Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Robust Bidirectional Processing for Speech-controlled Robotic Scenarios

Dissertation

submitted to the Universität Hamburg, Faculty of Mathematics, Informatics and Natural Sciences, Department of Informatics, in partial fulfilment of the requirements for the degree of Doctor rerum naturalium (Dr. rer. nat.)

submitted by

Johannes Twiefel

Hamburg, 2020

Submission of the thesis:

17.02.2020

Day of oral defense:

20.05.2020

Dissertation Committee:

Prof. Dr. Stefan Wermter (advisor)

Dept. of Computer Science

Universität Hamburg, Germany

Prof. Dr.-Ing. Wolfgang Menzel (advisor)

Dept. of Computer Science

Universität Hamburg, Germany

Prof. Dr. Victor Uc Cetina (chair)

Dept. of Computer Science

Universität Hamburg, Germany

Für

Hans-Heinrich und Anke,
Martin und Aude,
Oma Lore,
Opa Hanns und Oma Emmi,
Peter und Silke
und meine Freunde.

Abstract

Automatic Speech Recognition (ASR) is often employed for applications like dictation, where the aim is to cover a broad range of vocabularies. Also, ASR is a central interface for humans to communicate or control a system. Those systems can perform a fixed set of actions and follow a well-defined goal. Audio is recorded using a microphone, the ASR system produces text hypotheses, and a natural language processing (NLP) system derives machine-readable representations from text. These representations are afterwards employed to instruct the system to perform a defined action to achieve a goal. At a first glance, this approach of orchestrating a unidirectional processing pipeline appears to be reasonable and is often followed in practice. In this thesis, we demonstrate, that there are better approaches to address this kind of tasks and present a more suitable one.

A well-known issue of ASR systems is that a growing vocabulary of words that could be recognized by the system leads to a higher word error rate (WER). For applications like dictation, this issue is hard to address, but for the before-mentioned problem of controlling a system, we are able to address it. Usually, the number of goals and possible actions of the system is limited; the possible text instructions are also limited. This leads to a smaller vocabulary, which improves the performance of the ASR system. Another limitation of the unidirectional processing chain approach is the assumption of NLP systems to receive correct text input. Although these systems are trained on (clean) text, it is still a challenge to recognize a correct natural language representation from it. As the processed text is produced by an ASR system, it is possibly incorrect, making it hard for the NLP system to recognize the correct meaning from incorrect text. If afterwards a spoken command cannot be executed by the system, it is rejected, and the user needs to repeat the instruction.

In this thesis, we present a self-trained ASR system that performs better than Google's cloud-based ASR on a benchmark data set. We also define a

novel and simple natural language representation called *Semantic Logic Predicates (SemaPreds)*. In our experiments, we show that we can successfully recognize *SemaPreds* from speech input. The approaches we developed make it possible to interpret *SemaPreds*, find and correct errors inside them, and evaluate their plausibility regarding a given situation. We test our novel bidirectional processing chain in a human-robot interaction scenario and show that it works robustly and performs better than a unidirectional processing pipeline. These results indicate that the novel representation and the bidirectional processing chain can be useful for other speech-controlled system scenarios.

Zusammenfassung

Automatische Spracherkennung wird häufig für Diktieranwendungen verwendet, welche ein großes Vokabular aufweisen. Außerdem ist die automatische Spracherkennung eine Hauptschnittstelle, um mit einem System zu kommunizieren oder es zu kontrollieren. Diese Systeme können einen festen Satz von Aktionen ausführen und folgen einem wohldefinierten Ziel. Audiodaten werden von einem Mikrofon aufgenommen, die Spracherkennung erzeugt Texthypothesen und ein System zur natürlichen Sprachverarbeitung erkennt maschinenlesbare Repräsentationen des Textes. Diese Repräsentationen werden danach vom System genutzt, um eine definierte Aktion auszuführen und ein Ziel zu erreichen. Auf den ersten Blick macht es Sinn, eine eindirektionale Verarbeitungspipeline aufzubauen, dieser Ansatz wird häufig in der Praxis verfolgt. In dieser Arbeit zeigen wir, dass es bessere Ansätze für diese Art von Aufgaben gibt und präsentieren einen besser passenden Ansatz.

Ein wohlbekanntes Problem mit Spracherkennungssystemen ist, dass ein größeres Vokabular zu einer höheren Wortfehlerrrate führt. Für Diktieranwendungen ist dieses Problem schwer zu behandeln. Für die zuvor genannte Anwendung des Kontrollierens eines Systems sind wir in der Lage, dieses Problem zu behandeln. Normalerweise ist die Zahl der Ziele und möglichen Aktionen für diese Systeme limitiert. Dadurch sind auch die möglichen Texteingaben begrenzt. Dies führt zu einem kleineren Vokabular, was die Performanz eines Spracherkennungssystems verbessert. Eine andere Limitierung der eindirektionalen Vererbeitungspipeline ist die Annahme, dass korrekte Texteingaben vorhanden sind. Obwohl diese Systeme auf sauberen Texteingaben trainiert wurden, ist es immer noch eine Herausforderung korrekte Repräsentationen wiederzuerkennen. Da aber der Eingabetext aus Spracherkennungssystemen stammt, welcher möglicherweise inkorrekt ist, wird die Erkennung durch ein natürlichsprachliches Verarbeitungssystem zusätzliche erschwert. Wenn ein Befehl nicht ausgeführt werden kann, wird dieser zurückgewiesen und der Benutzer muss seine Instruktion wiederholen.

In dieser Arbeit präsentieren wir eine selbsttrainiertes Spracherkennungssystem, welches auf einem Benchmark-Datensatz besser funktioniert als Googles Spracherkennungssystem. Außerdem definieren wir eine neue und einfache Repräsentation für natürliche Sprache, genannt *Semantic Logic Predicates (SemaPreds)*. In unseren Experimenten zeigen wir, wie *SemaPreds* aus natürlicher Sprache erkannt werden können. Die entwickelten Ansätze können *SemaPreds* verarbeiten, Fehler darin finden und diese korrigieren. Außerdem können sie die Plausibilität der *SemaPreds* in einer gegebenen Situation evaluieren. Wir testen unsere neue bidirektionale Verarbeitungskette in einem Mensch-Computer-Interaktionsszenario und zeigen, dass sie robuster und besser als eine eindirektionale Verarbeitungspipeline funktioniert. Unsere Resultate zeigen, dass die neue Repräsentation und die bidrektionale Verarbeitungskette nützlich für sprachgesteuerte Systemszenarien sind.

Contents

Chapter 1

Introduction

1.1 Motivation

The processing in speech-controlled agents like robots is performed in a unidirectional processing pipeline, leading from automatic speech recognition (ASR) to command execution. This pipeline involves ASR, natural language processing (NLP), natural language understanding (NLU), execution parameter extraction and actuators like movable joints. A common and naive[1] approach to develop such a pipeline is the following: First, an ASR system is chosen. Human-robot interaction (HRI) systems often choose Google's cloud-based ASR system, because it can be used as a blackbox model which does not require training and works out of the box (see Section 3.6). A disadvantage is a high response time, as it requires an internet connection and the speech processing is performed in the cloud. Also, these systems often do not allow domain adaptation leading to the occurrence of out-of-domain words in the ASR outputs. The next step of creating the pipeline is to choose an appro-

[1] In this case, a naive approach is an intuitive approach to tackle a problem, without considering or possessing deeper knowledge about the problem, leading to a low performance compared to other more informed approaches.

priate NLP strategy. Many systems employ keyword spotting leading to a limited flexibility and performance. Another strategy is using a more sophisticated natural language representation like dependency trees or similar tree-like structures. Nowadays, tree parsers achieve a high performance and provide a more useful natural language representation compared to simple keywords. Usually, these structures are trained on text input. Problems occur, when the input to the parser is noisy or incorrect. This is the case, when the text input is coming from an ASR system. As long as the text produced by the ASR system contains errors, it is not possible to generate a correct tree and finally execute the command. There are approaches to tackle this issue (see Section 3.7), but we consider them to be not suitable for real applications and argue they only work in theory.

We developed different requirements how to tackle the mentioned issues in the unidirectional processing pipeline. The first requirement is to provide accessibility to the language models of the ASR system. This way, the language model can be adapted to the domain, resulting in a smaller vocabulary. It was already shown that a smaller vocabulary may result in a lower word error rate (WER) (Twiefel et al., 2014). To reduce the response time of the ASR system, we propose to train an acoustic model and replace the cloud-based ASR system with a local one. Another requirement is a natural language representation that can be recognized from noisy or incorrect input sentences. We consider the repair mechanisms for dependency trees to be too limited and state that we require a novel natural language representation. This novel representation requires to be (partially) recognizable even when the input is incorrect. Also, it requires to be correctable if parts of it are incorrect. The representation needs to be interpretable by an interpreter and executable given a concrete scenario, for example an HRI scenario.

In this work, we extend the known unidirectional processing pipeline to a bidirectional processing chain (BPC). We present our own ASR system called *SlimSpeech*, which we trained on free and public audio and text data.

Its language models can be adapted to a domain. Additionally, we introduce a novel postprocessing system that works within a domain and repairs ASR outputs. The structures employed by the postprocessing system are called *Sentence Templates* (STs). We omitted the idea of using dependency trees or performing keyword spotting and define a novel and more suitable natural language representation for this kind of problem called *Semantic Logic Predicates* (*SemaPreds*). We show how to successfully recognize *SemaPreds* from ASR outputs. For this purpose, we developed the *SemaPred Recognizer* (SPR) that is based on convolutional neural networks (CNNs). We show that *SemaPreds* can be encoded using the logic programming language *Answer Set Programming* (ASP), which is performed by the *SemaPred Interpreter* (SPI). Additionally, we integrated the *Semantic Evaluator* (SE) that is able to measure the quality of the recognized *SemaPreds* and identify potentially incorrect parts using a semantic evaluation process that compares the text input to the recognized *SemaPreds*. This way, the unidirectional processing pipeline is transformed to a bidirectional processing chain. We add another bidirectional component to the processing chain called the *Crossmodal Corrector* (CC). It introduces *wildcard* slots to the *SemaPreds*. These slots can be filled with more plausible assignments and are determined by the logic system inside the SPI. The CC employs the SE to identify potentially incorrect parts of the recognized *SemaPreds* and replaces them with *wildcards*. Afterwards, it tries to put plausible assignments to the *wildcard* slots. This way, the partially incorrect *SemaPreds* may be repaired or the user can be informed about implausibilities.

We show that our BPC approach is suitable for domain-dependent scenarios. For this purpose, we chose the *Blocks World* scenario that contains a discrete grid world and a robot arm. The robot is instructed to move differently colored objects around (cubes and pyramids). The *Train Robots* data set comprises training and test data from this domain, including text instructions, tree-like annotations and scene descriptions. We extended the data set by recording audio test data and annotated *SemaPreds* for the training and test set. To test the behaviour of the system under noisy conditions, we also

created a noisy variant of the data set by adding artificial noise to the audio data. The extended *Train Robots* data set is called *Knowledge Technology Train Robots* (KTTR) and is used to show that our BPC approach performs better than the unidirectional processing pipeline. For this purpose, we created such a unidirectional processing pipeline using Google's ASR system together with our SPR and SPI. We also present a simulator that is able to display scenes from the data set and that can be controlled using speech instructions. The system was also implemented on a real robot; a user can instruct it to move real cubes.

1.2 Research Questions

In our work, we focus on domains using ASR as a control, for example, HRI scenarios. In these scenarios, a fast responding system is required. It is also required to have control over the ASR system, which is not given for cloud-based systems. This leads to the first research question:

Question 1: How is it possible to develop a local ASR system that is usable in realtime and achieves similar performance as state-of-the-art speech recognition models?

If the first question can be answered by presenting such a system, we want to overcome the unidirectional processing pipeline approach of using a black-box cloud-based ASR system together with an NLP system that can be adapted to the domain. We do this by changing the strategy of using a black-box system by adapting the now controllable and modifiable ASR system to the domain, leading to the question:

Question 2: Are there better alternatives for speech applications than taking a black-box ASR together with a domain-dependent NLP system?

In our motivation, we stated that a well working ASR system with a low WER may not provide a correct representation of an utterance. This error is expected to be propagated through the NLP system leading to a partially incorrect meaning representation which perhaps cannot be executed. Here, we see a need for a novel meaning representation that might contain incorrect parts that can be corrected and though commands can be executed, resulting in the question:

Question 3: Is it possible to define a novel NLP representation that can be corrected if parts of it are incorrect? Is the novel representation suitable to be interpreted to compute concrete execution parameters?

It is expected that the quality of sentences coming from ASR systems is decreasing with an increase of noise inside the audio signal. It is especially interesting to analyse the behaviour of our system under very high noise levels, which may occur during extreme situations in real-world applications. We ask the question:

Question 4: Is the developed system also working under very noisy conditions?

1.3 Novelty and Contribution to Knowledge

In this work, we propose different novelties that we consider useful for other researchers and real-world application engineers.

- We propose a novel processing strategy, the bidirectional processing chain. It contains an ASR system, a novel natural language representation called *SemaPreds* and is able to robustly process speech utterances in (HRI) domains by introducing a bidirectional repair system for speech

utterances.

- We present an ASR system that can be trained using local computing power and freely available training data. The system is a smaller variant of *DeepSpeech 2*. The components of the architecture are already contained or similar to the original system. The lower number of the layers leads to faster training and execution time.

- Two novel language models (*Sentence Template Grammars* and *Sentence Template N-Grams*) are presented. The first one is able to learn a grammar-based language model in an unsupervised way by presenting training sentences. The second one extends the model into a hybrid system of a grammar and a statistical N-gram language model and is especially useful as a post-processing system for cloud-based ASR.

- We define the novel natural language representation *SemaPreds*, which possesses a low complexity and which is especially suitable for real-world applications. *SemaPreds* can be corrected using the context, to be able to handle possibly incorrect text coming from an ASR system.

- We present a novel CNN-based NLP model that is able to recognize *SemaPreds* from a natural language sentence and that does not require recurrent neural networks to handle sentences making it fast in execution and training. This way, the model can be trained in domains with a low size of training data.

- Another novelty is the *SemaPred Interpreter* that is able to generate execution parameters from *SemaPreds*, making it useful for, for example, HRI scenarios. We developed the concept of the interpreter, while the implementation was performed by Tobergte (2017) during his Bachelor's thesis (under our supervision).

- We developed a novel *Semantic Evaluation* mechanism that is able to check the quality of *SemaPreds* by providing a confidence value to each of

their parts. It is based on *fastText* representations and the Needleman-Wunsch-Algorithm. This way, potentially incorrect parts within the *SemaPreds* can be identified.

- The *SemaPred Interpreter* is extended to support *wildcard* slots within *SemaPreds*. These slots are identified as potentially incorrect by the *Semantic Evaluator* and refilled to find an assignment that is consistent with the context. In our example scenario, an abstraction of a visual scene is used, making it a *Crossmodal Corrector* that corrects hypotheses coming from the speech modality using knowledge from the visual abstraction. This system increases the robustness of the whole speech-to-execution chain.

- For the example scenario, we present a novel simulator that is able actually to perform commands coming from a microphone. It is web–based and can also be used to simulate other scenarios containing a robot arm.

- We also present a novel concrete HRI application using the NICO robot. Our system was used to control the robot in a real-world scenario. This system is a basis for further applications and research.

1.4 Structure of this Thesis

In this chapter, we presented the motivation of our work, derived research questions, and listed the novelties and contributions to knowledge of our work. Chapter 2 contains the methods we use for our developed approaches. We shortly describe the neural networks, algorithms, and word embeddings we used for our models. We give an introduction to Answer Set Programming, a declarative programming language used by our *SemaPred Interpreter*. Chapter 3 contains related work. The Blocks World scenario is introduced, which serves as a basis for our example scenario. Also, we describe the *Train Robots* data set that contains data of the Blocks World scenario. Additionally, we describe

an NLP system that was used as an inspiration and we describe the current state-of-the-art in ASR. We present our previous work that is adapted in our novel language models. Afterwards, we give an overview about related works regarding speech-controlled HRI scenarios. Finally, we give an overview about strategies for dependency parsing to handle ungrammatical text input, which is a similar task to *SemaPred* recognition from speech input. Chapter 4 contains all approaches we developed including the definition of the novel *SemaPred* representation, the *SemaPred Recognizer*, our ASR system *SlimSpeech*, the *Sentence Template* language models, the *SemaPred Interpreter*, the *Semantic Evaluator*, the *Crossmodal Corrector*, the simulator and the real-world application using the NICO robot. Chapter 5 describes the datasets we used in our experiments. It contains the TIMIT Core Test Set, an ASR benchmark test set used to measure the performance of ASR systems. It also contains the *Knowledge Technology Train Robots* (KTTR) dataset that comprises clean and noisy audio data, text and *SemaPreds* to measure the performance of the whole system. Then, we give a short summary of the approaches evaluated in our experiments. Chapter 5 also contains the experiments performed and their results. Chapter 6 contains discussions and the conclusions taken from the experiments and presents answers to the research questions asked in this chapter. It also contains a summary of possible future work.

Chapter 2

Methodology

2.1 Introduction

In this chapter, we give an overview of the methods used for our presented approaches. First, we give a brief description of Convolutional Neural Networks that we use for one of our natural language processing modules and our acoustic model. Afterwards, Gated Recurrent Units are introduced, which we employ for our acoustic model. Then, we described *Connectionist Temporal Classification*, a loss function utilized for sequence-to-sequence training of neural networks, which is performed when training our acoustic model. Thereafter, we describe *Beam Search Decoding*, which is used to decode the output of our neural acoustic model, providing language model support. Next, we introduce word embeddings, which are utilized as input representation for one of our natural language processing modules and also for our *Crossmodal Corrector*. After this, we give an overview of Answer Set Programming, a logic programming language similar to *Prolog*. Our *SemaPred Interpreter* is based on Answer Set Programming. Finally, we describe the Needleman-Wunsch Algorithm, an alignment algorithm used by our *Crossmodal Corrector*.

2.2 Convolutional Neural Networks

For the proposed NLP and ASR system, we employ Convolutional Neural Networks (CNNs) (LeCun et al., 1998) in different variants. CNNs contain kernels of a given dimensionality, usually 1D, 2D, or 3D, which slide over the input and learn local features on parts of it. For common CNNs, the learned features are overlapping, which is achieved via "sliding" over the input. For 1D convolution, the kernels are sliding over one dimension of a 2D input. Given an input matrix of 5x10 and the convolution being performed over the first dimension, a kernel with a height of 1,2,3,4,5 and a width of 10 is possible. If choosing a height of 2, the output of the convolutional layer will be 4 features.

