribution of samples from additive noise is Gaussian. The method should be reliable for SNR interval 0-20dB and produces estimates of SNR values. By setting an appropriate threshold, we can discard the audio data that are too noisy, i.e. the SNR value is too low.

**Diarization:** eventually, the segments are further divided into single-speaker segments by diarization. The subsequent NLP tasks like call-sign recognition or command extraction need to operate on a message that comes from exactly one speaker. So, in this sense, it is strategic to identify the speaker turns already before the automatic transcription is done. Our diarization is based on Bayesian HMM clustering (VBx) [4]. The diarization is also very useful for annotators, as otherwise they would have to divide speakers manually. Currently, we use diarization just to split speech into single-speaker segments. We don’t use diarization to separate ATCO – pilot speech, nor to track the utterances of the same speaker within or across recordings.

### 3. Automatic Speech Recognition

After the recordings are pre-processed, the segments are transcribed by speech-to-text system, that is also frequently called Automatic Speech Recognition (ASR). It is a key component in our pipeline, since it affects the performance of the downstream tasks, and we are also interested to deliver automatic transcripts of high quality. That is why we trained ASR system tailored for ATC domain. The performance of a COTS ASR would be poor on ATC data.

In this work, we used a standard hybrid speech recognizer, in which the temporal dynamics of speech are modeled by Hidden Markov Models (HMM). The recognizer consists of language model and acoustic model. The scores of the two models are combined together to obtain the best transcript of observed speech. The decoding itself is done with the token passing algorithm, which operates within a recognition network (HCLG graph). The HCLG graph is a WFST composition [5] of a graph with phone-level HMMs\(^1\) \(H\), context-dependency graph \(C\), pronunciation lexicon graph \(L\) and language model graph \(G\). The algorithm is using beam search heuristic to prune the improbable searched paths as the decoding progresses over time. All likely paths are then stored in a compressed format called lattice, i.e. a structure with timing information and pronunciation of each likely sentence. The final transcript of the segment is generated by taking 1-best hypothesis from lattice. The generated transcripts inevitably contain some errors (search errors, OOVs, etc.). Our goal is to build a system producing the least errors as possible.

For language modelling, we first created ATC text corpora from 7 ATC databases we work with (see Section 5.1 in [6]). We enriched the text corpus by list of call-signs gathered by OpenSky Network in years 2019 and 2020\(^2\). The call-signs are expanded into words by our verbalization tool, the expansion follows the ICAO standard [7] plus other common variants are generated. The expansion tool is described in our previous works [6,8,9]. We also collected a list of runways and air navigation waypoints that exist in Europe from Traffic [10], and expanded these into idiomatic contexts for language model training. It should ensure that the ASR is able to recognize all known call-signs,

\(^1\) Actually, we use bi-phones, i.e. a context dependent phonemes.

\(^2\) Crowdsourced air traffic data from The OpenSky Network 2020 : https://zenodo.org/record/5644749
waypoints and runways. The final language model is an interpolated 3-gram in ARPA format trained with SRILM toolkit [11].

For acoustic modelling, we train a CNN-TDNN-F [12] neural network model from Kaldi with Lattice-free MMI objective function. The input features are high-resolution Mel-frequency cepstral features (MFCC) with online Cepstral mean normalization (CMN). The features are extended with online i-vectors [13]. The model produces posterior probabilities of senones (states in the HCLG recognition network) that are used by the HMM decoder.

3.1. Call-sign boosting

In speech-to-text, we experimented with contextual adaptation. This improves the ASR performance by integrating rapidly changing textual context into decoding. Since we have access to various information about the processed communication (e.g. time, location of a receiver and ADS-B surveillance data), we are able to increase a chance of correctly recognizing some specific words, in our case call-signs.

