Extraction of Waypoints

Table of Contents

Learning Objectives

- Using STL
- Programming in a team, you may work in teams up to 5

Exercise at a glance

- Implement functions to read from JSON files
- Create an evaluation function for Word Error Rate calculation
- Implement a waypoint (and correction) extraction function
- Implement a more efficient waypoint extraction

Detailed Exercise Description

Exercise 3-1: Who is doing what in your team

Send a short description, who will do what in your team, if you are working in a team. If you are working alone this is not necessary. It is currently enough to send a plan just for the part of the exercise described in this file. Deadline 12.11-2025.[1]

Exercise 3-2: Simple extraction of callsigns

Extract waypoint names from given utterances from communication between air traffic controllers and pilots:

Examples are[2]

- "lufthansa seven one two proceed direct to geska" (GESKA)
- "lufthansa seven one two hello identified maintain flight level three three zero proceed direct to geska" (GESKA)
- nor shuttle five eight romeo maastricht identified continue direct to maseg (MASEG)
- confirm euro wings three november kilo is proceeding to Magdeburg (MAG)
- easy one one zulu proceed direct oscar sierra november (OSN)
- easy one one zulu proceed direct osnabrueck (OSN)
- easy one one zulu proceed direct osnabrueck oscar sierra november (OSN, and only once)
- speed bird six five tango cleared to pebep(PEBEP)
- i say again you are cleared to laret i spell lima alfa romeo echo tango (LARET and only once)
- gogsi direct speed bird six two four (GOGSI)
- london good day fraction five one nine uniform descending flight level one two zero inbound lydd (EGMD)

A subtask might be to decide whether the given utterance contains a waypoint or not. Most of the utterances do not contain waypoints.

For this task you do still not need the Levenshtein distance. You just need to know which spoken word sequences correspond to waypoint names (the green string) and what is the resulting waypoint name concept (marked in yellow above). You find the mappings in the file AtcConcepts.json, which is described below.

Exercise 3-3 Use Context information

In this exercise you will benefit from your Levenshtein distance implementation.

Use the information of the allowed waypoint names. This could be more than 600.

In the previous exercise we used the output from a perfect speech recognizer, which does not exist in the real world. So, your algorithms need to handle these errors.

- "eight four six three roger and turn right direct to lydd", but recognized was "fource xix three yeah roger turn right direct to lid" (EGMD)

[1] It is mandatory to send that list. You can change it at any time, but you should have already now an idea, what you want to do (and I can give then early feedback whether you are working in the right direction or not).

[2] In green we mark the waypoint names and in yellow we mark the requested output. You get a mapping table from words to the waypoint names: e.g. OSN for osnabrueck or oscar sierra november.

- "speed bird eight seven papa charlie route directbenbo" but correct would be "peed bird eight seven papa charlie route direct benbo" with a blank
- direct bebos ebi speed bird eight seven papa charlie, but correct is "direct benbo sp* speed bird eight seven papa charlie" ("sp* means that the word starting with "sp" is not fully spoken).
- speed bird eight papa whiskey resume own navigation sitet and contact london one three five decimal zero five five good bye, but "citet" is recognized.
- climb level two
- one zero direct lelna easy eight seven yankee Quebec, but "lelm" recognized.
- "descending altitude four thousand level evata fraction two one two romeo", but we recognize "descending alposute four thousand level evato fraction two one two romeo". This example shows that there could be also problems with other words.
- "dark knight four charlie direct benbo climb flight level one seven zero", but we recognize "dark knine four charlie directbenbo climb flight level one seven zero"

Use your Levenshtein algorithm to extract the correct waypoint name and to correct the recognized utterance.

Exercise 3-4 Implement a faster Levenshtein distance calculation

The number of allowed waypoints could be quite high, e.g. in the order of 600.The complexity of the Levenshtein distance algorithm is O(n*m), where n and m are the lengths of the two words. In most cases, n and m are equal to 5, but you have seen examples with 10.

