17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 362

Evaluation of approaches for the recognition of semantically similar voice-commands based on public data

Tobias Kolb[a*], **Falk Schiffner**[a]

[a] *German Aerospace Center (DLR e.V.), Space Operations and Astronaut Training, Berlin, Germany, [tobias.kolb, falk.schiffner]@dlr.de*
[*] *Corresponding Author*

Abstract

The new focus of the international community to return to Moon [1] and Mars has sparked the need for new ideas of future manned missions. Unmanned vehicles are being considered to aid astronaut activities on extraterrestrial surfaces [2]. This paper is exploring the concept of controlling unmanned ground vehicles (UGV) with voice commands by evaluating existing technologies; specifically, DeepSpeech (DS) [3] and its pre-trained v0.6.0 model. The evaluation entails the capability to transcribe spoken English and distinguish edge cases of discrete commands. It will be discussed in depth how the model behaves for various categories of edge cases, including anagrams, rhymes, default commands, and coincidental inclusions. Moreover, technologies like natural language processing and machine learning will be discussed regarding functionality and benefits for command recognition in post-processing as enhancements to plain speech to text conversion for building a voice command interface for an unmanned ground vehicle.

Keywords— speech-to-text, interface, machine learning, voice data

Acronyms/Abbreviations

German Aerospace Center (DLR), unmanned ground vehicle (UGV), DeepSpeech (DS), natural language processing (NLP), Speech-to-Text (STT), voice user interface (VUI), part-of-speech (POS), recurrent neural network (RNN), pulse code modulation (PCM), inversion of control (IoC).

1 Introduction

1.1 Motivation

The German Aerospace Center (DLR)[1] and other interest groups are sharing the vision of future manned missions to moon and mars. To aid these missions, new technologies are being developed and tested by space agencies around the globe. One use case is an unmanned ground vehicle (UGV), which supports astronauts when exploring planetary surfaces. Controlling these is traditionally being done with joysticks or pre-programmed routines. The controls could be extended by using speech input when the astronaut is on the surface and in line of sight of the UGV to free his hands.

[1] Deutsches Zentrum für Luft- und Raumfahrt e.V. :`https://www.dlr.de`

1.2 Work Objective

This work aims at a subset of a voice-user-interface (VUI). It will specifically investigate transcription failures in speech-to-text (STT) technology when used for command recognition. The main focus of the work is to analyze DeepSpeech (DS) [3] as the systems speech-to-text engine and its capability to distinguish between edge-cases of a discrete command set. The test cases, to validate the functionality of this command set, can each be classified as one of these basic commands: *forward, backward, left, right, stop*. Voice transmission and voice interface control is beyond the scope of this work. The VUI should understand different synonymous ways of issuing the same command to give users a degree of freedom when operating the vehicle. These commands should come from natural language, instead of a limited discrete command set. Hence there is no prefixed trigger word before a command to make the VUI expect a command. This requires software with semantic understanding, in order to identify sentences with the same content e.g. *move left*, *to the left* and *go left* should all be identified as the command *left*. In this respect it is also important to test for false positives, like *We left the base*, *Nobody gets left behind*, *You are right* or *Our mission is very straight forward*. Natural language processing (NLP) can help break down spoken sentences and analyze their meaning. To validate the effectiveness of the approach, voice data was collected via a survey. This survey consisted of standardized, simulated command scenario edge cases. Furthermore, a command recognition rate will be measured to benchmark the capabilities of the transcription engine model. STT capabilities were implemented using Mozilla's[2] STT engine DeepSpeech, whose pre-trained models are trained using Mozilla's Common Voice [4] data-set. This data-set was chosen because it is highly diverse, improving the likelihood of speaker independence. An over-representation of male over female voices could, e.g., result in a higher receptiveness for deep voices when interpreting commands.

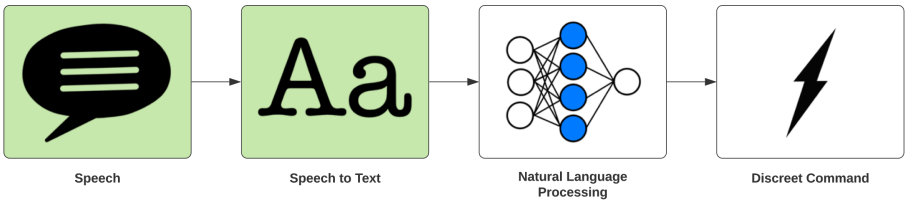

Figure 1: Focus of the recognition pipeline

Concluding this overview, Fig. 1 is showing the stages of the pipeline. The first two stages of the pipeline are describing the voice sample generation and STT transcription. The final two stages of the pipeline are the processing stage to classify the transcribed input and the action phase, resulting in a concrete reaction to the transcribed and interpreted command.