The whole process of contextual adaptation is depicted in Figure 2. For each recording, we know location and timestamp. With this info, we query the OpenSky Network database [14] for a list of call-signs using pyopensky python interface [15]. Next, the call-signs are verbalized into all its possible word sequences. Then, two boosting graphs are built from the list of verbalized call-signs, one for HCLG boosting, one for lattice boosting. These graphs are further used during the decoding, as we recently described in [8]. Note that this boosting technique allows us to do on-the-fly contextual adaptation. We have chosen to boost the call-signs, since this is the most important entity in the communication. However, the described technique, can be applied to any other entity, e.g. runway number, frequency, etc.

First, we apply the graph for HCLG boosting. This is done by WFST composition of the recognition network (HCLG) and the boosting graph $B_{HCLG}$:

$$HCLG' = HCLG \circ B_{HCLG}$$  \hspace{1cm} (1)

The boosting graph $B_{HCLG}$ contains language model score discounts for particular rare single words, that are then transferred into the recognition network $HCLG'$. This causes that the paths with the desired boosted words become less likely to be pruned out during decoding. So more of the boosted words are present in the lattice produced by ASR. The boosting graph $B_{HCLG}$ is depicted in Figure 3a. Note that the topology of the HCLG boosting graph is simpler compared to lattice boosting graph in Figure 3b. This is needed for an affordable run-time of the composition with the relatively large HCLG graph.
Next, after the lattice generation is finished, we apply the lattice boosting of call-signs. Similar to HCLG boosting, the lattice boosting is done by WFST composition of a boosting graph $B_L$ with the ASR output lattice $L$:

$$L' = L \circ B_L$$  \hspace{1cm} (2)

The toy example of boosting graph is shown in Figure 3b, we are boosting whole word sequences like "c_s_a alfa bravo". The graph 3b contains the upper part that encodes the word sequences to be boosted in a particular segment. The language model score discounts $-4$ or $-6$ are on the word links. The lower part contains all words from lexicon, which ensures that no words are dropped by the composition. Also, the lower part is accessed only if the partial word sequence from the lattice cannot be matched with the upper part. The HCLG boosting is done before lattice boosting just to increase a chance that the lattice contains the boosted rare words. And, the lattice boosting increases a chance of the boosted word sequence to appear in the final 1-best hypothesis, i.e. in the final transcript.

The effect of call-sign boosting is shown in Table 1. The performance was measured on our internal LiveATC test set and our public ATCO² test set for which we have the surveillance data available. With boosting we reduced the word error rate (WER) by 12.5% relative on LiveATC test set and 6.5% relative on ATCO² test set. Note that we obtained better results with both HCLG and lattice boosting than with just lattice.

<table>
<thead>
<tr>
<th></th>
<th>LiveATC</th>
<th>ATCO²</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WER [%]</td>
<td>CA [%]</td>
</tr>
<tr>
<td>no-boosting</td>
<td>35.9</td>
<td>46.8</td>
</tr>
<tr>
<td>HCLG-boosting</td>
<td>35.4</td>
<td>50.0</td>
</tr>
<tr>
<td>lattice-boosting</td>
<td>31.8</td>
<td>70.2</td>
</tr>
<tr>
<td>HCLG + lattice-boosting</td>
<td>31.4</td>
<td>72.8</td>
</tr>
<tr>
<td>Oracle (correct transcripts)</td>
<td>0.0</td>
<td>89.6</td>
</tr>
</tbody>
</table>

Table 1: Performance improvements from various call-sign boosting strategies in speech-to-text. Measured on LiveATC test set and public ATCO² test set in terms of word error rate (WER) and call-sign accuracy (CA). [ATCO² test set: https://www.atco2.org/data]
# Table 2: Performance improvements from semi-supervised learning (SSL) and test phase call-sign boosting. Measured on LiveATC and ATCO² test set.

<table>
<thead>
<tr>
<th></th>
<th>LiveATC</th>
<th>ATCO²</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 seed-system</td>
<td>35.9</td>
<td>21.4</td>
</tr>
<tr>
<td>2 SSL + gradient weighting</td>
<td>30.6</td>
<td>18.6</td>
</tr>
<tr>
<td>3 (2) + lattice boosting</td>
<td>27.2</td>
<td>17.6</td>
</tr>
<tr>
<td>4 (2) + HCLG+lattice boosting</td>
<td>26.8</td>
<td>17.6</td>
</tr>
<tr>
<td>Oracle (correct transcripts)</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

boosting. The effect of boosting is even stronger for call-signs recognition accuracy (CA), where it removed roughly one third of errors. The call-sign accuracy improved by 26.0% absolute on LiveATC test set and by 8.0% on ATCO² test set.

