Chapter 2

Automatic Speech Recognition and Understanding

Either a speech recognizer can be treated as one big Machine Learning (ML) problem or it can be broken down into an ML problem for the so-called acoustic model (AM) and a separate ML problem for the so-called language model (LM) [10].

The output of the usage of the acoustic model is a sequence of phonemes and together with the language model we get a sequence of words, e.g.:

el al five zero nine two climb flight level one six zero proceed direct victor oscar zulu

This is called *speech-to-text* (S2T or STT), which, however, is only one part of the main task. As already Alan Turing pointed out in the early 1950s, speech recognition is **not** speech understanding.

The output of the following word interpretation is then a sequence of concepts (task *text-to-concepts*, TTC or T2C), which are further combined to commands. In the example case above the expected sequence of commands would be: [1]

ELY5092 CLIMB 160 FL
ELY5092 DIRECT_TO VOZ none

The semantics of these commands are: The aircraft with the callsign ELY5092 is now starting to climb or may continue its climb to flight level 160, which is approximately 16'000 feet.[2]. Furthermore this aircraft should fly to the direction of the waypoint VOZ, which is a waypoint whose coordinates are defined and all pilots are knowing them. *n*one means that the direction of the turn, i.e. left or right, is not specified in this sequence of words.

[1] A special set of rules (i.e. an ontology) was proposed by different European partners within the SESAR (Single European Sky ATM Research, ATM = Air Traffic Management) funded solution PJ.16-04 [9], how a sequence of words of an ATCo utterance should be transformed into commands.

[2] 1 foot is 0.3048 meter, i.e. 16,000 feet is depending on the current air pressure approximately 4900 meter

The command extraction models splits the sequence of words into concepts, e.g:

```
<callsign>el_al five zero nine two</callsign>
<command>climb flight level one six zero</command>
<command>proceed direct victor oscar zulu}</command>
```

These tags can be further split into:

```
<callsign>
    <airline>el_al</airline>
    <identifier>five zero nine two</identifier>
</callsign>
<command>
    <type>climb</type>
    <unit>flight level</unit>
    <value>one six zero</value>
</command>
<command>
    <type>proceed direct</type>
    <value>victor oscar zulu</value>
</command>
```

Although the callsign appears only once in the utterance the ontology requires that it is repeated in each command for an aircraft. The main elements of the ontology are: `Callsign` and `Instruction`.

Figure 2.1: An `Instruction` consists of a repetition of callsigns and instructions

An `Instruction` itself always consists of a command (darker green part is mandatory in Fig. 2.2) and one or more optional (orange) conditions described in detail later. A command is composed of a type, one or more values and a unit (e.g. FL or ft or none). Then an optional qualifier follows (e.g. LEFT, RIGHT, OR_LESS, BELOW). The categories *Speaker* and *Reason* are introduced to be able to annotate also utterances from the pilot.

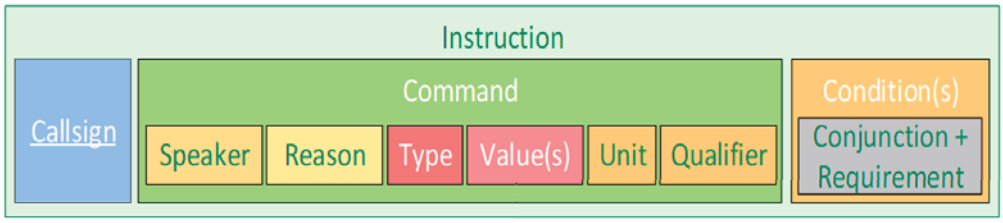

Figure 2.2: An instruction consists always of a type, the following tags are type depending

Fig. 2.3 shows vertical commands types.

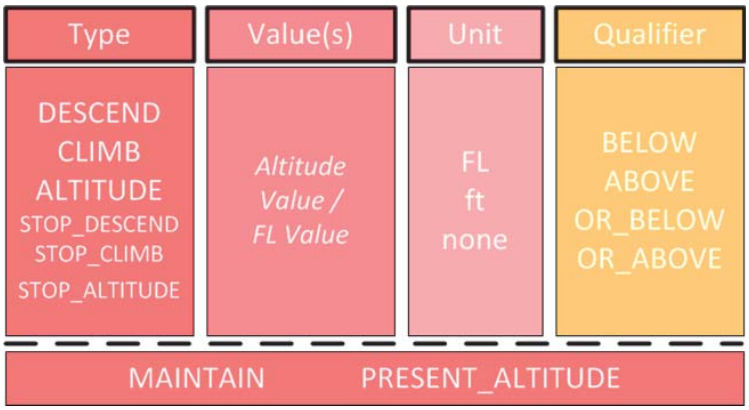

Figure 2.3: Vertical types of the ontology

Besides vertical command types approximately 15 other command type categories are defined by the ontology (e.g. speed commands, horizontal commands). The phraseology an air traffic controller (ATCo) should use, i.e. the sequence of words. is standardized by ICAO. If ATCos strictly follow these rules, which are defined by ICAO, the transformation into concept would be easy and could be easily performed by a context free grammar. The problem is as always what human should do and what they do.