1.3 Proceeding and Structure of the Work

The first part of this work outlines core technologies and their use-case for command recognition. As described in subsection 1.2, command recognition within this work is regarded as a multi-step process: recognize the human voice, identify commands, and turn them into actions. In the second part we will focus on the pipeline design, data acquisition and testing methods. The

[2]Mozilla: `https://www.mozilla.org`

last part will present the test result and performance evaluation speech-to-text transcriber. As a basis for this evaluation, the commands from subsection 1.2 were developed into test cases. It is importants to understand that these test cases are designed to highlight the potential limitations of the transcriber and should be understood as edge cases. They were classified into ten categories. In order to practically test the STT transcriber, recorded test data was required. Due to the unique requirements for the data, it could not be provided by open data. As a consequence, a survey was designed and conducted. Key steps of the proceeding therefore are:

1. Theoretical formulation of a command recognition pipeline.

2. Conduction of an survey to raise voice recordings, matching criteria to evaluate specific edge cases for the pipelines transcriber.

3. Testing the transcriber model to evaluate its effectiveness with the inquiries data and discussing the results.

1.4 Technologies

The concept description for a voice command recognition interface as outlined in 2.2 requires some core technologies, described here.

1.4.1 DeepSpeech

DeepSpeech is an open source STT engine, implemented in TensorFlow[3]. It is the implementation of a recurrent neural network (RNN) model, utilising beam search and is based on a research paper by Baidu[4] [3]. The tooling surrounding the model characterizes the Mozilla's STT engine and sets it apart from Baidu's research. The tooling gives users access to analytical features like recognition confidence scores for individual transcriptions and an easy setup process, including pre-trained models.

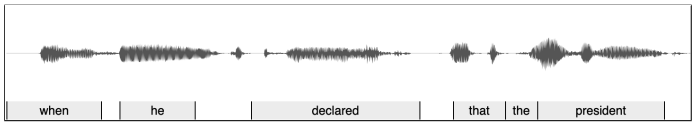

Figure 2: DS Transcription example [5]

Furthermore, the library provides the ability to retrain models. Data for the pre-trained model is provided by Mozilla's Common Voice database, which allows access to labeled voice samples. DS fundamentally works by recognizing characters in an audio stream and matching a waveform signature with learned signature-character pairs using its acoustic model. The recognized waveforms are approximated, and therefore all returned characters have a certain probability. Fig. 2 shows how the DeepSpeech decoder serves the acoustic model with audio features, which then get transcribed into strings.

[3]Tensorflow: `https://www.tensorflow.org`
[4]Baidu: `https://www.baidu.com`

1.4.2 Common Voice

One core problem in voice recognition software is the lack of suitable training data. Common Voice[5] [4] is a crowd-sourcing project aiming at solving this problem with its growing database of publicly available voice data. These types of data-sets are often lacking in diversity [6]. Mozilla describes Common Voice as a data-set that is more diverse than others. Common Voice provides the training data pre-trained DeepSpeech models.

2 Analysis

This section discusses how testing and data collection can be broken down into specific requirements. subsection 1.2 described the subject of this work from a use-case-oriented standpoint. It is subsequently necessary to break down the problem into manageable parts.

2.1 Requirements analysis

When the system receives a voice command, it need to be correctly transcribed. Therefore the system is benchmarked by how well it performs. Performance is defined as how well a command is translated into the user's expected action within a defined skill-set. Discrete commands are surjective command event pairs. Therefore every command must result in a defined state and subsequent action. Processing a voice command takes multiple steps:

1. Analog human voice is digitized to pulse code modulation (PCM).
2. Digitised voice data needs to be interpreted in a way, that differentiates human voice from acoustic noise. Noise will be specified as everything that is recorded alongside a command, but is not actually part of the command and therefore needs to get filtered or ignored.
3. After validating the data, the voice sample needs to get transcribed, using a speech-to-text engine.
4. The transcribed data can then refined with natural language processing to normalise the input and map it to discrete commands.