For the 2D convolution, the kernel width is smaller than the width of the input, and the kernel is sliding over the two dimensions of the input. For the given input matrix of size 5x10, a convolution with a kernel size of 2x2 would produce 4x9=36 features. The output features can be arranged in a 2D feature map, keeping the position of the learned features in the same position it was in the input. A convolutional layer can be followed by a *Max Pooling* layer. *Max Pooling* layers take parts of the output feature map and choose the maximum value in that area. This way, the network is forced to learn only relevant features; others are not propagated to the next layer. A parameter for the pool size has to be defined to indicate the size of the *Max Pooling* areas.

2.3 Gated Recurrent Units

For our ASR system, we employ Gated Recurrent Units (GRUs) (Cho et al., 2014), which represent a simpler variant of the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). GRUs are recurrent neural networks (RNNs), which aim at sequential learning tasks. They are state-of-the-art methods in these tasks. Figure 2.1 shows the GRU architecture.

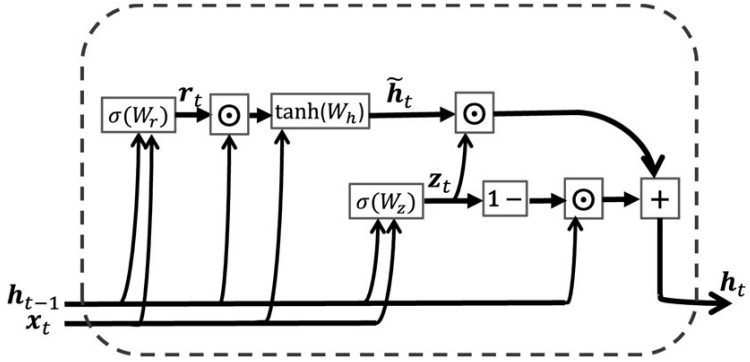


Figure 2.1: This figure shows the GRU architecture; image taken from Zhou et al. (2016).

A GRU consists of GRU cells, where each cell contains an update gate (z_t) and a reset gate (r_t), a hidden state (h_t), which is also the output, an input weight vector (W_h) and a hidden weight vector (U_h) for the new hidden state. The update gate controls how much information of past time steps is carried over to the next state. It also possesses input weights (W_z) and hidden weights (U_z) and is calculated with:

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1}) \tag{2.1}$$

where x_t is the input at time step t, and σ_g is an activation function providing values between 0 and 1, e.g., the sigmoid function.

The reset gate controls how much of the past state (h_{t-1}) is removed from the memory and is calculated similarly using the reset gate input weights (W_r) and the reset gate hidden weights (U_r):

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1}) \tag{2.2}$$

Using the reset gate state (r_t) a new state ($\tilde{h}_t$) can be calculated:

$$\tilde{h}_t = \sigma_h(W_h x_t + r_t \odot U_h h_{t-1}) \tag{2.3}$$

where $\odot$ is a Hadamard product (or element-wise product) and σ_h is an activation with values between -1 and 1, like the hyperbolic tangent.

The new state ($\tilde{h}_t$) is used to calculate the new hidden state (h_t):

$$h_t = \tilde{h}_t \odot z_t + h_{t-1} \odot (1 - z_t) \tag{2.4}$$

2.4 Connectionist Temporal Classification

RNNs are often used to perform sequence to sequence learning tasks. For example, in ASR, the input data is chunked into frames, while the output data may consist of characters or phonemes. In this case, a direct mapping from input chunk to output character is not possible, as a character may span over multiple chunks. The timescales are different in this case. A training sample consists of an audio file and the corresponding text but misses the exact content of each chunk of audio, an alignment is missing here. As there are fewer characters than chunks in a sample, we label the rest of the chunks with a blank label (−).

The idea now is to find a proper alignment of audio chunks and characters. To find this alignment, one could use a trained ASR system. However, we want to train a new ASR system, so this is not an option. Instead, we calculate all possible alignments by modulating all combinations of possible characters as a Hidden Markov Model (HMM). The probabilities for different combinations of outputs are summed up to get a score for the alignment. The probabilities are calculated by feeding the input through a randomly initialized RNN. The calculated score is used as the propagated error to adapt

the weights inside the network. This way, the network is forced to learn relevant information by aligning the input to the output. Connectionist Temporal Classification (CTC) was first introduced by Graves et al. (2006). We employ CTC to train our RNN-based ASR system.

During decoding, the input is fed through the network. The output of the network is a sequence of character distributions. The simplest decoding method is taking the argmax of each timestep, for example, `--H-EEE-L-LL-OOO-`. As a first processing step, characters spanning over multiple chunks, which are represented by multiple characters (`EEE`, `LL`), are reduced to one character (`E`, `L`). Also, all blank tokens are removed. In this case `--H-EEE-L-LL-OOO-` is transformed into `HELLO`. This way of decoding is called *Best Path Decoding* or *Greedy Decoding* and is the simplest decoding variant. There is a more sophisticated decoding mechanism called *Beam Search Decoding* that we describe in the next subsection.

2.5 Beam Search Decoding

As mentioned in Section 2.4, we use CTC to train our ASR system. The output of the acoustic model, which is a sequence of character distributions, can be decoded using a *Greedy Decoder*, which only chooses the most probable character from each time step. It lacks information about spelling and language rules. This issue can be addressed when employing a language model for decoding the output of a neural ASR system. Those language models are usually statistical N-gram models. They are trained on a corpus of sentences by calculating the probabilities for one word following another word. This variant is called a bigram model. When two words are used to predict the next one, this is called a trigram model and so on. The probabilities are calculated by counting how often one word follows another word or two other words etc. Afterwards, the counted numbers are normalized to produce a probability for each word. Usually, these models use start and end tokens that mark the start

and the end of an utterance. Start tokens can be used to predict the first word of the utterance.

It is now possible to score the first part of the input sequence of character probabilities against all the words following a start token. For each of these words, the next possible words are used to be scored against the next part of the input by performing a best-first search. Obviously, this results in a combinatorial explosion and is not practical in most applications. Instead, only the most promising paths are kept. This is achieved by ordering the paths by their probability and keeping the most probable ones, which is called a beam search. That is why this decoder is called *Beam Search Decoder* (Hannun et al., 2014b). It is possible that not the best path is selected when decoding is finished, because it may have been pruned before. The beamwidth parameter defines how many paths are kept while decoding, the more paths, the higher the chance of keeping the best path. The more paths are kept, the more computational effort is required to decode the input. The beamwidth needs to be chosen to deliver good results while not spending too much processing time; it needs to be chosen depending on the application. Our ASR system has different variants, using a *Greedy Decoder* (described in the previous subsection) or a *Beam Search Decoder*.

2.6 Word Embeddings

For NLP tasks, word embeddings are usually used as word representations, especially for inputs of neural networks. Instead of encoding a word as a one-hot vector representation, the embedding vector contains a distributed representation with values between e.g., -1 and 1 in all slots of the vector. To generate word embeddings, no labelled data is required; unlabelled text is sufficient.

In this work, we employ *skip-gram* word embeddings, which are gener-

ated by training a neural network model. The model consists of an input layer receiving a one-hot encoded word vector. For generating the word vectors, the number of different words in the corpus has to be determined, for example 10,000. The input layer is connected to a hidden layer with usually 300 neurons. The hidden layer has a linear activation function and is connected to an output layer with again e.g. 10,000 units. The model is trained by predicting the context of a word by the word itself. For the sentence `the traffic light is green`, one would generate the following training samples:

- `(the, traffic)`
- `(traffic, the)`
- `(traffic, light)`
- `(light, traffic)`
- `(light, is)`
- `(is, light)`
- `(is, green)`
- `(green, is)`

For this example, the window size is 1, meaning there is taken one word before the input and one word after the input as a context word. The context words for `light` are `traffic` and `is`. If the window size is larger, more context words are taken for training.

The network is quite vast, possessing $10,000 * 300 + 300 * 10,000 = 6,000,000$ weights. 10,000 words are considered to be a small vocabulary; common word embeddings are often trained on 3,000,000 input words, making the network hard to train. It is only possible in reasonable time, because of three optimization steps. First, only a small amount of computations needs

to be performed in the first part of the network. As the input representation is one-hot encoded, 9,999 weights need to be multiplied by 0 leading to a 0-activation, one needs only to compute the activation of weights connected to the activated input word.

The second step is to remove words with low relevance from the context. For example the word `traffic` contains more context information for the word `light` than the word `is`. To determine if a word is relevant for the context, the occurrence of each word is counted. This frequency is used as a probability to randomly remove the word from the context while training a sample. Even with using this technique, many output activations need to be computed.

The third step to reduce the computational effort is called *negative sampling*. Here, the aim is not to adjust all hidden-to-output weights, but only a small portion of them. As the output is again a one-hot vector, the weights connected to the activated output are adjusted. Then, only a small portion of negative samples is selected, which means outputs that are there for the given input training sample. The words are again chosen using their frequency in the corpus. A high frequency in the corpus is considered to be more relevant. The negative samples are selected using random sampling based on their occurrence probability. Instead of computing 3,000,000 weights (for 10,000 outputs), one could calculate 3,000 (for 10 outputs). For commonly used corpora, the vocabulary size is even larger (3,000,000 words), so the mentioned tricks are the only way to train the network.

After training, the input and output layers are dropped; only the hidden weights are kept. They contain a matrix of 10,000x300 or 10,000 vectors of size 300. These vectors are the word embeddings. The described type of word embeddings is referred to as *Word2Vec* (Mikolov et al., 2013b,a). For the presented work we employ an extension called *fastText* (Bojanowski et al., 2017). Often, rare words are modelled badly or not at all due to their rare occurrence. For this reason, *fastText* also learns subword embeddings that can

be used to construct the word embeddings for rare and unknown words. For example, the word `traffic` is split to the subwords `<tr`, `tra`,`raf`, `aff`, `ffi`, `fic` and `ic>`, and also the word `<traffic>` itself. These subwords are then taken to train the model. Word embeddings can be constructed from the subword embeddings afterwards.

2.7 Answer Set Programming

To be able to interpret predicate structures, we choose a logic programming paradigm, as language predicates can be transferred to logic predicates. As a method, we chose *Answer Set Programming* (ASP) (Lifschitz, 1999; Baral, 2003), a declarative language similar to *Prolog*. One of the differences between Prolog and ASP is the execution order, Prolog's statements are executed in the order they are listed, ASP is independent of this order. Logic predicates are a well-fitting representation, as parts of a predicate description can be removed or changed while keeping the whole description valid or at least syntactically correct. ASP consists of facts, rules, and integrity constraints. ASP facts consist of a predicate describing arguments and evaluate to `true`. The following predicates describe the weather and the sun being nice today:

Listing 1 Two facts in ASP.

```
1 nice(weather,today).
2 nice(sun,today).
```

ASP rules consist of a head and a body. When a rule head is evaluated, the rule body needs to be evaluated. A rule is true if all elements in the rule body are true. The following rule defines a day to be a good day if there are nice weather and sun: This rule can be used in different ways. When asking for `goodday(Day)`, ASP will list all days that have nice weather and sun. When asking for `goodday(today)`, the answer will be `true`. Also, ASP can be used to validate commands and to find execution parameters. The difference is

Listing 2 A rule in ASP.

```
1 goodday(Day) :- nice(weather,Day), nice(sun,Day).
```

that arguments with lower case letters are used as constants, while arguments starting with capital letters are interpreted as variables. The rule body can also be used to perform basic calculations. Rules can also be used to create conditions. For example, a day is always considered as good if the weather is nice, but if there is no nice sun, it is not a good day:

Listing 3 A condition in ASP.

```
1 goodday(Day) :- nice(weather,Day),
2                 #false: not nice(sun,Day).
```

Another kind of description is integrity constraints. They consist of a rule without head and are used to form a consistent world. For example, the following integrity constraint makes sure that a day cannot have nice and bad weather at the same time:

Listing 4 An integrity constraint in ASP.

```
1 :- nice(weather,Day), not nice(weather,Day).
```

The described tools of ASP can now be used to form a scenario and find concrete execution parameters from commands. In this work, we use the *clingo* framework (Gebser et al., 2011) to interpret our ASP description.

2.8 Needleman-Wunsch Algorithm

The *Needleman-Wunsch Algorithm* (Needleman and Wunsch, 1970) is used to calculate an alignment of two sequences having a different length. It is

possible to use a custom similarity score function that scores the elements of the sequence. Also, it is possible to define a custom gap cost function. The algorithm is defined as follows:

$$M(0,0) = 0 \tag{2.5}$$

$$M(i,0) = M(i-1,0) + f(i), 1 \leq i \leq m \tag{2.6}$$

$$M(0,j) = M(0,j-1) + f(i), 1 \leq j \leq n \tag{2.7}$$

$$M(i,j) = \max \begin{Bmatrix} M(i-1,j-1) + w(a_i, b_j) \\ \max_{1 \leq k \leq i}\{M(i-k,j) + f(k)\} \\ \max_{1 \leq l \leq j}\{M(i,j-l) + f(l)\} \end{Bmatrix}, 1 \leq i \leq m, 1 \leq j \leq n \tag{2.8}$$

with:

- a, b: the sequences to be aligned,
- m, n: the length of a and b
- $M(i,j)$: the maximum similarity score of a prefix of a ending at i and a prefix of b ending at j
- $w(c,d)$: similarity score function
- f: gap cost function

The output of the algorithm is created by taking the number in $M(i,j)$ which represents the score of the alignment. Also, it produces an alignment of the two sequences. The idea of the algorithm is to provide an extension of the Levenshtein distance presented in Section 3.5 to support a custom similarity score function. We employ the *Needleman-Wunsch Algorithm* to perform semantic evaluation between input and output.

Chapter 3

Related Work

3.1 Introduction

In this chapter, we give an overview of the methods used for related work that inspired our approaches or with which we compare our approaches. First, we described the Blocks World scenario, a grid world containing cubes and prisms, together with a data set about this scenario. The presented data set is extended and used for our experiments. A detailed description of our novel extended data set can be found in the experiments section (5.2.2). Afterwards, a related work that employs Convolutional Neural Networks to perform natural language processing is introduced, which was an inspiration for our *SemaPred Recognizer*. Next, we give a brief introduction to cloud-based speech recognition, namely the service offered by Google, with which we compare our ASR system. Thereafter, we present *Phonemic Postprocessing*, one of our previous works. Its mechanism is adapted to be employed by our *Sentence Templates* language models. The following section contains information about approaches using ASR in HRI scenarios. Finally, we present approaches containing dependency parsing on ungrammatical text, which is a similar task to recognizing *SemaPreds* on ungrammatical text like the output of ASR systems.

3.2 The Blocks World Scenario and the Train Robots Data Set

The models we present in this thesis are aimed at improving the ASR and NLP performance in restricted domains. To test our models, we chose a domain called *Blocks World*. It was originally described by Winograd (1972) in his work about the NLU system SHRDLU. Winograd's blocks world was adapted by Dukes (2013a,b, 2014b) to develop a robot scenario in which NLP systems like parsers can be tested by creating the *Train Robots* data set. The blocks world consists of a grid containing 8*8*8 discrete positions. On the grid, there may be differently shaped objects like cubes and pyramids, possessing different colors. Next to the grid, there is a robot arm that is able to grasp the objects and move them around on the grid. Figure 3.1 contains an example scene.

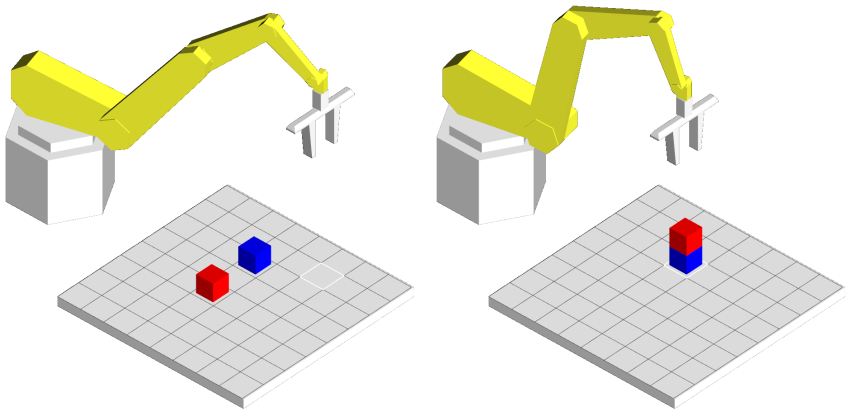


Figure 3.1: Example from the corpus. An example for a board scene before (left board) and after (right board) the command: `Move the red brick on top of the blue brick`.

The aim of Dukes' work (Dukes, 2013a) was to provide a robust parser and a spatial planner that is able to execute natural language command within the mentioned blocks world. The natural language commands can be arbitrarily complex and nested. The first step was to collect a data set containing realistic natural language and a machine-readable and executable NLP rep-

resentation. For this purpose, Dukes presented the Robot Control Language (RCL). It consists of visual scenes as shown in Figure 3.1 and a natural language command directed to the robot. The left scene is the initial scene; the command describes the action performed by the robot arm to transition into the subsequent scene shown on the right. The RCL annotations for the command shown in Figure 3.1 is shown in Listing 5.

Listing 5 Definition of shapes

```
1  (event:
2   (action: move)
3    (entity:
4     (id: 1)
5     (color: red)
6     (type: cube)
7    (spatial-relation:
8     (relation: above)
9     (entity:
10     (color: blue)
11      (type: cube)))))
```

It consists of a hierarchical machine-readable structure describing the action to be performed by the robot arm. At the leaves of the tree structure, one can find prototype words like `cube`, while the input word was *brick*. Each word connected to a leaf is transformed into its prototype word. The data set contains a fixed set of prototype words. The structure can then be interpreted by a spatial planner to perform the action. The scenes are annotated using XML structures (Listing 6). The RCL commands possess a tree-like structure. They can be used to derive a graphical tree which is depicted in Figure 3.2.