## 3.2. Semi-supervised learning

We used our collected data for semi-supervised learning (SSL) experiments. This improved our acoustic model by adding the untranscribed data into the training. The untranscribed data were processed by our pipeline that filtered out noisy and non-English data. In SSL, a seed-system is used to produce automatic transcripts and confidences. We incorporated acoustic word confidences generated by Minimum Bayes Risk decoding of lattices [16]. The word confidence is a probabilistic value taken from a confusion network that has lists of candidate words for word slots.

We applied the word-confidences to discard 10% sentences with lowest mean confidence. Next, we discarded all the words with confidence lower than 0.5 (5% words). And finally, we used the word confidences to scale the gradients in the back-propagation training. We noticed that the confidences were biased towards high values even for incorrect words. To mitigate this, we applied power 4.0 which transformed the word-confidences to lower values. This is rather an empirical calibration, and we can afford to scale down some of the correct words from the automatic transcripts.

In our experiments we used 1190 hours of untranscribed speech, partly from ATCO² platform, partly from LiveATC. The results are shown in Table 2, for system 2 the WER was reduced by 14.8% for LiveATC and 13.6% for ATCO² test sets relatively. The combination of SSL and test phase call-sign boosting achieved the best performance of 26.8% WER for LiveATC and 17.6% WER for ATCO² test sets respectively. Currently, this is the best model we have. We further plan to start using call-sign boosting for automatic transcripts in SSL, as we previously tried in [17].

## 4. English language detection

We have developed and deployed a suitable English language detection system (ELD) [18] to discard non-English utterances in newly collected data. We tested an acoustic based system with x-vector extractor, but then we decided to use an NLP approach that processes ASR output with word confidences, as its performance was better. The NLP approach can jointly use outputs from several ASR systems, which further improves the results.

For the processing pipeline we integrated the NLP based English detector operating on Czech and English ASR. The integrated English detector consists of TF-IDF for re-weighting the accumulated soft word counts, and a Logistic regression classifier to get the English non-English decision.

For each recording we extract bag-of-words statistics from the automatic transcriptions generated by the ASR systems. We concatenate the word lists from lexicons of both ASR systems. And, the statistics are accumulated from the posterior probabilities in the bins of the confusion networks. The TF-IDF adjusts these per-word statistics by
Table 3: Performance of the English language detector with different training data (with Czech-English data CZEN, or with Czech-French-German-English data).

<table>
<thead>
<tr>
<th>ASR</th>
<th>Train data</th>
<th>Equal Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CZEN</td>
<td>CZEN FREN GEEN</td>
</tr>
<tr>
<td>EN+cz</td>
<td>CZEN</td>
<td>0.0470 0.2397 0.3433</td>
</tr>
<tr>
<td>EN+cz</td>
<td>CZEN+FREN+GEEN</td>
<td>0.0617 0.1338 0.2602</td>
</tr>
</tbody>
</table>

deweighting words that appear frequently in other documents (i.e. recordings). Finally, the binary logistic regression classifier decides between the English and non-English classes.

On our evaluation set of Czech-English examples (CZEN) we achieved Equal Error Rate of 0.0470 (see Table 3). By training on more languages, the CZEN equal error rate increased to 0.0617, but for French (FREN) and German (GEEN) the error rate got smaller. If we set a threshold 0.8, we can almost completely remove the non-English utterances, while discarding only a small amount of English data.

5. Post-processing by NLP/NLU

The goal of the Natural Language Understanding (NLU) part is to extract knowledge from the text produced by the speech-to-text system. In ATCO we focus on these tasks:

- Call-sign recognition (i.e. locate the call-sign, convert it to code like “DLH81J”)
- ATCO – pilot classification (i.e. decide who is speaking in the entire utterance)
- ATC-Entity recognition (i.e. highlight the call-sign, command and value in text)

These tasks were selected in cooperation with our industry project partners (Honeywell, Airbus).