2.2 Concept

Requirements, as specified in subsection 2.1 outline two specific stages that a user-issued command needs to pass through, in order to get interpreted. Fig. 1 shows the stages a voice voiced command needs to pass through, in order to become actionable. The user's voice needs to get parsed into string format. This string is interpreted and processed in order to get classified as a specific command. The main reason for including a NLP stage is to interpret and process the output of the speech-to-text transcriber utilizing normalization and part-of-speech (POS) tagging. At this point, it is also unclear how well the speech-to-text engine processes edge-cases like similar sounding words, anagrams, words that rhyme or include a word co-accidentally. The inclusion of a natural language processing component provides the freedom to handle such cases while also providing a platform for later analysis and customization of the project for different use cases. The general advantage of machine learning versus hard rules is that machine learning models have the option

[5]Common Voice: `https://commonvoice.mozilla.org`

to get trained for different domain-specific scenarios and can get fine-tuned for the task at hand. The concept for achieving this goal is structured in the following way:

1. Proof of concept that DS is able to transcribe given data formats
2. Implementation of a pipeline to processes given text samples
3. Conceptualising a survey to raise data in order to test edge cases
4. Using the data to evaluate the DS model
5. Analysis of DS viability for future applications and how the model can be used in the pipeline

2.3 System boundary

As the system is getting tested for a limited number of commands, it will lack scalability in interpreting broad command sets. The system's reliability can only be bench-marked by how well it recognizes commands. To the current scientific understanding, the human voice is a unique biometric marker [7] and therefore testing the system implies inputting a sufficient quantity of voice data in the form of labeled, pre-recorded, or live spoken commands. The voice interface will then interpret the samples, enabling observation of the systems performance in terms of recognition rate and transcription quality. As a result, the system is constrained to a limited number of commands and tested for a specific use case. If the scope of possible commands expands, so does the cost of testing, and therefore the reliability is expected to drop. Another limitation of the system is that it is not designed to interpret commands outside discrete commands while in production.

2.4 Implementation

DS is available as a Python package[6]; therefore, the implementation is also in Python. The applications [8] architecture has been centered around the inversion of control (IoC) principle [9]. The application is designed to import generic versions of processor and transcriber objects. This was done to test various approaches like streaming audio into the DS transcriber to get a real-time transcription or importing recordings for asynchronous transcription. The implementation is a proof of concept, as later applications might use this code as a basis for a transcriber outside of discrete commands. The model will most likely get fed with a real-time audio feed and react to it in a production input.

2.5 Data Collection

In order to collect the data, a web app was build to survey the required voice samples. For a multitude of reasons, including lack of funding and customization options, commercial options like Amazons *Mechanical Turk*[7] were ruled out.

[6]DS Python Package: `https://pypi.org/project/DS/`
[7]Amazon Mechanical Turk: `https://www.mturk.com`

2.6 Survey Concept

During this survey, participants were guided through a set of situations where they record the test cases. The underlying backend [10] then saves and labels the data. The following criteria where important to meet:

1. Use a sentence list to record voice samples
2. Full control over the websites content to comply with legal requirement
3. Provide a form to raise contextual data as discussed in 2.1

Based on the criteria mentioned earlier, a concept was developed that defines the user story for such a website as shown in Fig. 3.

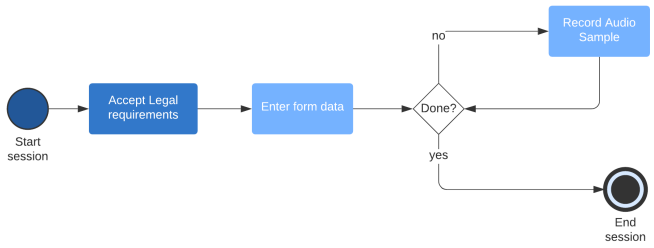

Figure 3: User story of the inquiry from a participants perspective

The participant was asked to enter their age range, first language, and gender. These data points were inquired to add context to the voice samples to measure their influence on the transcription success (e.g., overrepresentation of a particular group and model bias). The order of the sentence list is randomized for each participant, as it is anticipated that some participants will end ongoing sessions before recording all the voice samples. Missing voice recordings will therefore be distributed evenly in the data set.

Table 1 shows some of the 54 test cases. The *STRING* column is the grammatically correct text that the inquiry participants will read. The CLASS column is a grouping of the specific test cases. It is, therefore, also the expected transcription for a resulting sound file that participants record. It was essential to find test cases that test both norm and edge cases. These test groupings are:

Two discrete commands: The reasoning behind this test case is that when two commands are correctly transcribed, it would be interesting to see which command gets precedent during command recognition. An example of this would be *right left*.

Anagram: Cases that consist of words with the same letters as the command they test. Since DS works based on letter recognition, it was interesting to see how well these tests would be transcribed and how consistent these transcriptions would be. An example of this is *girth* and the command *right*.

Coincidental inclusions: Words that coincidentally include a command, like *clefting*, which includes the command *left* in order to test how the transcriber handles words with a similar acoustic profile.