The data set consists of 1,000 scenes, each containing two layouts. Crowdsourcing was used to collect the natural language commands. For this purpose, a website[1] was installed. On the website, users could take part in a competitive online game. The players were presented scenes and had to

[1] http://www.trainrobots.com

Listing 6 XML structure describing the layout of the left scene of Figure 3.1

```
<layout id="1">
        <gripper position="7 5 7" open="true" />
        <shapes>
                <cube color="red" position="3 3 0" />
                <cube color="blue" position="5 3 0" />
        </shapes>
</layout>
```

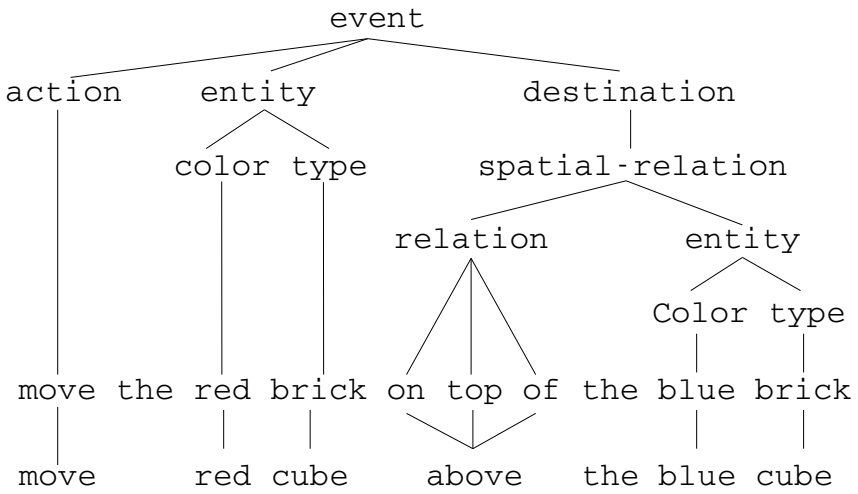


Figure 3.2: The tree representation of the RCL command shown in Listing 5.

provide natural language commands describing the scenes. The players were scored by other players providing a score between 1 and 5 measuring the quality and correctness of the generated commands. This way, 10,000 commands have been collected. 3,409 commands were usable and annotated manually using RCL annotations. Other commands contained errors produced by the users like confusing images, perspectives or providing spam commands. As the commands were collected using online forms, they contained many grammatical and spelling errors and did not represent a realistic transcription for spoken commands.

To evaluate our approaches, we extended the data set by correcting grammatical and spelling errors and recorded audio data. We also reanno-

tated the data using a novel representation, which is presented later in this thesis. There are different publications describing the *Train Robots* data set (Dukes, 2013a,b). It was used within an open NLP challenge called SemEval 2014 Task 6 (Dukes, 2014b). We also presented an approach using the original data, which is not described in this thesis (Twiefel et al., 2016b). Also, Dukes presented an approach, which used a different version of the data set (Dukes, 2014a) and therefore cannot be compared to the other approaches. The data set is available on the website of SemEval 2014, Task 6[2].

3.3 Sentence Classification using Convolutional Neural Networks

NLP tasks often contain classification and labelling tasks. Usually, classification tasks differentiate between multiple classes. Our work of recognizing *SemaPreds* in a restricted domain corpus is comparable to multiclass classification. The work of Kim (2014) describes a sentence classification task where a whole sentence has to be recognized, for example, as positive or negative. Kim employs CNNs for this task. Input sentences are transformed into word embedding sequences. As sentences are of arbitrary length, and CNNs work on fixed-length vectors, a maximum length is defined, and each input sequence is padded to reach this length. The first layer consists of convolutional layers with different filter sizes. Then, the filters are concatenated, and a *Max-over-Time Pooling* (Collobert et al., 2011) is performed, which means that the maximum activation for a filter over the whole sequence is kept, and the other activations are dropped. This layer is followed by a dense layer with softmax activation containing the class labels for the classification task. Our presented NLP approach is inspired by the work of Kim (2014).

[2]http://alt.qcri.org/semeval2014/task6/index.php?id=data-and-tools

3.4 Cloud-based Speech Recognition

Speech recognition systems like Google Speech Recognition (Sak et al., 2015, 2014; Sainath et al., 2015) or Baidu's Deep Speech 2 Amodei et al. (2015) employ a deep Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) together with Connectionist Temporal Classification (CTC) (Graves et al., 2006). LSTMs are able to learn long-range dependencies inside a sequence, compared to N-gram Hidden Markov Models (HMM), which are only able to model the local context. Frames of audio data are taken as input and trained to produce, e.g., phones, letters, or words. The problem of the different timescales between acoustic frames and labels is solved using CTC, which introduces a *blank* label to fill the gaps and performs an alignment between the two sequences. The outputs are postprocessed by a 5-gram statistical language model both for Google's and Baidu's ASR. For our experiments, we use the free web API[3] of Google's Search by Voice. As there is no information about the exact architecture behind the web API, we can only expect that it is the architecture just mentioned.

3.5 Phonemic Postprocessing

Our previous work (Twiefel et al., 2014) suggests that domain knowledge helps in improving the results of deep-neural-network-based (DNN) speech recognition by postprocessing them using this knowledge. Traditional speech recognition commonly used before the success of *Deep Learning* (LeCun et al., 2015) in general consists of an acoustic model that generates phonemes from acoustic data and a language model that generates words based on a grammar or a statistical n-gram model. The phoneme representations of these words are then scored against the phoneme sequences generated by the acoustic model to produce a probabilistic word sequence hypothesis. As described in Section

[3]https://pypi.org/project/SpeechRecognition/

3.4, state-of-the-art ASR can also generate an intermediate representation like phonemes. However, the acoustic model and the language model can also both be included in a large DNN, and phonemes are not necessary in this case as words are being generated directly.

The acoustic models used for traditional speech recognition are based on Mel Frequency Cepstral Coefficients (MFCCs) (Mermelstein, 1976), which are used to extract human speech from the audio signal. The acoustic model is trained on MFCC features derived from speech and the corresponding phoneme sequences.

A phoneme is the smallest meaning-distinguishing unit to express a word. Compared to text, a character would be the smallest meaning-distinguishing unit. Phonemes can be uttered using different phones or speech sounds, which makes phonemes a superclass of phones. Comparing this to characters again, a character can be expressed using different fonts. By this definition, a phoneme is speaker-independent, making it a suitable intermediate representation that can be used for scoring. We hypothesize that a phoneme is also spoken the same way independently from its domain, meaning phonemes are spoken the same way in a kitchen, football or Human-Robot Interaction context, which would make acoustic modeling domain-independent, and acoustic models could be transferred from one domain to another.

Another hypothesis is that language models are domain-dependent, as the training data for a model should follow the same distribution as the data of the environment it is used in, and this is only true if a general-purpose model is used in domains which are a subset of a general-purpose domain. If a general-purpose language model is used only inside a specific domain that does not follow the same distribution as the general-purpose domain, the language model is not the optimal one for this domain.

For this reason, we proposed a unified ASR system that consists of a large and well-trained DNN-based domain-independent acoustic model combined

with a domain-specific language model (Twiefel et al., 2014). Due to the nature of DNNs to require large amounts of data and the lack of this data in small domains, we recommended using traditional language modeling like statistical n-gram models and grammars.

One of the approaches is based on the traditional open-source ASR system Sphinx-4 (Lamere et al., 2003), which uses HMMs for acoustic modeling and a Viterbi decoder to find the best word sequence hypothesis. As the acoustic model relies on HMMs and is trained on a limited amount of labeled acoustic data compared to the massive amount of data companies like Google are able to generate and process, the acoustic model is the weakness of the Sphinx-4 system. The scoring is performed on the phoneme level, which offers the possibility to remove the acoustic model from Sphinx and directly feed in phoneme sequences. Instead of training our own domain-independent acoustic model, we employ the massive acoustic models of e.g. Google by delegating the acoustic processing to Google's Search by Voice. The unified system is called *DOCKS* (Twiefel et al., 2014) and supports language models in the form of grammars (*DOCKS Grammar*) or statistical bigram models (*DOCKS Bigram*).

Google's hypothesis for the reference text `'addressed mail'` could be something similar to `'a dressed male'`, which is completely incorrect on the word level. On the phoneme level, both grapheme sequences can be represented as `'AH D R EH S T M EY L'`. We employ the trainable grapheme-to-phoneme converter SequiturG2P (Bisani and Ney, 2008) and train it on CMUdict 0.7a [4] to be able to generate a phoneme sequence for any grapheme sequence coming from Google's ASR. These phoneme sequences are then fed to our postprocessing system. We showed that this principle works better than using the given acoustic models of Sphinx-4 (Twiefel et al., 2014).

Another approach contained in the *DOCKS* system is called *DOCKS*

[4] `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`

Sentencelist. If a list of all possible sentences that can be uttered in a restricted domain is known beforehand, this restricted but robust approach can be used. The approach is based on the Levenshtein distance (Levenshtein, 1966), which is a standard method to calculate a distance score between two sequences a and b, with i and j being the recursively processed indices of the sequences:

$$\mathcal{L}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathcal{L}_{a,b}(i-1,j) + 1 \\ \mathcal{L}_{a,b}(i,j-1) + 1 \\ \mathcal{L}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{3.1}$$

We convert the ten best hypotheses from Google's ASR to phoneme sequences and do the same for the list of expectable sentences. Then, a normalized Levenshtein distance is calculated over all ten best phoneme sequences (H) against all phoneme sequences of the sentence list (S):

$$\lambda \;=\; \text{argmin}\; \mathcal{L}_{h_k,s_l}(|h_k|,|s_l|) \tag{3.2}$$

where $\mathcal{L}$ is the Levenshtein distance. The confidence value was computed as

$$\gamma^A = \max(0, 1 - \frac{\mathcal{L}_{h_k,s_l}(|h_k|,|s_l|)}{|s_l|}) \tag{3.3}$$

with $h_k \in H$ (set of the ten best hypotheses) and $s_l \in S$ (set of reference sentences) both in phonemic representations. As this approach is the most restricted one, it performed best (WER around 0%) if all spoken sentences are known in advance (Twiefel et al., 2014). The method of rescoring ASR hypotheses on phoneme level using the Levenshtein distance is called *Phonemic Levenshtein Scoring* and employed by the language models presented in this thesis.

3.6 Speech and Language Processing in Human-Robot Interaction Scenarios

In this section, we describe related approaches using ASR in human-robot interaction (HRI) scenarios. Bastianelli et al. (2014) present a related work in the HRI domain. It consists of a processing pipeline containing a general-purpose ASR (Google Android). The next step is to perform syntactic parsing with a general-purpose parser. The following steps are frame prediction, boundary detection, argument classification and command generation. They measure precision, recall and F1-score in different experiments, which seem to be quite low (e.g. precision: 0.36; recall: 0.20; F1-score: 0.26 in one of experiment). A realistic performance in form of accuracy is not provided, but we expect it to be low.

Another approach is provided by Bastianelli et al. (2017). They present a pipeline consisting of a general-purpose ASR (Google Android), a morpho-syntactic analysis by creating a dependency graph, speech re-ranking, action detection, full command recognition, action grounding and argument grounding. Again, the performance is not evaluated regarding accuracy.

Other systems within the HRI domain employ simpler architectures like the work of Pleva et al. (2017) who employ the Julius ASR system (Lee and Kawahara, 2009) together with a two-word grammar-based language model. The system does not contain any sophisticated natural language processing (NLP) approach. Kennedy et al. (2017) employ different ASR systems (Google, Microsoft, PocketSphinx) to communicate with a Nao robot. The architecture is minimalistic, not containing any NLP systems. Manzi et al. (2017) employ Google's ASR to navigate a robot without further NLP processing. Fakhruldeen et al. (2016) use the declarative programming language *Prolog* to instruct a robot. They connected a not-mentioned ASR system to provide the input; but did not measure the performance of the system.

All mentioned approaches were not evaluated regarding accuracy. We do not believe that these approaches perform well in practice, especially because they lack a strategy to handle ungrammatical inputs like the text outputs of ASR systems.

3.7 Dependency Parsing on Ungrammatical Text

This section contains approaches for dependency parsers to handle inputs coming from ASR systems or other ungrammatical text inputs. There are different approaches concentrating on handling text coming from ASR systems that are processed by dependency parsers. The approach of Yoshikawa et al. (2016) introduces an error label to the tree. It is trained by matching gold standard training trees with trees generated from ASR output. The mismatches are labeled and the parser learns to handle the errors produced by the ASR system on the training data. The approach is not evaluated regarding accuracy and we expect that it may not work in practice. Bechet et al. (2014) adapt a syntactic parser to handle errors produced by an ASR system. There are no details mentioned about the ASR system and the system is not evaluated regarding accuracy making it not possible to determine the usefulness for robot applications. Honnibal and Johnson (2014) present a dependency parser able to handle disfluency in the text. They are able to successfully correct dependency trees that are created on incorrect data. We do not consider this approach to be capable of correcting ASR errors, because disfluent sentences contain the correct information while sentences from ASR systems may not. Sakaguchi et al. (2017) present a dependency parser that is able to handle ungrammatical text input. The system may be potentially useful when connected to an ASR system. It is not clear to which degree the system can handle ungrammatical input.

The approaches presented in this section provide strategies to handle ungrammatical text input. We believe that it is also important to differentiate between different sources leading to this ungrammaticality. Parsers able to handle disfluency are not suitable, as disfluent utterances contain the needed information, ASR outputs may not. The only strategy that seems to be more useful is the one of Yoshikawa et al. (2016). The problem here is, that it needs learn the type of errors produced by an ASR system. We do not believe, that it is possible to learn all different kinds of errors an ASR system may commit, but it may be useful to correct a portion of the errors.

Chapter 4

Novel Approaches Developed in this Thesis

4.1 Introduction

In this section, we present the novel approaches we developed. We designed a processing chain from audio data coming from a microphone to a simulated or real robot performing instructions uttered by a user. To be able to achieve this goal, we developed a novel machine-readable language representation called *SemaPreds* that we describe in the first subsection. These *SemaPreds* can be recognized from natural language sentences using our *SemaPred Recognizer* depicted in the following subsection. Next, we introduce our ASR system called *SlimSpeech*. Thereafter, we describe our external language model called *Sentence Templates*. Afterwards, we introduce our *SemaPred Interpreter* that is able to process the recognized *SemaPreds* and generate concrete execution parameters for the simulator or robot. A benefit of our novel *SemaPred* representation is that its quality can be evaluated. For this purpose, we developed the *Semantic Evaluator* presented in the following subsection. It is employed

by our *Crossmodal Corrector*, found in the following subsection, which uses this information to revise incorrect parts using a (simulated) visual modality. The *Crossmodal Corrector* also uses the *SemaPred Interpreter* to generate revised execution parameters. Finally, we describe our simulator and a real-world robot application that employs the generated execution parameters to perform actions.

4.2 Semantic Logic Predicate Representation

In this section, we describe our novel natural language representation called *Semantic Logic Predicates* (*SemaPreds*). One strategy to represent natural language in speech-controlled scenarios like human-robot interaction (HRI) scenarios is using keywords. These kinds of representation is limited and only suitable for simple scenarios. Another strategy is to recognize tree-like structures like dependency trees (see Section 3.6 and 3.7), or RCL trees (see Section 3.2). Dependency parsers are trained on clean text data and there are only limited strategies to handle ungrammatical text input like text coming from ASR systems (see Section 3.7).

Our strategy is developing a novel representation that can be validated for correctness. If a sentence had been recognized wrongly by the ASR system, the output of the NLP model would be wrong in most of the cases. To simplify the correction of the NLP output, the novel structure should be separable in valid and invalid parts. For this purpose, the wrong parts should be removable to check the validity of the rest of the structure. We chose logic predicates as a suitable representation. Logic predicates can be validated using logic programming languages like Prolog and Answer Set Programming (ASP). This is only possible to a certain degree when operating in the general-purpose domain. As our presented ASR system operates in restricted domains (Twiefel et al., 2014) , the connected NLP system will also work in restricted domains For our given HRI scenario, which contains robot instructions in a grid world,

we define three different kinds of logic predicates representing information of natural language instructions, namely *actions*, *attributes*, and *relations*. The following example shows a simple instruction and its corresponding *action* predicate:

`move the prism on the cube`
- `move(prism, on, cube)`

Another example could be:

`put the pyramid on top of the box`
- `put(pyramid, top, box)`

Both sentences possess the same information. We improve the quality of our training data by reducing the number of possible labels. For this purpose, words having the same meaning are clustered and replaced by a semantic prototype word. The semantic representation of both sentences is:
- `move(prism, above, cube)`

To summarize, an *action* has the signature:
- `action(entity, relation, entity)`

Another predicate type is *attributes*. These may contain colors or other properties of an entity. An example containing *attributes* is the following:

`put the` **`red`** `pyramid on top of the` **`blue`** `box`
- `move(prism, above, cube)`
- **`red`**`(prism)`
- **`blue`**`(cube)`

The signature of an *attribute* predicate is:
- `attribute(entity)`

A problem arises when a sentence contains objects of the same type, like:

`put the red box on top of the blue box`

- `move(cube, above, cube)`
- `red(cube)`
- `blue(cube)`

It is not possible to distinguish the two cubes. For this purpose, we introduce indices for the different entities:

- `move(cube`$_1$`, above, cube`$_2$`)`
- `red(cube`$_1$`)`
- `blue(cube`$_2$`)`

The indices occur in the order of corresponding words of the original sentence. Another predicate type is the *relation* predicate. It describes the relation between two entities. The following example contains a *relation* predicate:

`put the red box on top of the blue` **`box that is in the corner`**

- `move(cube`$_1$`, above, cube`$_2$`)`
- `red(cube`$_1$`)`
- `blue(cube`$_2$`)`
- **`is(cube`$_2$`, above, corner`$_3$`)`**

The signature of a *relation* predicates is:

- `is(entity, relation, entity)`

It is also possible to use multiple actions, relations, and attributes. A complex example is the following:

`grab the red pyramid located on the blue box in the bottom left corner and put it on top of the blue box that is on top of the blue brick that is next to the white pyramid`

- take(prism$_1$, above, cube$_2$)
- is(cube$_2$, above, corner$_3$)
- red(prism$_1$)
- blue(cube$_2$)
- back(corner$_3$)
- left(corner$_3$)
- drop(prism$_1$, above, cube$_4$)
- is(cube$_4$, nearest, prism$_5$)
- blue(cube$_4$)
- white(prism$_5$)

SemaPreds can be annotated using a comma-separated value (CSV) representation. For this example, it would look like:

Listing 7 CSV representation of *SemaPreds*

```
1  take, prism_1, above, cube_2
2  is, cube_2, above, corner_3
3  red, prism_1, ,
4  blue, cube_2, ,
5  back, corner_3, ,
6  left, corner_3, ,
7  drop, prims_1, above, cube_4
8  is, cube_4, nearest, prism_5
9  blue, cube_4, ,
10 white, prism_5
```

To summarize, the representation uses semantic prototype words to represent words of a sentence in a machine-readable form. The form consists of a logic

predicate representation. For this reason, we refer to the novel representation as *Semantic Logic Predicates* or *SemaPreds*. The limitations of the approach need to be evaluated. So far, we tested to annotate data in one domain, but we believe that this is also possible in other domains. Dependency parsers may not be able to recognize *SemaPreds*, as there is no direct link between the slots of a *SemaPred* and the words of a sentence.