Comparing 600 words could require a lot of processing time and if you also want to correct other word errors, you need to find some clever heuristics.

Evaluation criteria of your work

This will be a combination of how many correct waypoints you find and the how fast you are.

Therefore, first implement a naïve and slow, algorithm for finding the best Levenshtein distance and then concentrate on a good heuristic, ignoring first to generate all the word sequences for a callsign.

Your algorithm is evaluated on known utterances (test cases, which you get from me in advance or after some weeks), but also on unknown test cases, which I have on my computer. You just get the results of your algorithm.[3]

You will get directories with many (> 1000) files.
Each main directory contains a file with the name AtcConcepts.json and then some subdirectories follow. The subdirectories contain files with really spoken word sequence and the corresponding different outputs from different speech recognizers.

[3]The document does not really specify a complete interface. Try to early implement a framework which solves the task in principle, so that we together can tests whether your implementation might work on the test data.

Structure of AtcConcepts.json

- This is a JSON based structure. Important are the keywords “Name”, “KeywordSeq” and "ConceptType". The other elements you can ignore. “Name” is the output of the AtcConcept, which you should output, i.e. the yellow part in task 3.1.
- “ConceptType” specifies, whether it is a waypoint or not. You only need to consider the waypoints.
- “KeyWordSeq” is an array, containing all the words which could be spoken for this name. If the name consists of five letters, then you find also this word here. If not, please add.

Example:[4]

```
…   "AtcConcepts": [
{
 "Name": "GEVNI",
 "Locator": "LOWW",
 "KeyWordSeq": [
  "gevni",
  "golf echo victor november india"
 ],
 "CommandTypes": [
  "DIRECT_TO"
 ],
 "ConceptType": "WAYPOINT",
 "AdditionalInfo": [
  {
   "LatLong": "53.76166666666667 13.481666666666667"
  }
 ]
},
{
 "Name": "MTR",
 "Locator": "LOWW",
 "KeyWordSeq": [
  "mtr",
  "mike tango romeo",
  "metro"
 ],
 "CommandTypes": [
  "DIRECT_TO"
 ],
 "ConceptType": "WAYPOINT",
 "AdditionalInfo": [
  {
   "LatLong": "50.276666666666664 8.848333333333333"
  }
 ]
},
```

File extensions of subdirectories.

The following figure shows possible file name extensions which you find in the subdirectories.

[4] More information you can find in the real files, which I will upload together with this file as a zip file.

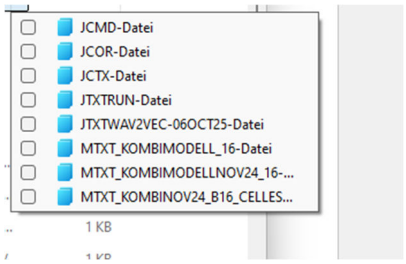

Figure 1 Possible file name extensions

- The jcor files contain, what was really said.
- jcmd files contain the semantic interpretation of the jcor file. Here you can also find examples for DIRECT_TO commands.
- jctx files you can ignore or use. It contains a list of callsigns which are currently in the air. Some callsigns are missing, i.e. a callsign like DLH12D for Lufthansa one two delta is said, but is not in this file.[5]
- jtxt* or mtxt* files contain the recognition of different speech recognition engines of from different parameter settings. The contents of these files are the input of your algorithm(s) to correct waypoint names. You can compare the original jtxt* or jmtxt* files against the jcor files, i.e. calculating the word error rates (WER). The WER should improve when you correct wrongly recognized waypoint names.

Structure of jcor,jtxt*, jmtxt* files

```
{
   "filename": "2018-01-01__11-00-00-00_P.wav",
   "abstraction_layer_word_sequence": "hello hel al zero zero eight level three nine zero"
}
```

The “filename” keyword you can ignore. The contents after "abstraction_layer_word_sequence" contains the real word sequence of the recognized one.