Table 1: A sample of the inquired text snippets (STRING) and their classification (CLASS)

STRING	CLASS
left	generic
drive left	combination
leftmost	compound
turn left	combination
clefting	coincidental inclusions
right-hander	compound
girth	anagram
fright	rhyme
right left	two discrete commands
right away	not a move order
backwardness	not a move order
bad word	sounds similar
manufactured	rhyme

Combination: Word combinations are a group of tests that consist of a word that is not a command and one that is a command while being syntactically and grammatically correct. An example of this is *drive left*, consisting of the command *left* and the verb *driving*. The premise was to test how the transcriber would get affected by two words after another.

Compound: Compound words are words that are compounded of a command and an additional letters while being a grammatically correct word. This group is similar to the class 2 tests, with coincidental inclusions. However, the command word is not embedded into other letters, but the commands lead or tails the test word. An example of this is *leftmost*, leading with the command *left*.

Generic: The generic test group is mostly a control group to test the commands in their generic form. Instead of testing an edge case, this group aims to provide data on how well the transcriber generally performs when transcribing the regular commands without trying to trap them. It consists of the cases *left*, *right*, *forward* and *backward*.

Negation: The negation test is similar to the class 3 tests, like *drive left*, but instead of verbs or other words, it uses negation words in combination with commands. The reasoning was the same as with combination test cases. Since negations of commands are much more common in natural language than combinations of commands with other words, this type of combination was isolated in a test class of its own. An example of such a test would be *don't go left* or *don't go forward*.

Not a move order: Control group, but instead of consisting of only a command, these tests consist of no command at all. These tests were conducted to see how the absence of a command-type word would affect the transcription performance.

Rhyme: Words that rhyme with a command. It was suspected that rhymes would get misidentified since their acoustic profile is similar. Especially with regards to the ending of a word. An example of this is *write*, being a rhyme of *right*.

Sounds similar: Cases that are not rhymes, but also similar acoustic profile to a command. Much like the rhymes, these words were included to test the precision of the transcriber with words that have a similar acoustic profile like *bad word* sounding similar like *backward*. It was suspected

that the model would not be able to transcribe these cases accurately.

Besides similar-sounding test cases, generic test cases were also included in a control setting where the command would be as literal as possible.

2.7 Related Works

In a similar command recognition project by Oualil et al. [11], a command recognition pipeline is used to transcribe air traffic control intercom traffic. Their approach differs from the described one regarding processing the transcribed controller commands with the help of predictive analysis to improve the recognition rate. It was important to aid the transcription processing by predicting a likelihood for possible commands. Their work showed that this approach improves the accuracy of their results and reduces errors by 30% to 50%.

Lv, et al. 2008 [12] uses a statistical approach to interpreting audio commands for robots while also utilizing a similar basic command set for testing purposes. The testing, however, is designed very limited in volume. Each of their seven commands was only tested ten times, however successful. Their system also seems not to be tested in cases where natural language could be misinterpreted as a command.

3 Data Analysis and Results

This section describes the data structure and the analysis of the results. First, the results of the survey will be outlined, then analyzed in detail.

3.1 Survey Results

The data obtained from the survey allowed testing, as well as providing metadata. The collected data consists of 1215 soundfiles, spread across 49 participants. Because the DS model is optimized for interpreting sound files with a sample rate of 16 kHz, a down-sampling from 44.1 kHz was required. Corrupted sound files were ignored in the analysis.

3.2 Applying the surveys data to command recognition

Completing the survey provided enough data to analyze how well DS was performing. It is expected that the strings got recognized with a varying degree of accuracy. Overall, the exact strings were only met with an accuracy of 31.1 %.

Table 2: Transcription results by gender

group	incorrect	correct	confidence
female	190	79	-16.387
indeterminate	44	64	-18.765
male	516	196	-15.765

The words that got transcribed ended up having a high similarity with the expected value. They primarily consist of phonetically similar letters or parts in the expected string. Instance of this are *byte*, which was transcribed to *by*, *bite*, *by it*. In severe cases, the model has transcribed potential move commands, like *write* and *wright* to *right*. In other instances, the model has difficulty

working with hyphens: *right-hander* was transcribed to *right hander*, using a white-space instead of a hyphen. Other cases showed misplaced white-spaces, such as *drawback* becoming *draw back*. These cases were labeled as false transcriptions due to hard comparison rules. In some instances, the result were similar words, like *scan*, becoming *skin* and *girth* becoming *girls*. As mentioned in subsubsection 1.4.1, DS also gives access to transcription confidence metrics. Overall confidence was almost equally high with almost -17 for the correct and about -15 for the incorrect transcriptions. The confidence value is defined as: *Approximated confidence value for this transcript, is roughly the sum of the acoustic model logit values for each timestep/character, contributing to the creation of this transcript.* – [13]

Table 3: Transcription results by age group

age group	incorrect	correct	confidence
18-21	111	71	-16.508
22-25	135	83	-16.789
26-29	40	13	-16.867
30-33	194	67	-16.100
34-37	90	72	-17.279
38-41	86	20	-15.165
42-45	46	8	-13.937
58-61	48	5	-13.961

A higher value means a higher confidence [14]. It can be concluded that the transcription confidence was slightly better for the incorrect results. Table 3 and Table 2 show the context, under which this confidence was generated. Both tables show the grouping of the transcription results by age group and gender, respectively, and reveal that results were the same for all age and gender groups for most instances. The best results were observed in in the age group *18 to 25*, with a significant drop in transcription success the older the participants are.