4.3 Semantic Logic Predicate Recognizer

In this section, we describe our method to learn *SemaPreds* in a restricted domain. For NLP, the method of choice is usually a recurrent neural network (RNN), often a Long Short-Term Memory (LSTM). As RNNs require a huge amount of training data, and there is a lack of training data in restricted domains, we chose an alternative to address this issue. We employ a convolutional neural network (CNN), which requires less training data and which was already successfully applied for sentence classification by Kim (2014) (see Section 3.3) and the architecture is depicted in Figure 4.1.

As CNNs operate on vectors, we need to convert sentences to a fixed-length representation. We do this by defining a maximum length of processable words (40 in this case). The words are converted to word embeddings using *fastText* (Bojanowski et al., 2017) (see Section 2.6). The dimensionality of *fastText* embeddings is 300, so the input data has a shape of 40*300. To fill in the missing vectors for a sentence shorter than 40 words, we employ padding by adding padding vectors. Instead of adding a vector containing only zeros, we add an embedding of a word not used in the data set (*null* in this case).

Like Kim (2014), we employ a one-dimensional convolution to the input. The filter size varies from 1 to 13, meaning there are 300*1, 300*2, 300*3 ... 300*13 convolutional filters. This architecture adapts the idea of n-grams containing relevant information by performing a kind of *chunking* similar to tra-

ditional shallow parsing. In dependency parsing, a nested and often complex dependency tree is constructed, which also contains long-range dependencies. To imitate this behaviour, several convolutional layers could be stacked. As deeper networks require more training data, we decided against this option. Instead, we chose filter sizes from 1 to 13, which modulate both short- and long-range dependencies.

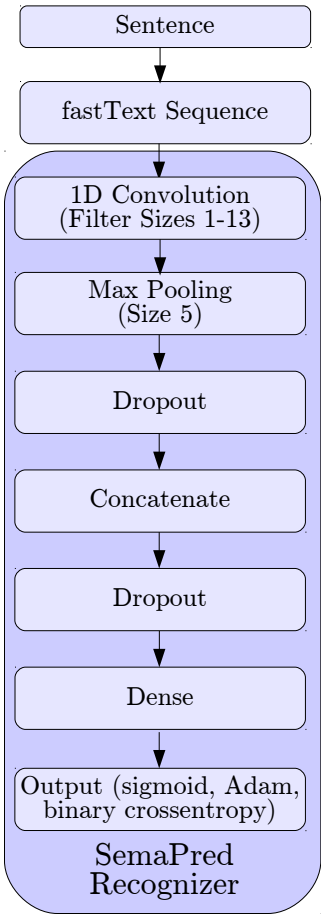


Figure 4.1: The architecture of our SemaPred Recognizer. The output of the system is a vector that contains a list of encoded *SemaPreds*. The encoding is explained later inside this section.

Consider the sentence: `grab the red and blue box and put it on the green block`. For this sentence, a filter of size 6 that would be able to learn the relation between `grab` and `box` would look like:

- **grab** the red and blue **box** and put it on the green block

Short filter sizes should be used for short-range dependencies like entities and their attributes and larger filter sizes should be used for long range dependences like actions and their entities or nested sentences containing multiple *Relation SemaPreds*. For the maximum filter size we chose 13, which is the average length of a training sentence in our data set.

The one-dimensional convolution layer is followed by a *Max Pooling* layer. Kim (2014) has chosen *Max-over-Time Pooling* (Collobert et al., 2011), as the purpose of his work to classify a whole sentence. For our task, ordinary *Max Pooling* is useful, as relevant information needs to be extracted from a sentence, and the same information may be relevant multiple times. The following example shows this:

take the **green block** in the corner and put it on the **green block** that is on the red block

In this case, the filter which learned green block is activated twice. When using *Max-over-Time Pooling*, the information of one filter will get lost. For this reason, we chose ordinary *Max Pooling* with a size of 5, which should be a good number to prevent the overlay of the same activated filter within the *Max Pooling* range. For regularization, *Dropout* is applied. Afterwards, the activations of the different filters are concatenated to a vector. Again, a *dropout* is applied. The concatenated activation vector is connected to the output layer via a hidden dense layer with a *Rectified Linear Unit* (ReLU) activation function. The parameters and the structure of the architecture is shown in Listings 8 and 9.

The output layer encodes *SemaPreds* into a binary vector. For the encoding, the maximum number of *actions*, *relations*, *entities,* and *attributes* must be defined. For the given data set, the maximum number of *actions* per sentence is 2, the maximum number of *relations* is also 2, the maximum

number of *entities* is 5, and the maximum number of *attributes* is 20 (4 per *entity*). As most *SemaPreds* representations possess less, the rest of the slots has to be filled with an empty token (EEE). The following example illustrates this:

- move(prism$_1$, above, cube$_2$)
- is(cube$_2$, above, corner$_3$)
- red(prism$_1$)
- blue(cube$_2$)
- back(corner$_3$)
- left(corner$_3$)

This *SemaPred* representation is created as follows to possess a coherent output representation (Figure 4.2):

Action 1-2	**move(prism_1, above, cube_2)**	EEE(EEE, EEE, EEE)			
Relation 1-2	**is(cube_2, above, corner_3)**	is(EEE, EEE, EEE)			
Attribute 1-5	**red(prism_1)**	EEE(**prism_1**)	EEE(**prism_1**)	EEE(**prism_1**)	EEE(**prism_1**)
Attribute 6-10	**blue(cube_2)**	EEE(**cube_2**)	EEE(**cube_2**)	EEE(**cube_2**)	EEE(**cube_2**)
Attribute 11-15	**back(corner_3)**	**left(corner_3)**	EEE(**corner_3**)	EEE(**corner_3**)	EEE(**corner_3**)
Attribute 16-20	EEE(EEE)	EEE(EEE)	EEE(EEE)	EEE(EEE)	EEE(EEE)
Attribute 21-25	EEE(EEE)	EEE(EEE)	EEE(EEE)	EEE(EEE)	EEE(EEE)

Figure 4.2: The *SemaPred* template for an example command. The missing *SemaPreds* and slots have to be filled with empty tokens (EEE). *Relation SemaPreds* always start with the word *is*.

The dataset contains 3 different actions, 21 different attributes, 10 different entities and 9 different relations. For each category, an empty token (EEE) is added, which represents an empty slot inside a predicate. To encode an *Action SemaPred*, a vector is created by concatenating 4 localist vectors. The first vector contains the action and has a size of 5 (4 actions+1 empty

token). The second vector contains the index for the first entity. For the dataset, the maximum number of entities is 5, so the vector has a size of 6 (5 indices+empty token). The third vector encodes the relation and has a size of 10 (9 relations+empty token). The fourth vector contains the index of the second entity and has the same shape as the second vector. Fig. 4.3 illustrates the *Action SemaPreds* vector.

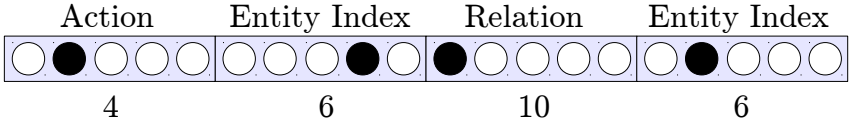


Figure 4.3: The schematic binary encoding of an *Action SemaPred*.

Relation SemaPreds are represented similarly. As they describe the relationship between two entities, they are encoded exactly like an *Action SemaPred*, excluding the action vector. Fig. 4.4 illustrates the *Relation SemaPreds* vector.

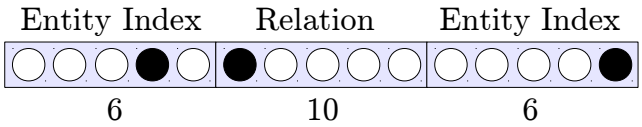


Figure 4.4: The schematic binary encoding of a *Relation SemaPred*.

To encode an *Attribute SemaPred*, the process is different. It consists of 5 localist[1] vectors. The first vector contains the attribute and has a size of 22 (21 attributes+empty token). As, for this data set, a *SemaPred* representation may contain 4 attributes per entity, the first four vectors encode the different entities. The last vector encodes the entity itself. Unlike for *Action SemaPreds* and *Relation SemaPreds*, it does not encode the index of the entity but the entity itself. The size of the vector is 11 (10 entities+empty token). Fig. 4.5 illustrates the *Attribute SemaPred* vector.

To produce the whole output vector, all vectors are concatenated. It consists of 2 action vectors, 2 relation vectors and 5 attribute vectors in this

[1] also called one-hot vectors

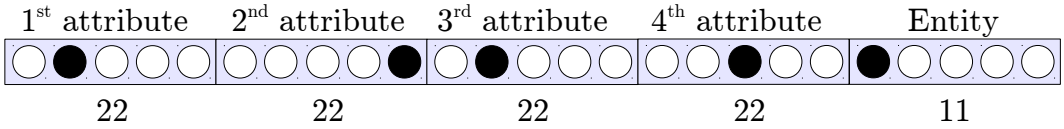


Figure 4.5: The schematic binary encoding of an *Attribute SemaPred*.

order. The output vector size is $2*(4+6+10+6)+2*(6+10+6)+5*(22+22+22+22+11)=591$ for the given data set. To be able to produce binary outputs, we chose a sigmoid activation function. As *softmax* is not applicable on the whole vector due to its sparseness, we chose binary cross-entropy as a loss function instead of categorical cross-entropy loss. For inference, we perform a winner-takes-all on each localist vector inside the output vector. The text output is produced by taking the output template and removing all *SemaPreds* containing only empty tokens. The whole model is trained using the *Adam* optimizer (Kingma and Ba, 2014).

4.4 Automatic Speech Recognition: SlimSpeech

When creating a processing pipeline for speech-controlled scenarios like HRI scenarios, the first step is to provide an ASR system to the pipeline. A common strategy is to employ a general-purpose ASR like Google's ASR (see Sections 3.4 and 3.6). The language models of general-purpose ASR system possess a large vocabulary, leading to a lower recognition performance. A large vocabulary is not necessary in most HRI tasks, as a robot is only to perform a limited number of actions. Also, we fully controlled vocabulary prevents the system from recognizing out-of-domain words that may cause difficulties to the NLP systems processing the text output of the ASR system. Another limitation of cloud-based ASR systems like Google's is the low response time due to a remote connection. For these reasons, we decided to train our own local ASR system. A difficulty is the lack of freely-available training data and

Listing 8 The following definitions contain the different parameters for the layers of the acoustic model of the *SemaPred Recognizer*.

```
s := sentence
n := length of sentence
e := embedding sequence
            word vectors from fastText
            size 300 * n
conv1 := 1D valid convolution with
            channels = 1,
            kernel sizes = 1
            stride = 1
            activation = ReLU
conv2-conv13 := 1D valid convolution with
            channels = 1,
            kernel sizes = 2,3,..., 13
            stride = 1
            activation = ReLU
max_pool := max pooling
            size = 5
flatten := remove dimensions
dropout1 := dropout regularization
            probability = ... (see Sec. 5.5)
concatenate := concatenate input to one vector
dropout2 := dropout regularization
            probability = ... (see Sec. 5.5)
dense := fully connected layer
            hidden size = ... (see Sec. 5.5)
            size = hidden size * 591
output := output layer
            size = 591 (encoding see this section)
            activation = sigmoid
optimizer := optimization function
            method = adam
            learning rate = ... (see Sec. 5.5)
            beta_1 = 0.9
            beta_2 = 0.999
            loss = binary crossentropy
```

Listing 9 The following pseudocode contains the data flow of the *SemaPred Recognizer*.

```
batch size = ... (see Sec. 5.5)

take batches

take s from batch

transform s to e

filter1 = dropout1(flatten(max_pool(conv1(e))))
filter2 = dropout1(flatten(max_pool(conv2(e))))
filter3 = dropout1(flatten(max_pool(conv3(e))))
...
filter13 = dropout1(flatten(max_pool(conv13(e))))

concatenated_filters =
      concatenate(filter1, filter2, ..., filter13)

output = dense(dropout2(concatenated_filters))
```

the vast amount of computational resources. We decided to adapt an existing ASR system by reducing the number of layers to be able to train the system in reasonable time.

The architecture of our model consists of a modified DeepSpeech 2 (Hannun et al., 2014a; Amodei et al., 2015) architecture and is a slightly different variant of the work of (Lakomkin et al., 2018). The original DeepSpeech 2 architecture contains three convolutional layers, while our system only has two. The convolutional layers are followed by 5 bidirectional GRU layers, while the original version uses 7. The GRU layers are followed by a fully connected layer. As the architecture is more light-weight than the original DeepSpeech 2 architecture, we refer to it as *SlimSpeech*.

As features, we extract power spectrograms from the audio sequence, each having a width of 20ms and a stride of 10ms. We standardize the features by subtracting the mean and dividing by the standard deviation, which results in a zero-mean and unit variance. For the first convolutional layer, the kernel size is 41*11, while having 32 filters. A padding of 10 steps is applied at the beginning and the end of the sequence. The first convolutional layer is followed by a *Batch Normalization* layer, and *hard tanh* is chosen as an activation function. The second convolutional layer has a filter size of 21*11, also containing 32 filters. Here, we also apply *Batch Normalization* and *hard tanh*[2]. The output is fed to the first bidirectional GRU layer. The output is *batch-normalized* and fed to the next bidirectional GRU layer. This way, five GRU layers are connected, each possessing a hidden layer with a size of 1024 units. The last GRU layers are also *batch-normalized*; the output is fed to the fully connected layer. The output of the fully connected layer employs softmax. The model output is a character index representation (29 units) or phoneme index representation (41 units). We refer to these models as *SlimSpeech Chars* (SSC) and *SlimSpeech Phonemes* (SSP). The network is trained using Connectionist Temporal Classification (CTC) (Graves et al., 2006). The whole architecture is illustrated in Figure 4.6. All parameters needed to implement the system can be found in Listing 10.

4.4.1 Speech Training Data

To train the system, we used five different English ASR datasets: *LibriSpeech*, *TED-LIUM v2*, *VoxForge*, *Mozilla Common Voice*, and *Google Speech Commands*. As we welcome the system to be used by the open-source community, we only chose freely available datasets. In total, the training data contains 1500 hours of speech (see Table 4.1), data from more than 4000 speakers, and 646,222 utterances. All the audio data was converted to single-channel, a frame-rate of 16 bit, and a sampling rate of 16kHz. We removed utterances

[2] *hard tanh* is a faster but inaccurate *tanh* and used as an activation function

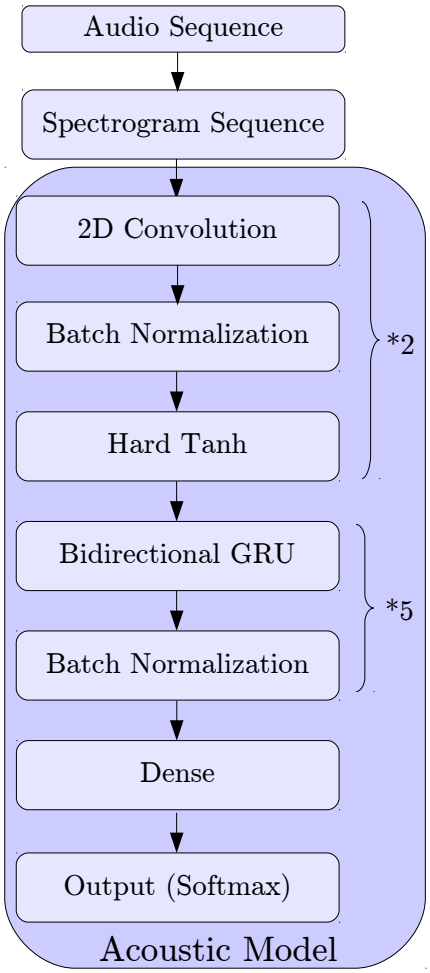


Figure 4.6: The architecture of the acoustic model of our self-trained *Slim-Speech* ASR system. The outputs of the system are character distributions per time step. These distributions can be transformed into characters using a decoder. We also created a phoneme-based variant that produces phoneme distributions.

longer than 15 seconds to overcome GPU memory limitations. The output labels consist of text strings containing 26 different characters. To train the phoneme variant, the text is transformed into phoneme sequences using a trainable grapheme-to-phoneme converter Bisani and Ney (2008) trained on CMUdict. The data consists of phoneme sequences, containing a set of 41 ARPABET phonemes.

Listing 10 The following definitions contain the different parameters for the layers of the acoustic model of *SlimSpeech*

```
a := audio sequence 16,000 kHz
n := length of the input sequence (variable)
s := spectrogram sequence
            shape: 161 * n
            frame_size = 20ms
            window shift = 10ms
conv1 := 2D same convolution with
            channels = 1,
            filters = 32,
            kernel size = 41 * 21,
            stride = (2,2)
batch_norm := batch normalization
            epsilon = 1e-05
            momentum = 0.1
conv2 := 2D same convolution with
            channels = 32,
            filters = 32,
            kernel size = 21 * 21,
            stride = (2,1)
gru1 := bidirectional gru
            input size = 672
            hidden size = 1024
gru2-gru5 := bidirectional gru
            input size = 672
            hidden size = 1024
dense := fully connected layer
            size = 1024 * 29
output := output nodes
            size 29 * n
            (29 = number of chars; 41 for phonemes)
loss := connectionist temporal classification
```

Table 4.1: ASR datasets used for training.