The following utterances result all into the same two commands, which are shown already above:

- al al five zero nine two climb flight level one six zero proceed direct victor oscar zulu,
- al al nine two climb flight level one six zero proceed direct victor oscar zulu,
- zero nine two climb level one six zero proceed direct victor oscar zulu,
- al al five zero nine two aeh climb level one six zero proceed direct victor oscar zulu,[3]
- al al five zero nine two climb one sixty proceed direct victor oscar zulu,
- al al five zero nine two climb one sixty proceed cheb,[4]
- al al five zero nine two climb one sixty direct to cheb,
- al al five two correction five zero nine two descend one fifty correction climb aeh one six zero direct aeh to cheb[5]
- and other sentences

Although some of theses sentences are not allowed by ICAO they are used in daily life. We could still model them by a bigger context free grammar. The problem, however, is that we are not really knowing, which sequence of words a controller is

[3]The hesitation word aeh could also be pause and if of course could occur also everywhere in a sentence.

[4]cheb is the pronunciation of the waypoint VOZ, which is normally used by a Prague controller. Only for pilots who seldom fly to Prague victor oscar zulur is said, but both mean the same.

[5]Although ICAO clearly defines that the keyword correction must be used, some correction occur without using the keyword correction.

using. This makes the difference between a good and a very good speech recognition and understanding system (ASRU).

Our final objective of this exercise is that we want to be able to automatically learn rules how to extract a climb command. At least we want to be able to extract some climb commands (and other types) from given utterances, with a context free grammar or with other hard coded rules. To make it even more complicated, we do not only want to extract the rules, from correctly transcribed word sequences, but these word sequences can also result from a speech recognizer and the output of the speech recognizer is not manually corrected and even manual corrections could be wrong. In other words, we are also learning from bad examples, not knowing that they are bad examples.

The problem is also investigated by two master theses supervised by Prof. Helmke. The work of Gusain [4] uses an ELMO based model extracting eight different command types and the corresponding values. One big model is trained to extract the different command types in one step. Müller [13] uses BERT to extract 21 different command types and the corresponding values. Different models, i.e. binary classifiers, are trained for each command type

2.1 Previous Exercises

The final goal of the exercises in WS 2020/21 was that you implement a software, which

1. extracts all numbers from a given utterance, e.g., *adria nine four eight speed two hundred descend three thousand feet cleared ils three four* should result in
   - 200 from *speed two hundred*,
   - 3000 from *descend three thousand feet*,
   - 34 from *ils three four*.

   The numbers of the callsign should be ignored in this output, i.e. 948 from *nine four eight*.
2. extracts the word sequences and command types, resulting in a command type. In the example *adria nine four eight reduce speed two hundred descend three thousand feet cleared ils three four* the following output is expected:
   - *reduce speed* results in *REDUCE*,
   - *descend* results in *DESCEND*,
   - *cleared ils* results in *CLEARED ILS*.

In WS 2022/23 the final goal of the exercises was to implement a software, which extracts all (many) callsigns, command types and all (many) values from a given set of utterances.

2.2 Current Exercise

This time our goals have changed, again. We just want to extract the callsigns from spoken utterances.[6] The utterances are the output of a speech recognizer, i.e. some words itself might be wrongly recognized. `for` instead of `four` is one of the simpler cases. You will get a lot of information, how to do this task. The challenge here will be that you do it fast, i.e. you will implement heuristics, which sometimes will miss the correct callsign.

The output for the above example *al al five zero nine two climb flight level one six zero proceed direct victor oscar zulu* should be then: `ELI5092` The same output is expected for *al al five zero nine to good morning* or for *five zero nine to good morning*. In the later two cases you of course need the information, which callsigns are currently in the air.

Your input would be a list of three letter codes in JSON format for the callsigns, e.g.

```
{
   "ELP": [ "aerosanluis" ],
   "ELS": [ "el sal" ],
   "ELT": [ "elliott" ],
   "ELU": [ "egyptian leisure", "egyptian" ],
   "ELV": [ "aereos selva" ],
   "ELW": [ "yellow wings" ],
   "ELX": [ "elan" ],
   "ELY": [ "el al" ],
   "ELZ": [ "hopper" ],
   "EMB": [ "embraer" ],
   "EMD": [ "eaglemed" ],
   "EME": [ "emair" ],
   "EMI": [ "blue shuttle" ],
   "EMJ": [ "light house" ],
   "DLH": [ "lufthansa", "hansa", "german lufthansa" ]
   }
```

The full lists you will never get, but your code is tested/evaluated against the full list, run on the computer of Prof. Helmke, but you will get a subset of the full list and also test data.

The final goal (dream) of the exercise, is that you get 500 test examples[7] and you are able to correctly extract more than 90% of the callsigns (e.g. ADR948).

More details and how the dream can be iterative *attack* are presented in the next chapters.

[6] More challenging tasks could be continued in a seminar study, in a project study and finally (or only) in a master thesis. Your ideas are welcome.

[7] You do not really get them, but your software on my computer is tested with them.