The following graphics present the transcription results. They show how often the transcriber was able to transcribe the recorded audio as expected. The right diagram of each figure shows the seven most common transcriptions, ordered by the number of occurrences. Each bar in the graph is labeled with both the ID of the test and the transcription result. The transcription result is preceded by an asterisk (*). The complete list is not included since 25 results are the maximum that is is reasonably well to read and can be generated from the data in the analysis repository [8].

Two discrete commands is shown in Fig. 4. It consists of two tests, *right_left* and *left_right*. Both tests were transcribed almost equally and with higher accuracy of 45 % than the overall average of 30.1 %. In this group, one can see a distinct group of 18 successful transcriptions, with the other transcriptions not occurring more than once in the group of the seven most common transcriptions.

Anagram (Fig. 5) however stands in contrast to the CLASS 0. It also consists of very few tests with a total of three, being *felt* (anagram of *left*), *drawback* (anagram of *backward*) and *grith* (anagram of *right*). Out of these three tests, only *felt* was transcribed correctly more than once. The most common transcription result has been *felt*, with a total of five instances. The other six most common transcriptions were all mistranscriptions. The 3 instances of *drawback* being transcribed to *draw back* being the most common. Other transcriptions have a maximum of two or fewer instances.

Coincidental inclusions (Fig. 6). This group consisted of two test cases with 40 instances to

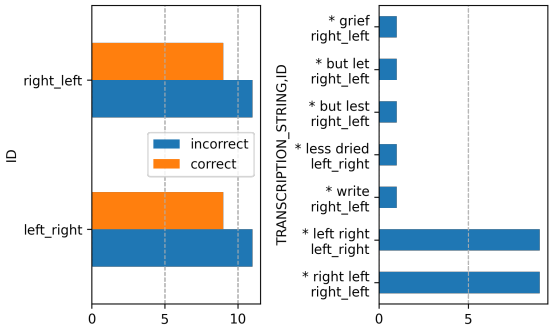

Figure 4: Transcription results of CLASS: *two discrete commands*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

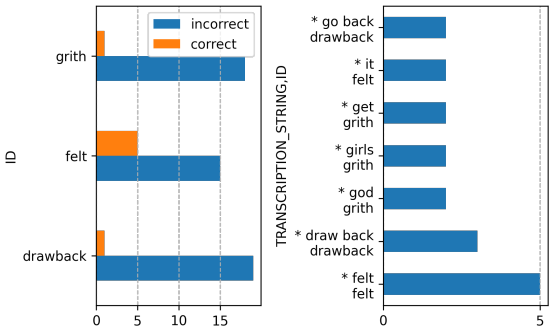

Figure 5: Transcription results of CLASS: *anagram*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

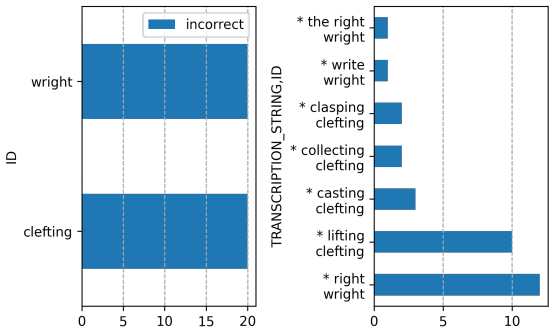

Figure 6: Transcription results of CLASS: *coincidental inclusions*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

test how well the transcriber would handle words that include commands. What stands out here is that they have zero instances of successful transcriptions.

In the case of *wright*, the transcriber did precisely what was suspected and transcribed the word to *right*. *Clefting*, did not transcribe to *left*, but to *lifting* in almost every instance. These two cases make up almost half of the transcribed cases, with 14 of the 40 instances. The other transcriptions were less frequent, with just 3 of them over one instance.