Dataset	Hours of Speech
LibriSpeech (Panayotov et al., 2015)	1,000
TED-LIUM v2 (Rousseau et al., 2014)	200
VoxForge[3]	100
Mozilla Common Voice[4]	300
Google Speech Commands (Buchner, 2017)	∼40
Total	∼1,640
Total Used	∼1,500

LibriSpeech

LibriSpeech contains read English coming from audiobooks. Examples:

- `chapter eleven the morrow brought a very sober looking morning the sun making only a few efforts to appear and catherine augured from it everything most favourable to her wishes`
- `declined giving any absolute promise of sunshine she applied to missus allen and missus allen's opinion was more positive she had no doubt in the world of its being a very fine day if the clouds would only go off and the sun keep out`
- `at about eleven o'clock however a few specks of small rain upon the windows caught catherine's watchful eye and oh dear i do believe it will be wet broke from her in a most desponding tone`

[3]http://www.voxforge.org
[4]http://voice.mozilla.org

TED-LIUM v2

TED-LIUM v2 consists of TED talks, which were transcribed afterwards. Examples:

- `i'm here today to show my photographs of the lakota many of you may have heard of the lakota or at least the larger group of tribes called`
- `the lakota are one of many tribes that were moved off their land to prisoner of war camps now called reservations the pine ridge reservation`
- `now if any of you have ever heard of aim the american indian movement or of russell means or leonard peltier or of the stand off at oglala`

VoxForge

VoxForge is a dataset containing crowd-sourced speech. It also contains other language; we chose only the English subset. Examples:

- `it seemed the ordained order of things that dogs should work`
- `and that was the last of francois and perrault`
- `mercedes screamed cried laughed and manifested the chaotic abandonment of hysteria`

Mozilla Common Voice

Mozilla Common Voice is a dataset containing crowd-sourced speech. Examples:

- `did you actually give tamara that gun`
- `good luck to you`
- `now it will be with hope`

Google Speech Commands

Google Speech Commands is a dataset containing crowd-sourced speech. It contains 35 different words, each sample has a duration of around one second and contains one word. Examples:

- `yes, no, up, down, left, right, on, off, stop, go`

4.4.2 Training details

We apply data augmentation by randomly adjusting the tempo and loudness of the audio samples. For the first epoch, the utterances are sorted using *SortaGrad* (Amodei et al., 2015), which intends to improve the convergence of the model. We employ stochastic gradient descent with a learning rate of 0.0001 to train the model. After each epoch, the learning rate is reduced by dividing by the factor 1.01. As a validation set, we chose the *LibriSpeech* validation set. The model was trained until it got saturated regarding the word error rate (WER) on the validation set. For the phoneme version of the model, the validation set was transformed to phonemes, and the phoneme error rate (PER) was measured.

4.4.3 Decoder

In test mode, we use two different decoder types, the *Greedy Decoder*, and the *Beam Search Decoder*. The *Greedy Decoder* performs best path decoding. The output of the network per timestep is a vector containing the character or phoneme distribution per timestep. It merely takes the $argmax$ function over the character or phoneme distribution per timestep. If the found output contains the "blank" character, the character is removed. The output of the CTC is a label per timestep. If a character or phoneme takes longer than one timestep, the $argmax$ function will find that character or phoneme in multiple timesteps in a row. These duplicates are removed. The algorithm is fast but may commit errors, in particular since information from a language model is not considered.

The other decoder variant is the *Beam Search Decoder* (Hannun et al., 2014b). The idea is to keep multiple different paths through the timestep-character trellis. It is a variant of Best-first Search, which sorts all paths using a heuristic reflecting the quality of the path. In this case, the heuristic is the score of a path. The principle of the *Beam Search Decoder* is described in Section 2.5. It uses language information coming from N-gram language models, in our case, bigram, trigram, or quadrogram language models.

4.5 External Language Model: Sentence Templates

A commonly-used variant of language models in ASR systems are N-gram-based statistical language models. These models are able to correct local errors within a text hypothesis, but lack a global context about the whole sentence. Grammar-based language models may provide this global context, but they need to be hand-crafted for the application. Our idea is to combine the

principles of N-gram-based and grammar-based language models and present this approach in this section. The concept is learning the grammar from the data. It is possible to use this grammar as a language model or to integrate parts of the N-gram concept in a second step. The developed approaches are considered as external language models, because they do not perform decoding of acoustic models; instead they process the text output of a general-purpose ASR system. This work is inspired by the work of Hinaut and Dominey (2013), who proposed a neural computational model for thematic role labeling from incoming grammatical constructions (Goldberg, 1995). Our approach does not employ neural networks.

One hypothesis for language acquisition is that children learn "templates of sentences" (Tomasello, 2009) (e.g., grammatical constructions (Goldberg, 1995)). The θRARes model proposed by Hinaut and Dominey (2013) learns to assign θ-roles (thematic roles) to the semantic words (SW) or content words in a sentence. E.g., the sentence:

```
put the pyramid on the cube
```

is mapped to the predicate: `put(pyramid, cube)`. This mapping is called a grammatical construction (Tomasello, 2009). The sentence is preprocessed by removing all semantic words from the sequence and replacing them by the wildcard token `X`:

```
X the X on the X
```

In this work, we call these structures like a *Sentence Template* (ST). These STs are used for our language model. The system learns to assign the slots of the predicate to the SWs of the sequence and requires to determine if a word is an SW and has to be replaced by the wildcard token. As SWs are an open class, which means that new words can be added to this class, it is not trivial to determine if a word is an SW or not. Instead, non-semantic words, which are function words, are identified, as they belong to a closed

class, meaning that there is only a finite number of them. All words that do not belong to the closed class of function words are considered to be SWs. As function words belong to a closed class, they are known beforehand even for new domains.

The idea of our work is to be able to perform domain-restricted language modeling while also making use of general-purpose language models. The text output coming from general-purpose ASR is corrected and not completely overridden unlike in previous approaches (see Sec. 3.5, DOCKS *SentenceList*) where only the phonemic representation was processed. The ST approach also handles the syntactic structure. We consider function words to be domain-independent. As STs only consist of function words, we consider STs to be domain-independent. Instead of only processing the phonemic representation of a hypothesis, we are now able to exploit the benefits of a general-purpose language model.

Figure 4.7 depicts the processing pipeline of our system. We take the word sequence coming from a general-purpose ASR system and build the ST out of it. The training data for our restricted domain consists of grammatically and syntactically correct sentences. As the hypothesis coming from the general-purpose ASR system may be grammatically or syntactically incorrect, we try to find the closest sentence producible from the training data. To be able to generate variations that are not covered by the training data, we do not match the concrete sentence (like in Sec. 3.5), but only its ST against the ST of the training sentences using the normalized Levenshtein distance mentioned in Section 3.5. This produces a ranked list of STs closest to the one that was generated by Google's ASR.

Like Hinaut and Dominey (2013), we interpret the SWs of the training data as terminal words (non-replaceable words) in a grammar, and the wildcard slots as the non-terminals to be replaced. We call these sets of possible SWs *Terminal Bags* (e.g. $t0$, $t1$ below). This way, the model is able to generate variations of sentences for the same ST that may not be contained literally in

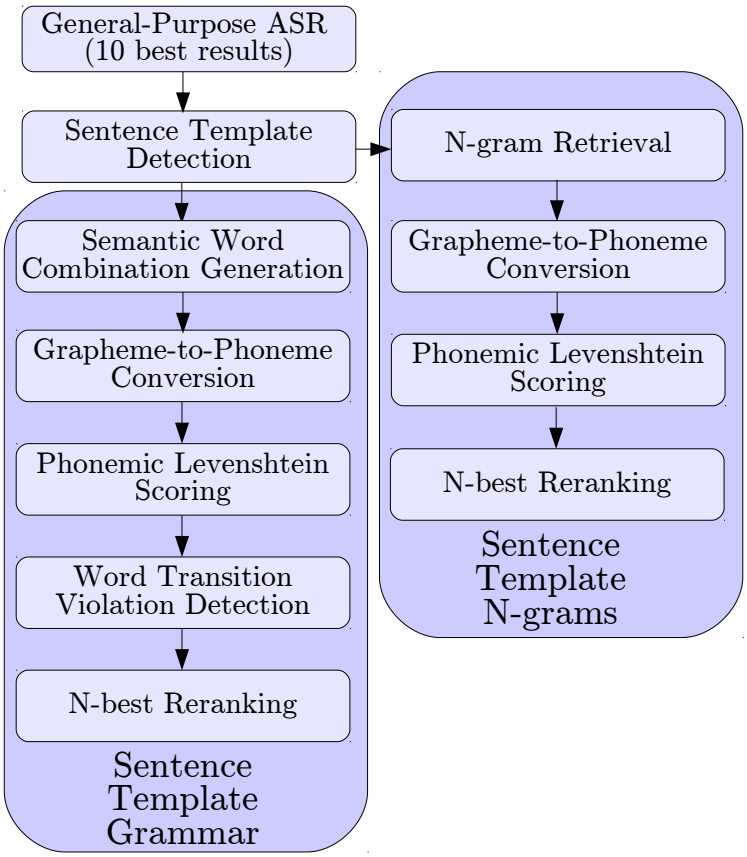


Figure 4.7: This figure shows the processing pipeline of our postprocessing system. First, the system tries to use the *Sentence Template Grammar*. If the confidence for individual words or subsequences is below a threshold, this part of the hypothesis is postprocessed by the *Sentence Template N-gram* module.

the training data. The following example explains the principle. The training data consists of the sentences:

- `put the pyramid on the cube`
- `move the prism on the block`
- `move the prism to the left`

The ST for the first two sentences is `X the X on the X` and `X the X to the X` for the third one. The training is performed by generating a grammar:

- ⟨s0⟩ = ⟨t0⟩ the ⟨t1⟩ on the ⟨t2⟩
- ⟨s1⟩ = ⟨t3⟩ the ⟨t4⟩ to the ⟨t5⟩
- ⟨t0⟩ = put | move
- ⟨t1⟩ = pyramid | prism
- ⟨t2⟩ = cube | block
- ⟨t3⟩ = move
- ⟨t4⟩ = prism
- ⟨t5⟩ = left

This grammar is able to generate, e.g., the sentence put the prism on the cube, which is not present in the training data. Testing a speech utterance is performed by generating a hypothesis using, e.g., Google's ASR. In this example, where the reference text is put the prism on the cube, Google may produce the hypothesis pull the pistol on the cube. This hypothesis is transformed to its ST, which is X the X on the X. We employ the normalized Levenshtein distance (see eq. 3.3) to calculate the best-matching ST from the training data, which is, in this case, also X the X on the X. Then, the SWs are matched against each other to find the best-matching word from the *Terminal Bag*. This means that we calculate the normalized Levenshtein distance of pull against put and pull against move on phoneme level for the first SW gap. We do this for all SW gaps and get the most probable sequence put the prism on the cube.

In some cases, the hypothesis coming from the general-purpose ASR system cannot be converted to a correct ST, e.g. put **them** prism on the cube. In this case, we calculate the best matching ST and generate possible combinations for the incorrect part of the sequence, e.g. put the pyramid, put the prism, move the prism, move the pyramid. This guarantees

a correct ST, which can be interpreted by a θ-role assignment model (e.g. Hinaut and Dominey (2013)).

As the normalized Levenshtein distance can be used as a confidence value for each word, words that the system is uncertain about can be identified. This information can be used to repair a hypothesis in the second step in case some of the words possess a low confidence value (threshold 0.5).

Listing 11 The following pseudocode contains the training process of the *Sentence Template* external language models.

```
for sentence in training_set
      create sentence_template for sentence
      if not sentence_template stored yet
            store sentence_template
            save non_terminal_words in terminal_bags
            link terminal_bags with \
                  slots of sentence_template
      else
            add non_terminal_words to terminal_bags \
                  of existing sentence_template

      for terminal_word in sentence_template
            for terminal_word2 in sentence_template
                  store all words between terminal_word \
                                    and terminal_word2
```

For long and nested sentences, the correct hypothesis cannot be constructed due to the lack of training data, meaning not enough samples for a specific ST and a lack of possible SWs. This means, the *Terminal Bags* do not contain many words and possibly not the word that was uttered. We use a threshold for each word and repair all words that were recognized with a confidence lower than this threshold (0.5). For this, we train the system by collecting all n-grams inside the training data up to 10-grams. For example, if we have the input sentence `put the yellow prism on the cube`, we possibly could only generate `put the blue prism on the cube`, if the word `yellow` was never used in this SW gap. The function words `the` and `on` can

be used as "anchors" to retrieve n-grams starting with `the` and ending with `on`. Then, another scoring process is started using the normalized Levenshtein distance to calculate the best-matching n-gram from the training data. This way, we can generalize from other sentences and STs in the training data.

Listing 12 The following pseudocode contains the test process of the *Sentence Template* external language models.

```
create sentence_template for input_sentence
find best matching stored sentence_template using \
                                        Levenshtein distance
for all terminal_bags of sentence_template
      for non_terminal_word of terminal_bag
            score input word of slot against \
                  terminal_word using \
                  Phonemic Levenshtein Scoring
            take best matching non_terminal_word
            fill slot of sentence_template with \
                  best matching non_terminal_word
            save confidence of non_terminal_word
output = filled sentence_template


%for Sentence Template N-grams model only:
find words where confidence < 0.5 in output
for all connected words
      find anchor terminal_word before and after
      rescore connected words against stored \
            (n-1)-grams, n-grams and (n+1)-grams
      take best matching gram
      replace words of output
```

To reduce the number of incorrect sentences, we check each generated hypothesis for non-existing word transitions (bigrams) in the training data, and, if occurring, we can drop that hypothesis. Also, we perform the postprocessing for the 10 best hypotheses coming from the general-purpose ASR system, and afterwards sort our generated hypotheses by average word confidence.

The approach is expected to work well, when there are not to many variations regarding the sentence structure. The more variations, the more examples are needed to fill the *Terminal Bags*. For the n-gram-based approach, it is sufficient to cover only the structure, the vocabulary may be found within the stored n-grams.

The original ST language model was presented by Twiefel et al. (2017) and used as a postprocessor for Google's Search by Voice. In this work, we connect the system to our self-trained ASR system. We extended the model to be able to also process phonemes instead of words directly. This way, the model can also be connected to one of our ASR variants that produce phonemes instead of characters (*SlimSpeech Phonemes*). The algorithm is shown as pseudocode in Listing 11 and 12.

4.6 Semantic Logic Predicate Interpreter

SemaPred representations are novel and simple descriptors of natural language. A requirement for *SemaPreds* to be considered as useful is the ability to be able to process *SemaPreds* in concrete scenarios. In this work, we present a *SemaPred Interpreter* that is able to process *SemaPreds* and extract concrete execution parameters. A set of *SemaPreds* is usually valid or invalid, depending on its context. One application is the movement of entities in a Blocks World, but also other tasks are possible. In this case, the *SemaPreds* contain the information about which entity has to be moved to which position in a scene. The entities can be differentiated using their shape (cube, pyramid). The input of the model is a set of *SemaPreds* describing the command, the outputs are the source and destination coordinates for a movement action.

The ASP code snippets in this section are (partially) taken from Tobergte's Bacherlor's thesis (Tobergte, 2017). We developed the ideas and conceptions behind this approach, while Tobergte performed the implementation (under our supervision).

4.6.1 Logic Modeling

The *SemaPred Interpreter* is based on declarative logic. We employ ASP to describe the scene and the *SemaPreds*. The following section will describe how to define the scene for a *SemaPred* scenario. As an example, we use a Blocks World, for other scenarios or tasks, the process is equivalent. First, the world has to be described. If we have a 8*8*8 grid world:

Listing 13 Definition of the grid world

```
1 size(8).
2 grid(X,Y,Z) :-
3       X = 0..S-1,
4       Y = 0..S-1,
5       Z = 0..S-1,
6       size(S).
```

Next, we define the so-called *references* which refer to positions in the grid (Listing 14).

Listing 14 References in the grid

```
1 % points on the floor
2 ref(X,Y,Z) :- grid(X,Y,Z), Z=0.
3 % points with shapes in them
4 ref(X,Y,Z) :- shape(X,Y,Z).
5 % points that are directly above shapes
6 ref(X,Y,Z+1) :- shape(X,Y,Z).
7 % point with gripper in it
8 ref(X,Y,Z) :- gripper(X,Y,Z).
```

Afterwards, the board has to be defined. It is below the grid, which is indicated by a -1 as Z coordinate. Also, we define the position of the robot itself:

Listing 15 Definition of the board and the robot

```
1 board(X,Y,-1) :- grid(X,Y,Z), Z = 0.
2 robot(X,Y,Z) :-
3       X = -(S/2),
4       Y = S/2,
5       Z = 0,
6       size(S).
```

The board consists of edges and corners, which can be used as reference points. Edges contain the tiles at the edge of the grid, a corner contains the tile directly in the corner of the grid. We group edges and corners in a class called rim:

Listing 16 Definition of the edges and corners

```
1  % there are 4 edges, each having 8 coordinates
2  % back edge (near robot)
3  edge(X,Y,Z) :- X = 0, Y = 0..S-1, Z = -1, size(S).
4  % right edge (from robot perspective)
5  edge(X,Y,Z) :- X = 0..S-1, Y = 0, Z = -1, size(S).
6  % front edge (far from robot)
7  edge(X,Y,Z) :- X = S-1, Y = 0..S-1, Z = -1, size(S).
8  % left edge (from robot perspective)
9  edge(X,Y,Z) :- X = 0..S-1, Y = S-1, Z = -1, size(S).
10 % the 4 corners
11 corner(0,0,-1 ; 0,7,-1 ; 7,0,-1 ; 7,7,-1).
12 %rim superclass
13 rim(X,Y,Z) :- corner(X,Y,Z).
14 rim(X,Y,Z) :- edge(X,Y,Z).
```

The board consists of different regions, the front, back, right, left, and center regions. They are defined using the X and Y coordinates of the board, while the Z coordinate is not relevant. It has to be set to be below the board, as entities are positioned on top of it. We chose -3 as the Z coordinate to separate it from other descriptions and parts of the board. The sizes of the regions can be calculated dynamically for differently sized boards. For the center region, we define several tiles, as the size of the board may be even, and the center region consists of 4 tiles.

Listing 17 Definition of regions

```
% right region
region(X,Y,-3) :- X= (S/2),Y = 0, size(S).
% left region
region(X,Y,-3) :- X= (S/2), Y = S-1, size(S).
% front region
region(X,Y,-3) :- X= S-1,Y = (S/2), size(S).
% back region (near robot)
region(X,Y,-3) :- X= 0, Y = (S/2), size(S).
% center region
region(X,Y,-3) :- X=(S/2), Y=(S/2), size(S).
region(X,Y,-3) :- X=(S/2)-1,Y= (S/2), size(S).
region(X,Y,-3) :- X=(S/2), Y=(S/2)-1, size(S).
region(X,Y,-3) :- X=(S/2)-1,Y= (S/2)-1, size(S).
```

The objects manipulated in this scenario are called shapes, which take one tile in the grid. They may contain cubes or prisms (Listing 18).

Listing 18 Definition of shapes

```
shape(X,Y,Z) :- cube(X,Y,Z).
shape(X,Y,Z) :- prism(X,Y,Z).
```

The shapes may have different attributes like a color, a position, or may be individual meaning it is not nearby other shapes. Attributes are defined by the `has_attribute` rule.