5.1. Call-sign recognition

For call-sign recognition, we use a completely neural-network based end-to-end model, which is based on BERT [19]. The model extracts the call-sign codes (e.g. DLH81J) from the speech-to-text output and the corresponding list of surveillance call-signs. We found, that this model architecture outperforms cascaded systems, where in the first step a named entity recognizer tags the call-sign in the transcript and in the second step a call-sign mapper converts the call-sign to the corresponding ICAO format.

The performance of our fully-neural call-sign recognizer was previously shown in Table 1 and Table 2. The performance is measured as call-sign accuracy (CA). We also see that integrating contextual information into speech-to-text engine via call-sign-boosting was essential to achieve good call-sign recognition results.

5.2. ATCO – pilot classification

As part of labelling the data, we aim to automatically classify the utterances based on whether it’s the controller or the pilot speaking. The classifier is trained on top of the ASR-output, since there exists some disjoint vocabulary between the pilot and the controller.

The classification experiments were done using three classifiers. The first one uses a TF-IDF per-word statistics followed by a binary logistic regression, the second one is

![Figure 4. End-to-end model for call-sign recognition, the inputs are speech-to-text output and surveillance call-signs (list of plausible call-signs from the OSN database).](image-url)
### Table 4: Performance of ATCO – pilot classifier.

<table>
<thead>
<tr>
<th>Model</th>
<th>Classification accuracy ATCO²</th>
<th>LiveATC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF + LR</td>
<td>77.8</td>
<td>76.6</td>
</tr>
<tr>
<td>CNN (no pre-training)</td>
<td>80.2</td>
<td>82.2</td>
</tr>
<tr>
<td>BERT (pre-training + fine-tuning)</td>
<td>91.0</td>
<td>87.0</td>
</tr>
</tbody>
</table>

### Table 5: F1 Scores - on LiveATC and ATCO² test sets.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Call-sign</th>
<th>Command</th>
<th>Value</th>
<th>Unknown Phraseology</th>
</tr>
</thead>
<tbody>
<tr>
<td>LiveATC</td>
<td>80</td>
<td>52</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>ATCO²</td>
<td>89</td>
<td>77</td>
<td>68</td>
<td>57</td>
</tr>
</tbody>
</table>

5.3. **ATC-Entity recognition**

Breaking down a transcript in its different components, respectively entities, can be implemented as Named entity recognition (NER) task. The entities of interest are call-sign <CAL>, command <COM>, value <VAL> and unknown phraseology <PHR>. The NER architecture is similar to Figure 4, but the output are NER tags in IOB format.

An example of a tagged transcript:

<COM> CLIMBING TO </COM> <VAL> FLIGHT LEVEL SEVEN ZERO </VAL>
<VAL> OSCAR KILO TANGO UNIFORM ROMEO </VAL>

We are currently building a database for training an ATC entity recognition network. Table 5 shows first results for a train|val|test split of 300|100|100 for the LiveATC and ATCO² test set.

6. **Conclusion**

We have successfully created an operating pipeline for processing and automatically annotating the ATCO and pilot air traffic control audio data. The pipeline discards noisy and non-English data, and generates automatic transcripts from which it extracts a call-sign code. Later, we will be able to automatically decide if it is ATCO or pilot speaking, and highlight entities in the text (call-sign, command, value). The purpose of ATCO² project is to collect a large database of ATC audio data, that will help develop better voice tools. Perhaps one day the voice tools will finally serve pilots and ATCOs.


**Funding:** This research was funded by European Union’s Horizon2020 project No. 864702 - ATCO².

**Data Availability Statement:** In this work we used 7 publicly available data, which we described in our previous work [6]. ATCO² data are freely available on [https://www.atco2.org/data](https://www.atco2.org/data).

**Conflicts of Interest:** The authors declare no conflict of interest.
References