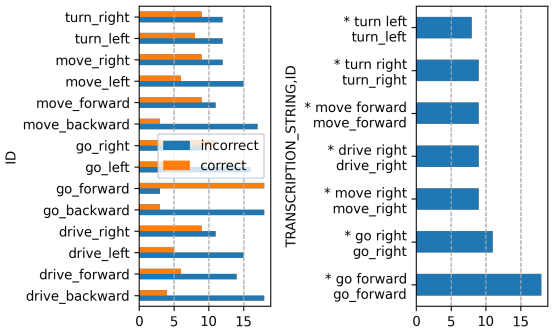

Figure 7: Transcription results of CLASS: *combination*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

Combination (Fig. 7) is a very big group with 14 tests. This group consists of tests that are designed to combine a command with a syntactically meaningful word, like *turn right*. These test classes of commands are thought to be relevant to real-life scenarios and therefore were thoroughly tested. While the results for *turn_right* and *turn_left* are almost identically in transcription performance (9 and 8 correct out of 12), results of *go_backward* and *go_forward* are the exact opposite of each other with 3 out of 14 either correctly transcribed or mistranscribed. The 14 correct transcriptions of *go_forward* are also the best-transcribed case overall in this group. Much like *turn_right* and *turn_left*, *drive_backward* and *drive_forward* have similar results. Another noteworthy pair are *go_left* and *go_right*. While the latter had as many correct as incorrect transcriptions, *go_left* performed worse, with only 4 out of 16 correct transcriptions.

Compound (Fig. 8) consists of four tests. The compound group consists of words that form a new meaning in combination with another word, either separated by a white-space or written together with a command, like *rightmost* or *left-hander*. This group did not yield any correct transcriptions. The right part of the figure shows the seven most prominent results and reveals that the reason for the failed transcription is that the hyphen (-) in *left-hander* is missing, but the word is otherwise correctly transcribed, but still technically a mistranscription. These cases make up 7 of 42 results. Another big part of the results is *leftmost* and *rightmost*. These words are written together but were transcribed with a white-space in between, changing the meaning completely. These make up another 22 cases, resulting in 29 out of 42 near misses.

Generic (Fig. 9) consists of four tests, which are the positive control group, mentioned in subsection 2.6. The group performed 51.2 % correct transcriptions, which is better than the overall average of all tests of 30.1 %. The right side of the graphic also shows the top 4 most frequent transcriptions of this group. The best transcribed test was *right*, with 14 out of 20 correct transcriptions, followed by *backward* and *left* with 10 each, and lastly *forward* with 8.

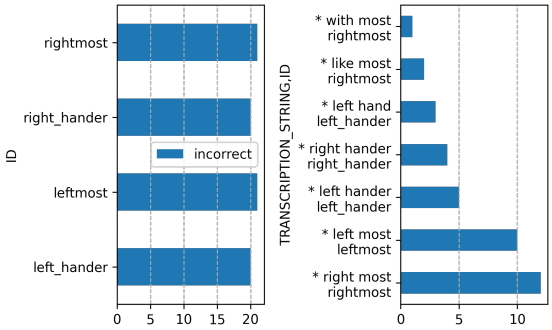

Figure 8: Transcription results of CLASS: *compound*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

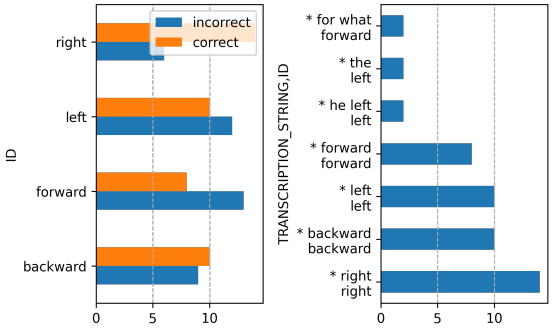

Figure 9: Transcription results of CLASS: *generic*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

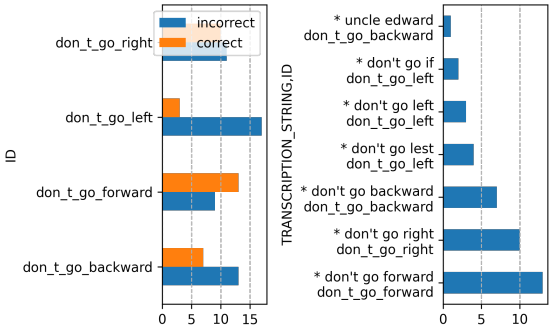

Figure 10: Transcription results of CLASS: *negation*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

Negation (Fig. 10) consists of four tests. The results from this group were just barely above average with 39.7 % success rate. The best performing was *don't go forward* with 13 out of 22 successful transcriptions, followed by *don't go right*. Noticeably worse than the other two tests performed *don't go left*, with just 3 out of 20 successful transcriptions. The majority of tests in this group were not transcribed correctly.