Listing 19 Definition of attributes

```
has_attribute(X,Y,Z, ATTR) :- has_color(shape(X,Y,Z),
                              color(ATTR)).
has_attribute(X,Y,Z, ATTR) :- has_position(X,Y,Z,
                              position(ATTR)).
has_attribute(X,Y,Z, individual) :- individual(X,Y,Z).
```

The definitions of the attributes are trivial and only mentioned in the appendix. There are different directionals, which are left, right, back, front, and

center. The following definition explains the left relation. It describes an entity to be on the left side of the board or to be on the leftmost location for edges and regions:

Listing 20 Definition of left

```
1 left(X,Y,Z) :-
2 pointer(X,Y,Z), size(S), Y >= (S/2),
3 #false: edge(X,Y,Z).
4 % A region / edge is (on the) left if it is located
5 % on the leftmost location on board, given the
6 % boardsize S.
7 left(X,Y,Z) :- edge(X,Y,Z), size(S), Y = S-1.
8 left(X,Y,Z) :- region(X,Y,Z), size(S), Y = S-1.
```

Again, for the center definition the description is a special case:

Listing 21 Definition of center

```
1 % center region is defined to match 4 region points
2 % in the center
3 center(X,Y,Z) :- region(X,Y,Z), X <= (S/2),
4             X >= (S/2)-1, Y <= (S/2),
5             Y >= (S/2)-1, size(S).
```

The entities in the scenario may be in different relations towards each other. The most important one is the `above` relation. It indicates that one entity is above another entity but on the same X and Y coordinate. There are two definitions of the `above` relation, one for other entities, the other one for rims.

For the different regions, the definition of above is similar, with a special case for the center region (Listing 23).

The shapes in our scenario can be moved into different direction like forward, backward, rightwards, and leftwards. When moving an object into a direction, a distance parameter has to be set to indicate how many steps it is moved. We need to define each direction twice, once for references and once for rims (Listing 24).

Listing 22 Definition of above

```
% Ref X,Y,Z is above A,B,C, if the altitude is
% higher, and they are on the same coordinates
% otherwise.
above(ref(X,Y,Z),ref(A,B,C)) :- ref(X,Y,Z),
                        ref(A,B,C), X = A,
                        Y = B, Z > C.
% Special case: rim (corner+edge) has no actual
% "level", so a point on the ground is above it
% if X and Y coordinates match.
above(ref(X,Y,Z), ref(A,B,C)) :- ref(X,Y,Z),
                        rim(A,B,C),
                        X = A, Y = B.
```

Listing 23 Definition of above

```
% Above rule for regions, center is special
above(ref(X,Y,Z), ref(A,B,C)) :- region(A,B,C),
                        shape(X,Y,Z),
            horizontaldistance(X,Y,Z,A,B,C,Dist),
                        Dist <= 0,
                        center(A,B,C).
above(ref(X,Y,Z), ref(A,B,C)) :-
            right(X,Y,Z), region(A,B,C), right(A,B,C).
above(ref(X,Y,Z), ref(A,B,C)) :-
            left(X,Y,Z), region(A,B,C), left(A,B,C).
above(ref(X,Y,Z), ref(A,B,C)) :-
            front(X,Y,Z), region(A,B,C), front(A,B,C).
above(ref(X,Y,Z), ref(A,B,C)) :-
            back(X,Y,Z), region(A,B,C), back(A,B,C).
```

There are three different actions, namely `take`, `drop`, and `move`. The `take` action requires an empty gripper before the execution, and afterwards, a gripper holding an entity. The `drop` action can be interpreted in different ways. If the gripper is not holding an entity, it implies that it first has to grab an entity and then drop it at a specific location or directly below the gripper if a destination is not provided. The `move` action always consists of taking an

Listing 24 Definition of forward

```
% as ref and rim are mutually exclusive, two rules
% are needed
forward(ref(X,Y,Z), ref(A,B,C), Dist) :- ref(X,Y,Z),
                                         ref(A,B,C),
                                         Y = B,
                                         X-Dist = A,
                                         Dist = 1..S-1,
                                         size(S).
forward(ref(X,Y,Z), ref(A,B,C), Dist) :- ref(X,Y,Z),
                                         rim(A,B,C),
                                         Y = B,
                                         X-Dist = A,
                                         Dist = 1..S-1,
                                         size(S).
```

entity and dropping it at a destination. The following descriptions define the actions. They use different auxiliary rules which are self-explaining and not shown here, but can be found in the appendix.

Listing 25 Definition of take

```
% normal take: from anywhere to gripper.
take(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
                                  ref(A,B,C),
                                  not_blocked(ref(X,Y,Z),
                                  free(ref(A,B,C)),
                                  gripper(A,B,C).
```

The `take` action determines the coordinates of the shape to be taken and the coordinates of the gripper holding the shapes afterwards. The shape must not be blocked, and the position of the gripper after the action needs to be free.

As the `drop` action may be interpreted in two different ways, it requires two different definitions. The first definition holds the description of a `drop`, which is preceded by a `take` action. It requires a non-blocked shape inside the gripper and a valid destination. Also, there could be the exception of

a shape being dropped on itself which needs to be prevented. The second definition describes the `drop` action without a preceding `take`, which is a `move` action. It does not need the shape to be inside the gripper. Again, it cannot be placed on itself.

Listing 26 Definition of drop

```
% normal drop: move from gripper onto free position.
drop(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
                        gripper(X,Y,Z),
                        not_blocked(ref(X,Y,Z)),
                        ref(A,B,C),
                      valid_pos_for_move(ref(A,B,C)),
                        shape_in_gripper(true),
    % but dropping the object on top
    % of itself is not possible:
    #false: X == A, Y == B, Z+1 == C.
% special drop: if no block in gripper, then
% drop should work with an object from anywhere
drop(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
                        ref(A,B,C),
                        not_blocked(ref(X,Y,Z)),
                        free(ref(A,B,C)),
                        shape_in_gripper(false),
    % but dropping the object ontop of itself is not
    % possible:
    #false: X == A, Y == B, Z+1 == C.
```

The `move` action has three cases: a shape can be moved onto the ground, into the gripper of on top of another shape. For the first case, the shape must not be blocked, and the space on the ground has to be free. The second case also determines the gripper coordinates and is valid for all positions, not only the ground. The third case handles the positioning on top of another shape. The shape cannot be placed on top of itself, and it cannot be put on a prism, as prisms are not able to hold other shapes.

Listing 27 Definition of move

```
% A shape X,Y,Z can be moved to a point A,B,C,
% if X,Y,Z is not blocked and A,B,C is free.
% A,B,C has to be on the ground,
move(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
                        ref(A,B,C),
                        not_blocked(ref(X,Y,Z)),
                        free(ref(A,B,C)),
                        C = 0.
% A shape can be moved into the gripper
move(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
                        ref(A,B,C),
                        not_blocked(ref(X,Y,Z)),
                        free(ref(A,B,C)),
                        gripper(A,B,C).
% or on top of another shape, but not on top of a
% prism the shape cannot be put on top of itself.
move(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
                        ref(A,B,C),
                        not_blocked(ref(X,Y,Z)),
                        free(ref(A,B,C)),
                    ontopof(ref(A,B,C), shape(P,Q,R)),
    #false: P == X, Q == Y, R == Z;
    #false: prism(P,Q,R).
```

4.6.2 Command Encoding

A command consists of a set of *SemaPreds* and a world layout. First, the world layout is used to ground the world. A world inside the data set consists of an XML representation holding the positions of shapes, their colors, and the position and state of the gripper. Listing 28 shows an example of a world layout.

Listing 28 XML encoding of a world layout.

```
<layout id="1">
      <gripper position="6 5 4" open="true" />
      <shapes>
            <cube color="blue" position="3 3 0" />
            <cube color="blue" position="3 3 1" />
            <cube color="red" position="6 5 0" />
            <prism color="green" position="6 5 1" />
      </shapes>
</layout>
```

The XML representation of Listing 28 is transformed to ASP facts:

Listing 29 A world layout (XML) transformed into an ASP representation.

```
1 gripper(6,5,4).
2 cube(3,3,0).
3 has_color(shape(3,3,0),color(blue).
4 cube(3,3,1).
5 has_color(shape(3,3,1),color(blue).
6 cube(6,5,0).
7 has_color(shape(6,5,0),color(red).
8 prism(6,5,1).
9 has_color(shape(6,5,1),color(green).
```

Then, the *SemaPreds* can be processed. For the given layout, the *SemaPreds* could be the following:

Listing 30 A command encoded as *SemaPreds* using comma-separated values.

```
19,0,move,prism_1,above,cube_2
19,1,green,prism_1,,
19,1,red,cube_2,,
```

The annotation is a comma-separated values (CSV) structure. The first column indicates the ID of the *SemaPreds*. Then, a descriptor, 0 or 1 is given, declaring if it is a `SemaPred` with four slots (actions and relations) or two slots (attributes). All entities possess an ID that is separated by an underscore. Now, the given *SemaPreds* can be parsed and transformed to ASP facts (see Listing 31). In this example, a `move` is performed, which is defined by moving a shape to an empty position. All entities are transformed into ASP facts (lines 1-2). Also, the destination of the shape has to be added (lines 3). Then, the attributes are parsed. In this case, there are only color attributes (lines 4-5). The attributes are linked to the entities via the coordinate variables: for example entity 1 has the same coordinates (A,B,C) for its shape (line 1) and its color attribute (line 4). The attributes are followed by the given relation (above). Again, the entities are linked via their coordinates (line 6). Afterwards, the action (move) is added. Its arguments are the shape to be moved and the unknown destination (line 7).

Listing 31 A list of *SemaPreds* encoded in ASP.

```
1 prism(A,B,C),
2 cube(D,E,F),
3 ref(G,H,I),
4 has_color(shape(A,B,C), color(green)),
5 has_color(shape(D,E,F), color(red)),
6 ontopof(ref(A,B,C), ref(D,E,F)),
7 move(shape(A,B,C),ref(G,H,I))
```

This list of facts is true if the command can be processed, which means that there is a valid and consistent allocation. To assure this, an integrity constraint is defined (see Listing 32, lines 1-7). The output of the system are the source

and target position of the shape to be moved. For this purpose, we define them using the same list of facts we generated. The `show` directive makes the solver display the variable assignments.

Listing 32 A list of *SemaPreds* encoded in ASP to derive source and destination coordinates for the robot arm.

```
1  :- #false: prism(A,B,C),
2        cube(D,E,F),
3        ref(G,H,I),
4        has_color(shape(A,B,C), color(green)),
5        has_color(shape(D,E,F), color(red)),
6        ontopof(ref(A,B,C), ref(D,E,F)),
7        move(shape(A,B,C),ref(G,H,I)).
8  source(A,B,C) :- ... %same facts as in line 1-7
9  #show source/3.
10 target(G,H,I) :- ... %same facts as in line 1-7
11 #show target/3.
```

4.7 Semantic Evaluator

The aforementioned approaches are arranged as a pipeline starting with a speech signal which is processed by an acoustic model, a language model, a natural language processor, and a logic interpreter. This pipeline is expected to work well as long as the quality of the output coming from each of its modules is acceptable. When acoustic noise comes into play, the acoustic model may create wrong hypotheses, which can only partially be corrected by the language model. These possibly incorrect text outputs are then processed by an NLP model that was trained on correct data. We expect that this task is hard to perform, and the NLP module benefits from assistance on the semantic level. For this purpose, we developed a *Semantic Evaluator* (SE) that evaluates if the predicates produced by the NLP module are meaningful in respect of the input.

The idea behind our approach is to provide a scoring mechanism for the produced hypotheses coming from our NLP module, which evaluates the quality of the output. An essential requirement is not only to score the output but also to provide a quality measure for its parts, making possible errors within the output identifiable. Usually, only parts of the output are expected to be incorrect, while others are correct. Following this idea, it would be possible to correct the incorrect parts by deriving them from the context.

The presented approach works on the *SemaPred* representations mentioned earlier. These representations are produced as output of our *SemaPred Recognizer* (SPR), and the aim of our approach is to measure the quality of these outputs. *SemaPreds* have a of a fixed structure containing words as units. To measure the quality, we need to define a semantic value for each word inside the *SemaPreds*. For this purpose, we chose the cosine similarity of word embeddings, which delivers high values for semantically similar words. The words need to be scored against the input of the SPI. When taking the following sentence and *SemaPreds*, it is obvious that the scoring cannot be performed in a trivial way:

`grab the green prism and put it on the blue brick that is on the red box`

Listing 33 Example SemaPreds

```
take, prism_1, EEE,  EEE
drop, prism_1, above, cube_2
is,   cube_2,  above, cube_3
green, prism_1
blue,  cube_2
red,   cube_3
```

To be able to score *SemaPreds* against a sentence, they both need to be represented as a sequence. We developed an algorithm that performs this step. First, we take all actions and add the attributes to the entities:

Listing 34 1st step of processing

```
take, green, prism_1, EEE, EEE
drop, green, prism_1, above, blue, cube_2
is, blue, cube_2, above, red, cube_3
```

Afterwards, we take all relations and search the actions for the first of its entities. Then, the second part (relation word and second entity) is appended. Also, we drop the word `is`:

Listing 35 2nd step of processing

```
take, green, prism_1, EEE, EEE
drop, green, prism_1, above, blue, cube_2, above, red,
    cube_3
```

In the final step, we omit all empty tokens and concatenate the action sequences. Also, we only keep the first occurrence of each entity and its attributes and relations:

Listing 36 3rd step of processing

```
take, green, prism_1, drop, above, blue, cube_2, above,
red, cube_3
```

For each of the words, we save the position inside the original *SemaPreds*. This way, we can give a score for each word in the *SemaPreds* while calculating the score of a sequence. For example, the saved position for `take` would be $action_1, word_1$, for `green` it would be $attribute_1, word_1$, for `prism_1` it would be $action_1, word_2$ etc.

Now, we have a sequential representation of the *SemaPreds*, which can be used to perform a sequence alignment against the input sentence. There are different requirements for the scoring algorithm. It needs to be able to align two differently-sized input sequences, and a user-defined scoring function. The *Needleman-Wunsch Algorithm* meets these requirements and is employed for our approach.

We define a scoring function that uses the cosine similarity as a positive score. For this purpose, the input sentence and the sequential *SemaPreds* are transformed into *fastText* representations. Then, the *Needleman-Wunsch Algorithm* is applied to calculate the best matching alignment of the two sequences using the cosine distance. For each word missing in one of the sequences, an empty token (`EEE`) is inserted while aligning. As *fastText* was trained on a massive amount of general-purpose data, and our approach is usually used inside a specific domain, the scoring function can be improved by adapting it to the domain.

The domain adaptation is performed by adding a domain-dependent semantic score. It is calculated using training data from the domain, in this case, 2,500 sentences and their *SemaPreds*. For all the sentences, we perform the described scoring process using only the cosine distance. Then, we calculate the frequency for one word being aligned to another. For example, if the `blue` is aligned to `cyan` three times and to `turquoise` twice, the semantic score would be 0.6 for `cyan` and 0.4 for `turquoise`. This mechanism is intended to stabilize the correct alignment. The score for each pair of words is stored and used when testing. The new score is calculated by:

$$\text{score}(a, b) = \frac{\text{cossim}(a, b) + \text{semscore}(a, b)}{2} \tag{4.1}$$

The best alignment for the given sentence and its *SemaPreds* could be:

Listing 37 Alignment of an input sentence and its *SemaPreds*.

```
grab the green prism   and  put it   on   the  blue brick
take EEE green prism_1 EEE drop EEE above EEE blue cube_2
0.7      1.0   1.0          0.8      0.7       1.0   0.7

that is  on     the red box
EEE  EEE above EEE red  cube_3
         0.7       1.0 0.9
```

For each of the words within the alignment, we calculated a score. This score can now be used as a confidence value for each word of the *SemaPreds* in Listing 33:

Listing 38 Scores for Example SemaPreds

```
0.7, 1.0,   ,
0.8, 1.0, 0.7, 0.7
  , 0.7, 0.7, 0.9
1.0, 1.0
1.0, 0.7
1.0, 0.9
```

These scores can be used as to calculate the confidence of each word. It can now be extended to identify probably incorrect parts of the recognized *SemaPreds*. The following example clarifies this. For example, the output of the ASR module was:

`grab the green prism and put it on the blue brick that is on the red box.`

The output of the SPR could be:

Listing 39 Example SemaPreds with wrong parts (bold).

```
take, cube_1,  EEE,  EEE
drop, cube_1, above, cube_2
is,  cube_2,  above, cube_3
white, cube_1
blue, cube_2
cyan,  cube_3
```

The word `prism` coming from the input was interpreted as `cube_1` by the SPR. Also, the word `green` was mapped to `white` and `red` to `cyan`. Again, the *SemaPreds* are transformed into a sequence and aligned against the input. The algorithm generates the confidence scores for each word inside the *SemaPreds*:

Listing 40 Scores for example *SemaPreds*.

```
0.7, 0.1,   ,
0.8, 0.1, 0.7, 0.7
  , 0.7, 0.7, 0.9
0.2, 0.1
1.0, 0.7
0.3, 0.1
```

These confidence values are now suitable to identify probably incorrect parts of the output. The whole architecture of the *Semantic Evaluator* is depicted in Figure 4.8. In summary, the module performs the following steps: Transform the *SemaPreds* to a sequence, convert this sequence and the input sentence to *fasttext* sequences, align the sequences using the *Needleman-Wunsch Algorithm*, use the scores as confidence for words inside the *SemaPreds*.

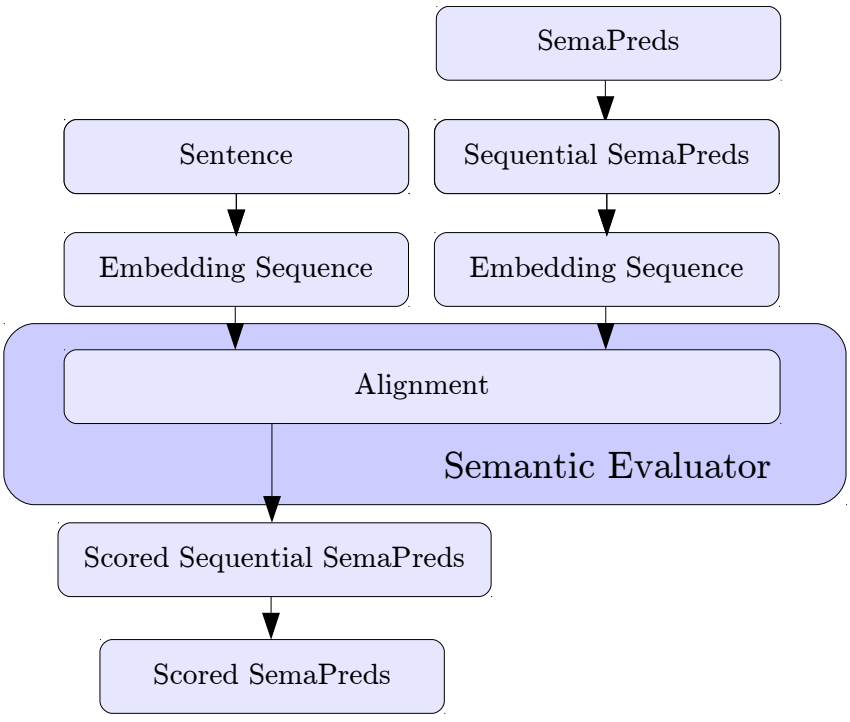


Figure 4.8: Architecture of the Semantic Evaluator.