In some cases, the abstraction_layer_word_sequence can be an array which contains the speaker and the utterance.

```
{
  "filename": "2018-01-01__20-05-26-06_P.wav",
  "abstraction_layer_word_sequence":  [
    { "recog_speaker": "Pilot", "recog_utter": "radar good day euro wings six charlie kilo three six zero inbound diton" }
  ]
}
```

If the filename contains a “_P” at the end, you can assume that the utterance originates from the pilot. Pilot utterances are mostly shorter and the speech recognition engine has more problems, i.e. the WER is higher.

[5] For some files the jctx files are missing. Maybe they are missing for all files of a directory. Then you find the callsigns in a file Dummy_Context.jctx, which contains the callsigns for each spoken utterance. Sometimes even the Dummy_Contxt.jctx is missing. Then you have no callsign information.

Word error rate (WER)

The WER of a recognized word sequence compared to the spoken utterance is defined as the Levenshtein distance divided by the number of words in the spoken utterance.

The following figure shows an example of a spoken utterance (gold utterance) and the recognized utterance. The WER is here 16.7%.

```
gold utterance               :i     call you back for climb
recogn utterance  *##* (LD  1):roger call you back for climb  //  subs: 1 / ins: 0 / del: 0 words: 6
WER: 0.166667
```

Task you need to program

You find here a lot of tasks, but you are in a team, so find a way to structure the whole task. Mr. Schmidt and Prof. Helmke of course can help you.

Reading json file

- You need to read the elements from the AtcConcepts.json files
- You need to read the elements from the jcmd files. (considered that you might have different structures especially for _P-files
- You need to be able to read from the jcor, jtxt* and mtxt* files.

Implement Levenshtein distance class as a template class

The following tasks requires to calculate word error rates of strings. Currently your algorithm only handles to calculate the Levenshtein distance between sequence of characters.

Calculation of WER for a given directory

This function or program gets as input a directory name and the extension of the files, which contain the recognitions, e.g. jtxtrun. The output should be the Word Error Rate for the whole directory.

Do not forget to implement a sufficient number of tests. You need test directories with e.g. 3 to 10 files.

Classification whether an utterance contains a waypoint

Output of corrected Waypoints

This function or program gets as input a directory name and the extensions of the recognized files. The output is the copy of the corrected files. The extension is get the suffix “_c”. So, if the input extension is jtxtrun the output extension is jtxtrun_c.

If you do not correct words of a file or the file does not contain a waypoint, nevertheless create a corresponding jtxtrun_c file, which has the same contents as jtxtrun file.

In this case, you can just use your program/function to calculate the WER of the new files. The word error should improve.

If you try to correct other words, also put them also in the same file jtxtrun_c[6], [7].

The output of corrected JSON file should have the following structure:


```
{"filename" : "2025-04-28__11-08-27-46_P_c.wav",
"old_word_sequence" : "delta echo lima papa alfa is cleared to paderbon via brava whiskey departure flight land two five six three yankee",
"abstraction_layer_word_sequence" : "delta echo lima papa alfa is cleared to paderborn via bravo whiskey departure flight land two five six three yankee",
"corrections": [{"wrong": "paderbon", "correct": "paderborn"}, {"wrong": "brava", "correct": "bravo"}]
}
```


The input file could have been


```
{
   "filename": "2025-04-28__11-08-27-46_P.wav",
   "abstraction_layer_word_sequence": "delta echo lima papa alfa is cleared to paderbon via brava whiskey departure flight land two five six three yankee"
}
```


If no words are corrected, then input file and output file are the same. You may change the filename to the version with "_c".

Correction Algorithms

Implement different versions of the corrections algorithm. Start with a slow approach and implement faster versions or version with more functionality step by step.

[6] Just an example of the extension to which you add „_c".

[7] Use your WER calculation algorithms and visually check the substitutions and errors. Missing or wrong words in callsigns are candidates, also words of a STATION or CONTACT command (see jcmd files, to find out, what are STATION and contact commands). If you want to attack the callsign problem, maybe you need a list of the callsigns used, like lufthansa "air nippon", etc.