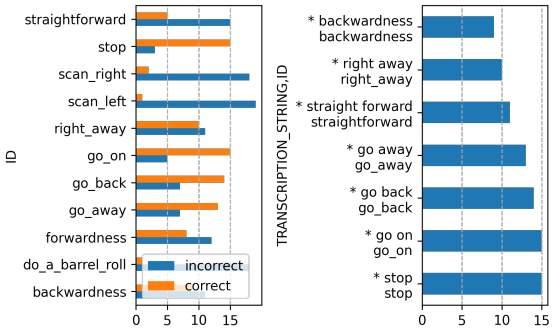

Figure 11: Transcription results of CLASS: *not a move order*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

Not a move order (Fig. 11) is also a control group, described in subsection 2.6. It serves the purpose of highlighting transcription precision for non-command related input. This group is also fairly large with 11 tests with the best performing transcriptions being *stop*, *go_on* and *go_back*. The groups transcription performance was 42, 5 % and therefore better than the overall average. Noticeably under-performing have been *scan_left* and *scan_right*. The most common transcriptions were the correct transcriptions, being *stop* (15), *go on* (15), *go back* (14) and *go away* (13). The only false transcription was a near miss, with *straightforward* being transcribed to *straight forward* (11).

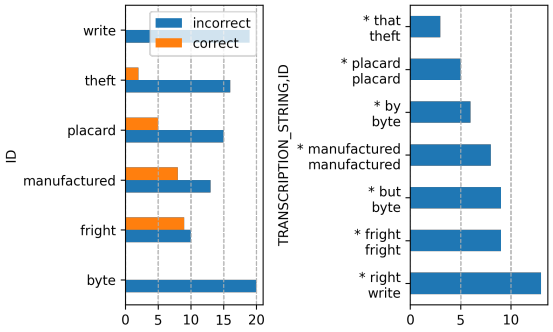

Figure 12: Transcription results of CLASS: *rhyme*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

Rhyme (Fig. 12) is a group with 6 tests. The idea is that words that rhyme with a command would be misinterpreted as a command because the words used as commands are more common

and have a similar acoustic profile like e.g. in *byte* or *write* and *right*. Best transcribed tests in this group were *fright* (rhyme of *right*) 9 out of 19 correct transcriptions, with *manufactured* (rhyme of *backward*) with 8 out of 21 correct transcriptions. The worst performing test in this group was *write* and *byte* (rhymes of *right*), having both zero correct transcription.

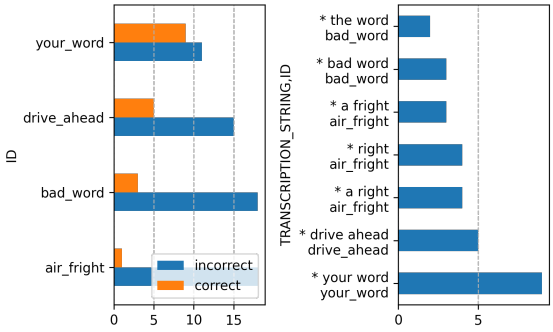

Figure 13: Transcription results of CLASS : *sounds similar*. The left plot shows the total amount tests conducted and their results. The right plot shows the seven most common recognitions among all results.

Sounds similar (Fig. 13) is related in intent to the rhymes and consists of 4 tests. The Best performing test was *your_word* (similar to *forward*) with 9 out of 20 correct transcriptions. The other three tests (*drive_ahead*, *bad_word*, *air_fright*), were consistently failed to get accurately transcribed.

4 Conclusion

Before going into the conclusion, it should be noted that the tests conducted were designed to test the model to failure in order to uncover weaknesses. It is unlikely that the model would have to differentiate between these test cases in a production environment. Knowing its limitations is essential to avoid these scenarios or comparing other models to it.

4.1 Consolidating the surveys results

Analysis of the transcription performance delivered mixed results in terms of reliability, as shown in subsection 3.2. The transcriber was able to transcribe the groups *two discrete commands*, *generic*, as well as some parts of *not a move order* with above-average success. Transcription accuracy for simple and commonly used words, such as *stop*, *go on*, *left*, *right* was high. The tests where the model struggled the most were long words like *drive backward* or compound words like *rightmost* or *right-hander*. In stark contrast to these tests were the results for the test groups *compound* and *coincidental inclusions* with zero correct transcriptions. Some minor issues like punctuation errors lead to an overall low performance. Altogether, 31 of a total of 42 of the tests could have been successful (73.8 %). This leaves much room for improvement, which can be achieved by the conceptualized deployment of a natural language processing stage in the pipeline. Tests of the group *rhymes* and *coincidental inclusions* showed that the model is not able to differentiate between words that have similar pronunciation or acoustic profile. It must be concluded, that pre-trained

v0.6.0 DeepSpeech model is not able to handle audio transcription requiring high accuracy. Post-processing could improve the transcription results and mitigate instances of mistranscriptions and help match the transcriber's output to the command set with knowledge graphs and word families. False positives, like *write* or *wright* becoming *right* will be very hard to avoid. One solution could be the use of a command indicator word. The recognized word would then signal the system that subsequent data is most likely a command. An additional way to supplement the performance, one could pre-train the model with domain-specific language to improve the results.