4.8 Crossmodal Corrector

The described *Semantic Evaluator* in Section 4.7 is able to generate confidence values for each of the words inside a list of *SemaPreds*. This information can now be used to identify and correct incorrect parts of the *SemaPreds*. First, a text hypothesis is generated by the ASR module. Then, the *SemaPred Recognizer* produces *SemaPreds*, which are then processed by the *SemaPred Interpreter* described in Section 4.6. As mentioned, the outputs of the module are 3D source and destination coordinates. Now, there are different error cases like a wrong formulation of the utterance by the user, or the misunderstanding of the ASR module or the *SemaPred Recognizer*. For example, users often confuse intrinsic and extrinsic perspectives (Twiefel et al., 2016a) and confuse, for example, left and right, or forward and backward. It is also expected that incorrect text coming from the ASR increases the chance of an incorrect output of the *SemaPred Recognizer*. The idea of the *Crossmodal Corrector* is to employ another modality to validate and correct the produced output. In this scenario, the output are 3D coordinates but may be different for other scenarios.

If the *SemaPred Interpreter* does not produce exactly one source position and one destination position, we expect that one of the mentioned problems occurred. Especially the case that no source or destination is found may be challenging to handle. The first step is to analyse which part of the produced output may be wrong. As this is not possible for a non-existent solution (no coordinates), the error needs to be searched at a deeper layer. The *SemaPreds* which were processed by the *SemaPred Interpreter* were incorrect or not consistent with the scene. As the scene information is correct, the error results of implausible *SemaPreds*. They may be too restrictive or may contain word confusions. To identify the parts leading to a non-executable command, we use the idea of semantic plausibility. If a part of a *SemaPred* is not plausible regarding the context, we expect a high chance of it being the cause of the error. We employ the *Semantic Evaluator* to generate confidence values for

each word of the *SemaPreds*. Afterwards, these confidence scores are used as semantic plausibility values. Values below a threshold (we chose 0.2) are considered to be possibly implausible. Now, that the potential causes of the errors are identified, they can be treated as variables which need a better assignment.

To be able to find these assignments, we could use trial and error by setting them to all possible values and feeding the new *SemaPreds* to the *SemaPred Interpreter*. We found a better way by changing the logic declaration within the *SemaPred Interpreter*. For this purpose, we introduce the concept of wildcards by declaring potentially incorrect slots as wildcard slots. A wildcard slot does not have a variable assignment. Instead, the ASP interpreter sets the assignment while finding a solution. For example, there is the scene shown in Figure 4.9.

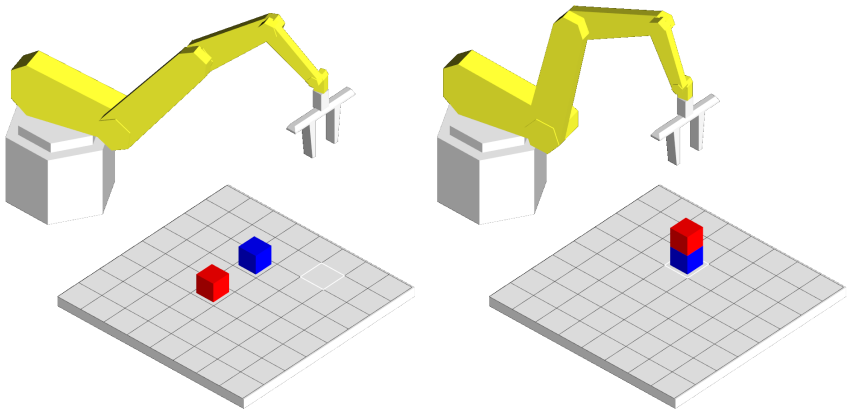


Figure 4.9: Example Scene.

The command could be: `move the` **`foo`** `box on top of the blue box` with the following *SemaPreds*; the *SemaPred Recognizer* recognized `foo` as `yellow` in this case:

Listing 41 Example SemaPreds with wrong parts (bold)

```
move,   cube_1,   above,   cube_2
yellow, cube_2,          ,
blue,   cube_2,          ,
```

The *SemaPred Interpreter* encodes the *SemaPreds* in ASP using the following description:

Listing 42 List of ASP facts encoding the command mentioned in Listing 41, containing the wrongly recognized word `yellow`

```
1 cube(A,B,C),
2 cube(D,E,F),
3 ref(G,H,I),
4 has_color(shape(A,B,C), color(yellow)),
5 has_color(shape(D,E,F), color(blue)),
6 ontopof(ref(A,B,C), ref(D,E,F)),
7 move(shape(A,B,C),ref(G,H,I))
```

The *Semantic Evaluator* scores the slot containing *foo* with a low confidence and thus identifying the words `foo` and `yellow` to be potentially implausible. This information can now be used to mark the slot of the word `yellow` as a wildcard slot. This is done by not allocating it with *yellow* but with a variable, for example `Wildcard_1`:

Listing 43 List of ASP facts encoding the command mentioned in Listing 41 including a wildcard

```
1 cube(A,B,C),
2 cube(D,E,F),
3 ref(G,H,I),
4 has_color(shape(A,B,C), color(Wildcard_1)),
5 has_color(shape(D,E,F), color(blue)),
6 ontopof(ref(A,B,C), ref(D,E,F)),
7 move(shape(A,B,C),ref(G,H,I))
```

This is done for all wildcards. The command `#show` can now be used to return the wildcard assignments. In this case, it would find `red`, as this is this only possible (see Figure 4.9). It is possible that multiple assignment are found. If this is the case, the assignments can be inserted inside the *SemaPreds*, which then can be rescored using the *Semantic Evaluator*, and the most probable one can be chosen. The whole architecture is depicted in Figure 4.10.

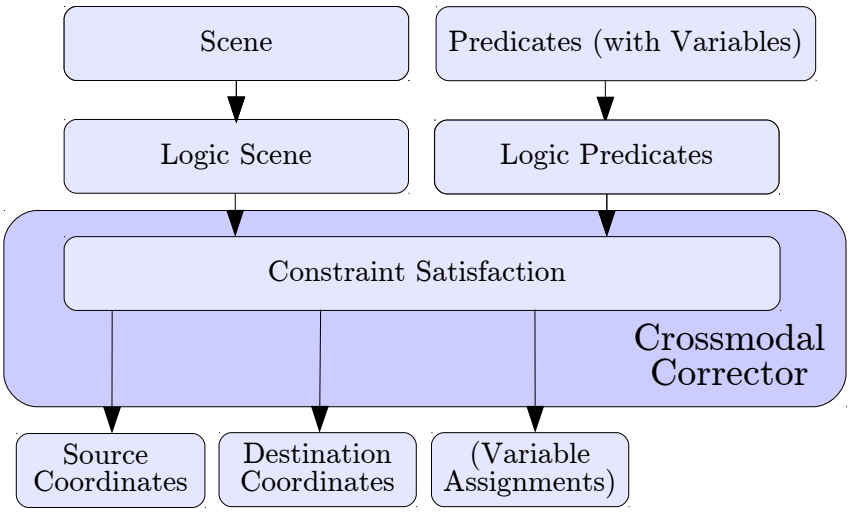


Figure 4.10: Architecture of the *Crossmodal Corrector*.

If, after this process, no solution is found, the user can be informed using a dialog. The same is possible for multiple solutions. We developed a simple dialog that produces natural language which can be displayed or read to the user. This is especially useful if the error was caused by the user. Regarding

the scene in Figure 4.9, the user could utter the command `take the cube`. As there are no further descriptions about the cube, there are two possible sources. In this case, the system will answer: `I found two cubes`.

It is also possible that the command is overspecified, which means, that the description is too restrictive. In this case, the ASR did not fail, and the *Semantic Evaluator* will not find any low scores. An approach to tackle this issue is to remove the *SemaPreds* that restrict the command too much, leading to no solution. We expect this to be the case for, especially, *Attribute SemaPreds*. If no solution is found using the *SemaPred Interpreter* and the *Crossmodal Corrector*, we focus on a new approach we call *Iterative Restriction*. We remove all attributes and only keep the actions and relations. Then, we iteratively restrict the command by readding and revalidating the *Attribute SemaPreds*. There are other strategies like *Constraint Relaxation* to address this problem which were not chosen but could potentially work better. For example, there is a scene like the one depicted in Figure 4.11.

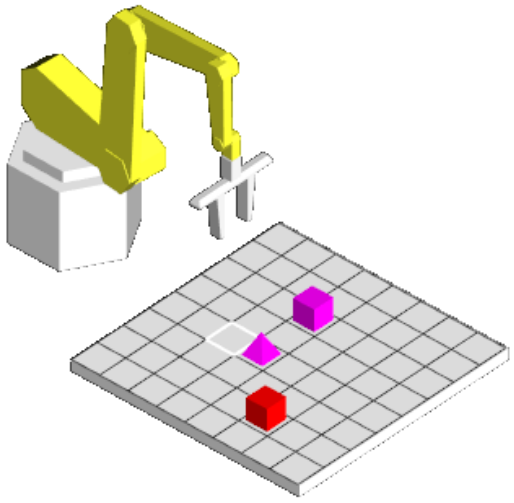


Figure 4.11: Example scene.

The produced partially incorrect *SemaPreds* could be:

Listing 44 Example SemaPreds with wrong parts (bold)

```
move,    prism_1,   above,  cube_2
blue,    prism_1,        ,
magenta, cube_2,         ,
```

In this case, the *SemaPred Interpreter* is not able to determine the source and destination position. Now, all attributes are removed, resulting in:

```
move, prism_1, above, cube_2
```

When reprocessing the *SemaPred*, the interpreter will find one source position (the prism) and two destinations (the two cubes). Still, there is no unique solution. We start to restrict the command iteratively by readding the attributes. When adding the first attribute (line 2), there will be no source again, as there are no blue prisms. In conclusion, we completely remove the first attribute from the stack. Then, we add the next attribute (line 3). The *SemaPreds* are reprocessed again, finding exactly one solution, namely the prism being the source and the magenta cube being the destination. This way, wrong *SemaPreds* can be corrected even if they possess a high semantic score.

4.9 Motion Simulator and Real Robot Application

To test the developed system in an application, we developed a web-based simulator and a robot application. The simulator is using Django[6] as a backend. The frontend is programmed using Three.js [7] The surface is displayed in Figure 4.12. It is possible to load scenes from the dataset, move the gripper

[6]https://www.djangoproject.com/
[7]https://threejs.org/

and open and close it. The user can also move around the objects and arrange new scenarios.

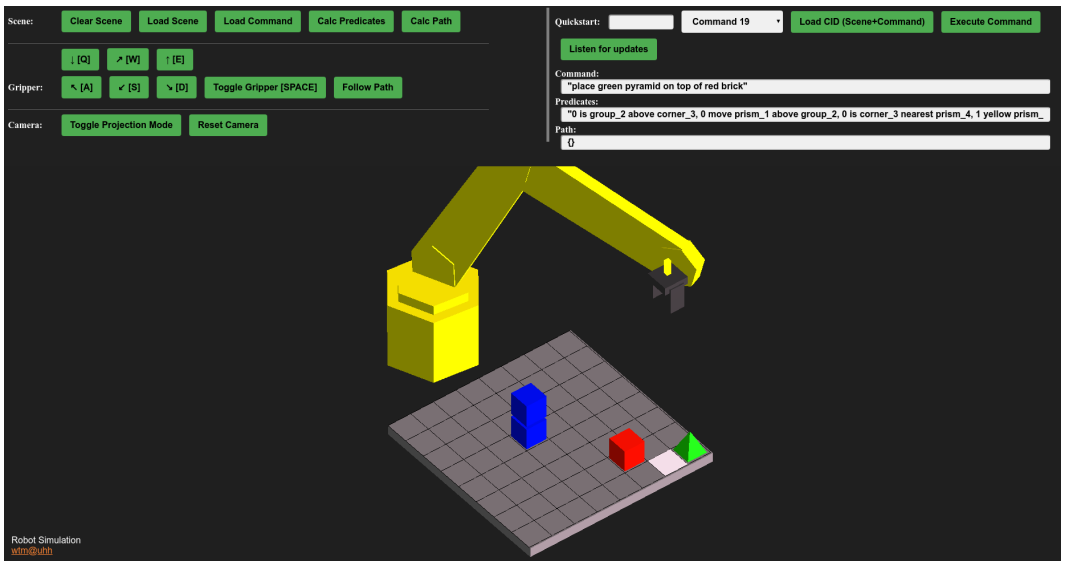


Figure 4.12: Screenshot of the web-based simulator. It can be controlled by the user (buttons on the left side) and scenes from the data set can be loaded (right side). The simulator can also be controlled via an API to be able to connect it to our bidirectional processing chain.

As the output of the system are only source and destination coordinates, a path between these coordinates is missing before a command can finally be executed. We employ the A* algorithm (Hart et al., 1968) to compute this path. It is commonly used as a pathfinding algorithm within a graph, which guarantees the shortest path if an admissible heuristic is provided. As a heuristic, we use the Euclidean distance between two points. As points, we define the discrete positions in our 8*8*8 grid. Positions holding a shape are removed before the graph is created. This way, collisions are prevented. The algorithm has to be employed twice, first, to get the path from the actual gripper position to the source position and then to calculate the path from the source to the destination.

We also applied our approaches in a real-world robotic scenario. Figure 4.13 shows the Neuro-Inspired COmpanion (NICO) robot (Kerzel et al.,

2017), which is a humanoid developmental robot with diverse capabilities like grasping. In Figure 4.13, the robot received the voice command:

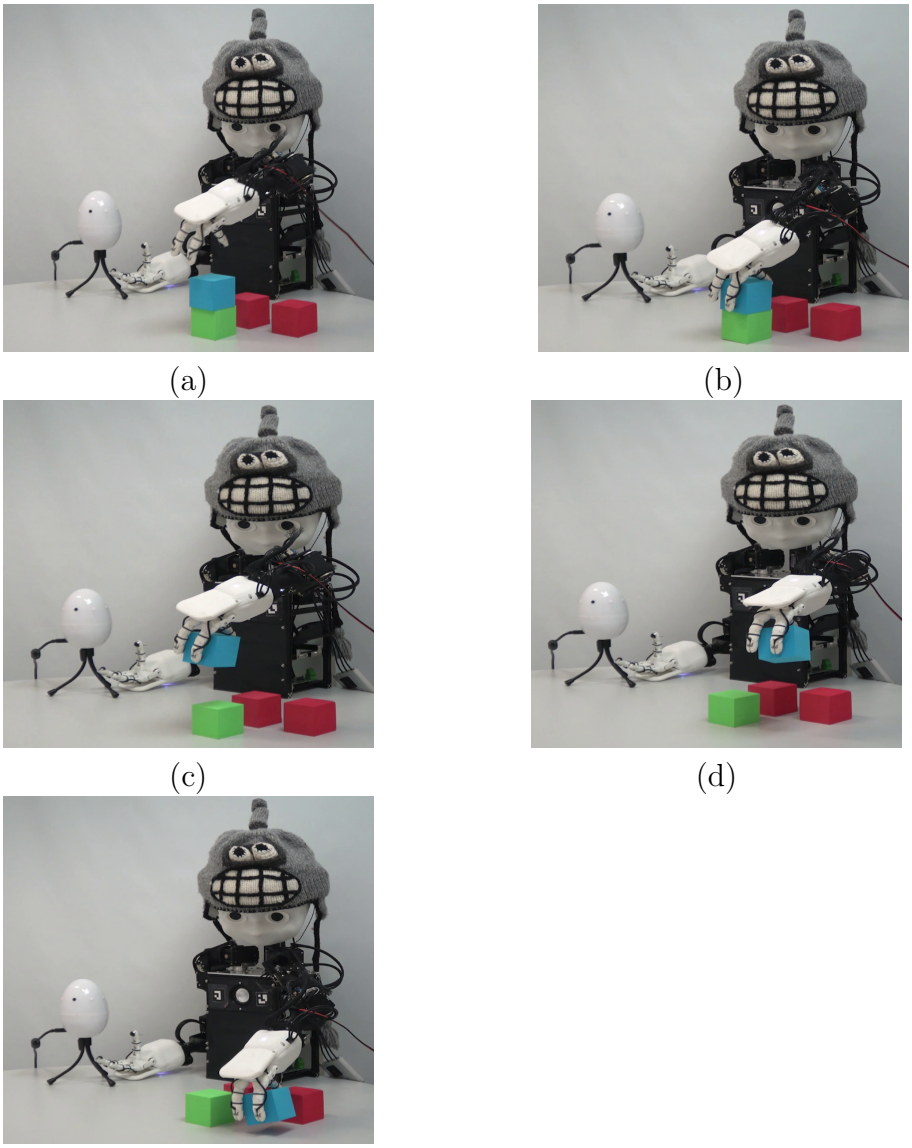
`take the cube box that is on top of the green cube and place it in the top left corner`

The audio is captured by a Tamago microphone[8] (the egg-shaped object left to the robot). Then, the audio is processed by the modules described before, producing source and destination coordinates. These coordinates need to be mapped to real-world positions. In the end, the robot is receiving joint angles to perform the movement.

The approach behind the grasping is not part of this work and will only be described briefly and fully published in another paper. To initialize the algorithm, a reference point is defined by placing a cube in the bottom left corner and manually moving the arm of the robot to that position. In this real-world scenario, we use a grid of 2*3*2 discrete positions (Y is 3, the others 2). The positions are calculated using the reference point. The calculation of the joint angles is then performed by an approach based on the work of Starke et al. (2016) and uses inverse kinematics to calculate the joint angles for the discrete positions employing a genetic algorithm. It only needs to be run once, as all joint angles can be precomputed. This way, the response time of the algorithm is very low, because only cached values need to be recalled.