4.2 Critical review

The proposed implementation of the voice command pipeline from subsection 2.2 is a similar approach to Oualil et al. [11] and has the potential to serve as a baseline for future research. It was shown that there was no notable bias regarding the age and gender of the speakers. This is an important result, since speaker independence helps lowering required adjustments for new users to a minimum and improving the user experience. Regarding architectures, comparing results between DeepSpeech's RNN approach and a long short term memory (LSTM) or even a temporal convectional network (TCN) could be worthwhile [15]. Because DS works on a letter by letter basis to transcribe individual words, it is unlikely that context would improve results.

For future work, the command recognition pipelines natural language processing stage is a first candidate for implementation and expansion of the project. It would help to validate the suggestions made in 4.1 to improve upon the shortcomings of some aspects of the pipeline where words have been correctly transcribed but misspelled. It could also help the identity context; the command is negated, or transcribed as more than one word like in the group *generic* (*left* transcribed to *he left*). Other NLP features could also introduce a list of commands with synonymous meaning and normalization of commands to improve matching transcriptions to the discrete command set. One way to potentially improve accuracy would be to introduce command prediction. If a command like *go forward* were given and executed, it would make no sense to issue the same command twice. It is, therefore, unlikely that a given command is supposed to be executed. On behalf of further testing, an unexplored possibility would have been to augment data to include Gaussian noise to simulate wind or static crackling to simulate interference. This would expand the test data and help understand how the model performs under non-ideal conditions.

Since the focus of this work was on DeepSpeech, future research should expand the scope of the tests developed in this paper by taking additional STT models and engines into consideration. This would enable comparing results across multiple candidates for the application.

References

[1] M. Smith, D. Craig, N. Herrmann, E. Mahoney, J. Krezel, N. McIntyre, and K. Goodliff, "The artemis program: An overview of nasa's activities to return humans to the moon," in *2020 IEEE Aerospace Conference*. IEEE, 2020, pp. 1–10.

[2] E. Allak, C. Brommer, D. Dallenbach, and S. Weiss, "Amadee-18: Vision-based unmanned aerial vehicle navigation for analog mars mission (avi-nav)," *Astrobiology*, vol. 20, no. 11, pp. 1321–1337, 2020.

[3] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[4] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common voice: A massively-multilingual speech corpus," *arXiv preprint arXiv:1912.06670*, 2019.

[5] R. Morais, "Deepspeech 0.6: Mozilla's speech-to-text engine gets fast, lean, and ubiquitous," 2019. [Online]. Available: https://hacks.mozilla.org/2019/12/deepspeech-0-6-mozillas-speech-to-text-engine/

[6] C. C. Perez, *Invisible women: Data Bias in a World Designed for Men*. Abrams, 2019.

[7] N. P. Trilok, S.-H. Cha, and C. C. Tappert, "Establishing the uniqueness of the human voice for security applications," *Proc. CSIS Research Day, Pace University, NY, May*, 2004.

[8] Tobias Kolb. (2020, Nov.) BlkPingu/VoiceControl: Thesis Release of VoiceControl. [Online]. Available: https://doi.org/10.5281/zenodo.4323529

[9] R. E. Johnson and B. Foote, "Designing reusable classes," *Journal of object-oriented programming*, vol. 1, no. 2, pp. 22–35, 1988.

[10] Tobias Kolb. (2020, Nov.) BlkPingu/SurveyBackend: Thesis Release of the SurveyBackend. [Online]. Available: https://doi.org/10.5281/zenodo.4295636

[11] Y. Oualil, D. Klakow, G. Szaszák, A. Srinivasamurthy, H. Helmke, and P. Motlicek, "A context-aware speech recognition and understanding system for air traffic control domain," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2017, pp. 404–408.

[12] X. Lv, M. Zhang, and H. Li, "Robot control based on voice command," in *2008 IEEE International Conference on Automation and Logistics*. IEEE, 2008, pp. 2490–2494.

[13] R. Morais. Deepspeech. [Online]. Available: https://github.com/mozilla/DeepSpeech/blob/430132c5a50f79c8cb28851e94963f8e10926d17/native_client/deepspeech.h#L45

[14] K. Davis. Deepspeech issue #3004. [Online]. Available: https://github.com/mozilla/DeepSpeech/issues/3004#issuecomment-631289058

[15] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.