In this scenario, the cubes are stackable in a 2*2 grid. The highest level is reserved for navigation. This way, the robot can more or less safely move a cube from one position to another without knocking over other cubes. Also, path planning is not necessary. The grid has a size 2*2 is chosen because the robots grasping capabilities are limited and it can not reach objects in a larger grid; with another robot, larger grids are possible.

[8] http://www.sifi.co.jp/system/modules/pico/index.php?content_id=39



(a) (b) (c) (d) (e)

Figure 4.13: NICO robot performing the action: `take the blue box that is on top of the green cube and place it in the top left corner`.

For example (Figure 4.13), the blue cube is moved from position (1, 1, 1) to position (0, 0, 1). The coordinates (X, Y, Z) are left (0) and right (1), up (1) and down (0), and forward (1) and backward (0) from the intrinsic perspective of the robot. From the source position (1, 1, 1), the blue cube is lifted up to the highest level (1, 2, 1), then moved towards the destination while holding the highest level (0, 2, 1). From there, the cube only needs to be dropped at the destination position (0, 0, 1) by lowering the level. After positioning the cube at the destination, the internal representation of the scene is updated and the robot can continue to move objects around.

It would be possible to extend the scenario to support a larger grid. Only motion limitations of the robot need to be taken into account. The approach does not require vision, as the decision and validation are performed only using its internal representation of the scene. Only at the beginning, the internal representation and the actual scene need to be made consistent. In future work, the approach can be extended to use vision to derive the internal representation from an RGB image.

4.10 Summary

In this chapter, we presented the novel natural language representation called *SemaPreds*, and the components needed to build our proposed bidirectional processing chain for speech-controlled scenarios. The following figure of the architecture shows the arrangement of the different modules (Figure 4.14). The bidirectional chain receives speech utterances from a user and, in this case, scene layouts from an HRI scenario. For other scenarios, other context-based data could be provided. The audio data is processed by our *SlimSpeech* ASR system, where different internal language models like domain-dependent or general-purpose N-gram models may be used. The next (optional) step is to postprocess the text output with an external language model like *Sentence Template* models. Then, the *SemaPred Recognizer* extracts *SemaPreds*

from the text. These are fed to the *Crossmodal Corrector*, that contains the *Semapred Interpreter*. It tries to interpret the *SemaPreds*; if this fails, the potential errors are found using the *Semantic Evaluator*. The parts of the *SemaPreds*, where errors are assumed, are tried to be corrected using *wildcards*. If the *SemaPreds* can be corrected, the simulator or a real robot can perform the actions. If the *SemaPreds* cannot be corrected, the user can be informed about the kind of error.

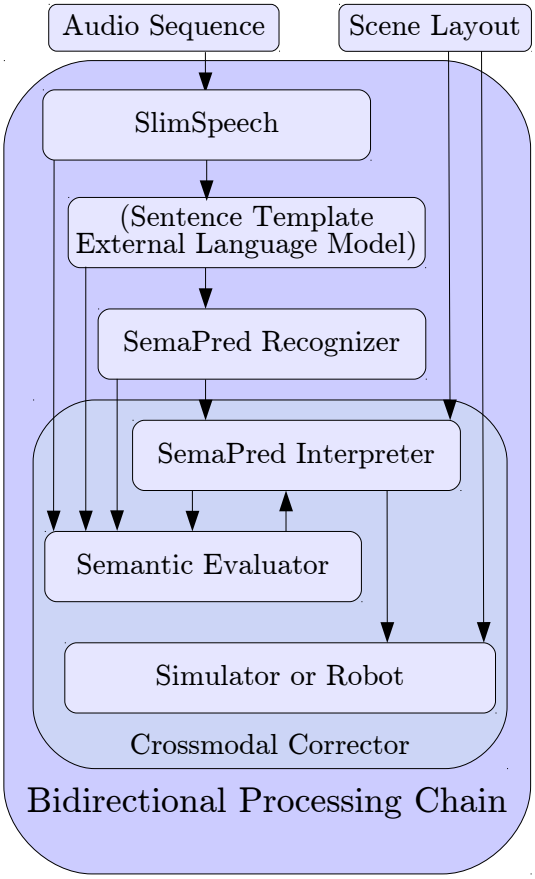


Figure 4.14: The architecture of the bidirectional processing chain.

Chapter 5

Experiments and Results

5.1 Introduction

This chapter contains the descriptions of the experiments we conducted to be able to answer our research questions. First, we provide an overview of the used data sets (Sec. 5.2), namely the TIMIT Core Test Set (Sec. 5.2.1), a benchmark data set for testing speech recognition performance. Then, we introduce our *Knowledge Technology Train Robots* data set (Sec. 5.2.2), containing speech, text, NLP labels, and other data. Afterwards, we provide a list of approaches we evaluate in our experiments (Sec. 5.2.3). The next subsection (Sec. 5.3) contains the experiment setup and results for our first experiment, where we measure the performance of our ASR system on the TIMIT Core Test Set. The next experiment (Sec. 5.4) measures the performance of our ASR system, together with our internal and external language models within a restricted domain. The performance is measured on clean and noisy audio data. The following experiment (Sec. 5.5) contains the results of the hyper-parameter optimization for our *SemaPred Recognizer*. Then (Sec. 5.6), we measure the performance of our ASR together with the *SemaPred Recognizer* to recognize text from speech and then create *SemaPreds*. The experiment

is performed on clean speech data; the following experiment (Sec. 5.7) has the same setup, but with noisy speech data. In the next two experiments (Sec. 5.8; Sec. 5.9), we extend the pipeline to process the *SemaPreds* with our *SemaPred Interpreter*. The first experiment uses clean speech data, the other one noisy speech data. After that, we repeat the same experiments (5.10; Sec. 5.11), adding our *Crossmodal Corrector* to the pipeline transforming it to a bidirectional processing chain. In the last experiment (5.12) we measure the processing time of the different chain modules.

5.2 Evaluation Data Sets and Evaluated Approaches

5.2.1 TIMIT Core Test Set

The TIMIT Core Test Set consists of 192 different sentences by 24 speakers using close-talking microphones (Garofolo et al., 1993). As the sentences spoken are very diverse, there is no speech recognition grammar available for TIMIT. The test set contains sentences like:

- `She had your dark suit in greasy wash water all year.`
- `Don't ask me to carry an oily rag like that.`
- `Production may fall far below expectations.`

The data set is commonly used to evaluate the performance of acoustic models, as the vocabulary is not domain-dependent and domain-dependent language models cannot be used.

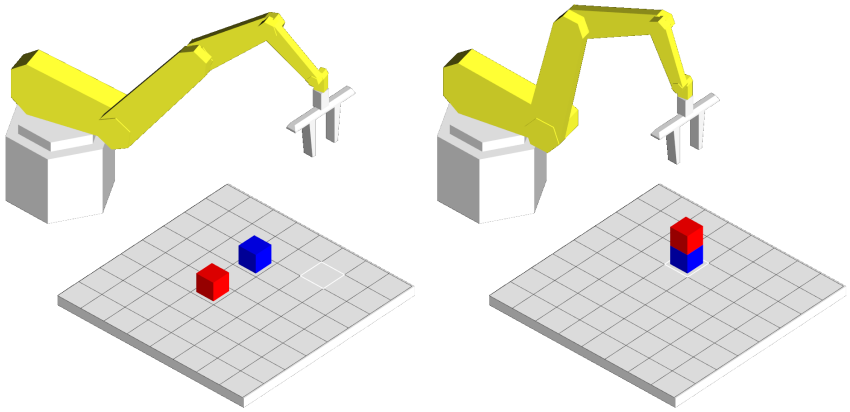


Figure 5.1: Example from the corpus. An example for a board scene before (left board) and after (right board) the command `Move the red brick on top of the blue brick`.

5.2.2 Knowledge Technology Train Robots Data Set

To evaluate our system, we created an extended version of the *Train Robots* data set (Dukes, 2013b); we call it *Knowledge Technology Train Robots* (KTTR). It consists of instructions directed at a robot arm that can move objects around like boxes and pyramids in a discrete world and is inspired by SHRDLU (Winograd, 1972). The text corpus contains 2,500 training sentences and 909 test sentences, which are linguistically rich, including anaphoric references, multi-word spatial expressions, and challenges like lexical disambiguation. The data set has a "before" and "after" scene for each of the commands, Fig. 5.1 shows an example. The sentences are, for example:

- `Move the red brick on top of the blue brick`
- `Pick up the blue block from the top of the green block and put it down on the blue block which lies next to another green block`

The original corpus was created via crowd-sourcing and contains many grammatical and syntactic errors, which would make a model learn incorrect

syntax. A prerequisite for our approach to work is that the system learns the correct syntax of the language. For restricted domains like the *Train Robots* scenario, we believe that there is not enough data available to be able to identify incorrect syntax in an unsupervised way. That is why we corrected grammatical errors in the training data to make sure the system learns only the correct syntax. We recorded audio data for the 909 test sentences with 9 different non-native speakers, each one uttering 101 sentences from the test set.

We also used the recorded audio to generate another very challenging variant of the data set by employing *audio-degrader*[1]. The gain is reduced, the sound is mixed with ambience sound (signal-to-noise ratio: 18dB) from a pub and convolved with a smartphone microphone input response (wetness level: 0.8). This way, very challenging conditions can be simulated. The original data set contains labels in the form of *Robot Control Language* (RCL). RCL is a variant of dependency trees in combination with semantic prototype words at the leave of the trees, e.g., `grab` and `grasp` are both mapped to the prototype word `take`. For all the 3,409 sentences, we annotated *SemaPreds*. The data set is used to evaluate the performance of our language models in combination with ASR and to measure the recognition of *SemaPreds* using a pipeline of ASR, language models, and the *SemaPred Recognizer*. The layout data is used to evaluate our *SemaPred Interpreter* and the *Crossmodal Corrector*.

5.2.3 Evaluated Approaches

The following list contains the models we analysed in our experiments together with their abbreviations. The :

- SSC: SlimSpeech Chars acoustic model
- Google: Google's cloud-based ASR system

[1] https://pypi.org/project/audio-degrader/

- SSP: SlimSpeech Phonemes acoustic model
- Gold Standard Test Sentences: the test sentences from the data set
- Gold Standard Test *SemaPreds*: the test *SemaPreds* from the data set
- GP3: internal general-purpose trigram language model
- GP4: internal general-purpose quadrogram language model
- DD2: internal bigram language model trained on the training sentences
- STG: *Sentence Template Grammar* model trained on the training sentences
- STN: *Sentence Template N-gram* model trained on the training sentences
- DBi: external DOCKS bigram model trained on the training sentences
- SPR: the *SemaPred Recognizer* creating *SemaPreds* from speech hypotheses
- SPI: the *SemaPred Interpreter* which creates source and destination coordinates from *SemaPreds* and scenes.
- CC: the *Crossmodal Corrector* that can be used to find the unique source and destination coordinates if the SPI did not deliver a unique solution

5.3 General-purpose Speech Recognition Performance

We evaluate our acoustic model on the *TIMIT Core Test Set*. The performance is measured for Word Error Rate (WER) and Phoneme Error Rate (PER). To be able to measure the PER, the best text hypothesis of each approach and

reference text is converted to phonemes using SequiturG2P (Bisani and Ney, 2008) trained on CMUdict.

Table 5.1 shows that the performance of *SlimSpeech* regarding WER is slightly lower than Google's cloud-based ASR system. If we employ a trigram general-purpose language model (SSC-GP3) there is a huge performance improvement (25.083 % vs. 18.807 %). When using a quadrogram general-purpose language model, the performance is even better (17.575 %).

The results in Table 5.1 indicate that all our local ASR systems perform better than Google's ASR regarding PER. Using our acoustic model without a general-purpose language model (SSC) leads to a better phoneme quality than Google's ASR achieved. If we do not convert the phonemes from graphemes but produce them directly from audio (SSP), the PER performance is even better. The best PER performance is achieved using SSC-GP3 (35.019 %) and SSC-GP4 (32.555 %).

Table 5.1: Performance regarding Word Error Rate and Phoneme Error Rate on the *TIMIT* data set.

Internal Acoustic Model	Internal Language Model	Word Error Rate	Phoneme Error Rate
SSP	-	n/a	41.15 %
SSC	-	25.551 %	44.682 %
Google	-	25.083 %	47.86 %
SSC	GP3	18.807 %	35.019 %
SSC	GP4	17.575 %	32.555 %

5.4 Domain-dependent Speech Recognition Performance

In this experiment, we measure the performance of the presented general-purpose ASR systems and compare it with the performance of the domain-dependent approaches. The experiment is performed on the clean and the noisy KTTR dataset. The language models for the domain-dependent ASR systems DBi, DD2, STG, and STN are trained on the 2500 training sentences of the given corpus. Figure 5.2 shows the results on the clean test set.

The best-performing approach is our *SlimSpeech* system with an internal DD2 language model. It is followed by the approaches using an external DBi language model. Then, the approaches that use STN language models follow, and the general-purpose ASR systems and, finally, the ones using STG language models. For all external language models, the ones using Google's speech hypothesis as input perform slightly better than their equivalents using the output of SSC as input. For all external language models, the phoneme-based *SlimSpeech* (SSC) performs the weakest. All internal (DD2, GP4, GP3) and external language models (DBi, STN) except STG, together with our SSC or SSP system, perform better than Google's ASR. The huge gap in performance can be observed when comparing Google's ASR (15.854 %) against our best performing approach SSC (3.85 %) which is an absolute WER reduction of 12.004 %.

Figure 5.3 shows the performance of the different approaches on the noisy KTTR dataset. The performance dropped drastically for all approaches. Like on the clean KTTR data set, the performance of the SSC-DD2 is the best, followed by the external DBi language models with character input, while the performance with phoneme input is much lower. It is followed by the STN language models. For the noisy dataset, the character-based STG language models perform better than the general-purpose language models

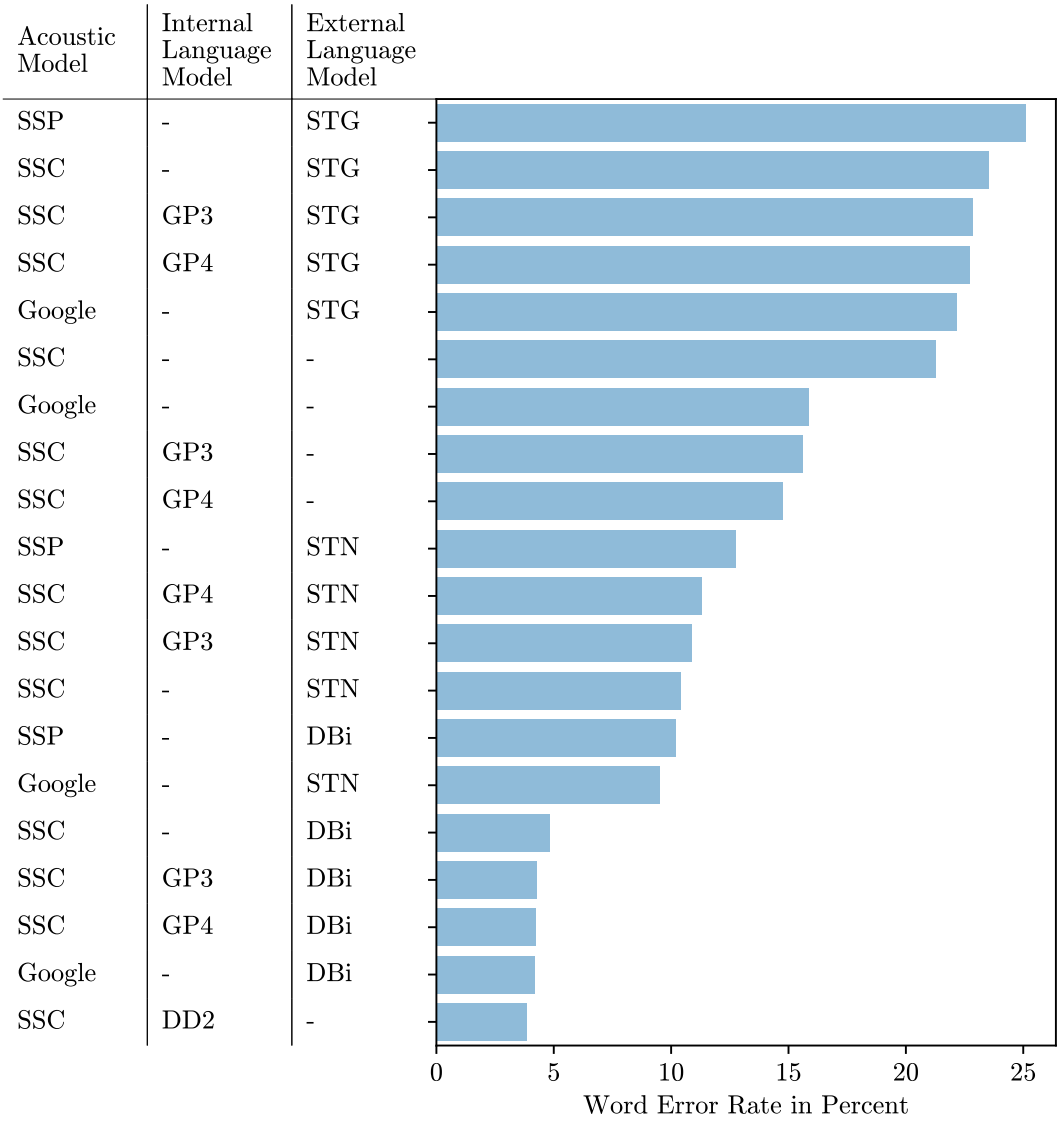


Figure 5.2: Performance regarding Word Error Rate on the clean KTTR data set.

or no language models. Here the performance gap between our best approach (SSC-DD2) and Google is even more drastic with 27.349 % vs. 49.902 % which is an absolute WER reduction of 22.553%.

Acoustic Model	Internal Language Model	External Language Model
SSC	-	-
SSC	GP3	-
SSC	GP4	-
SSP	-	STG
Google	-	-
SSP	-	STN
SSC	-	STG
SSP	-	DBi
SSC	GP3	STG
SSC	GP4	STG
Google	-	STG
SSC	-	STN
SSC	GP3	STN
SSC	GP4	STN
Google	-	STN
SSC	GP4	DBi
SSC	GP3	DBi
Google	-	DBi
SSC	-	DBi
SSC	DD2	-

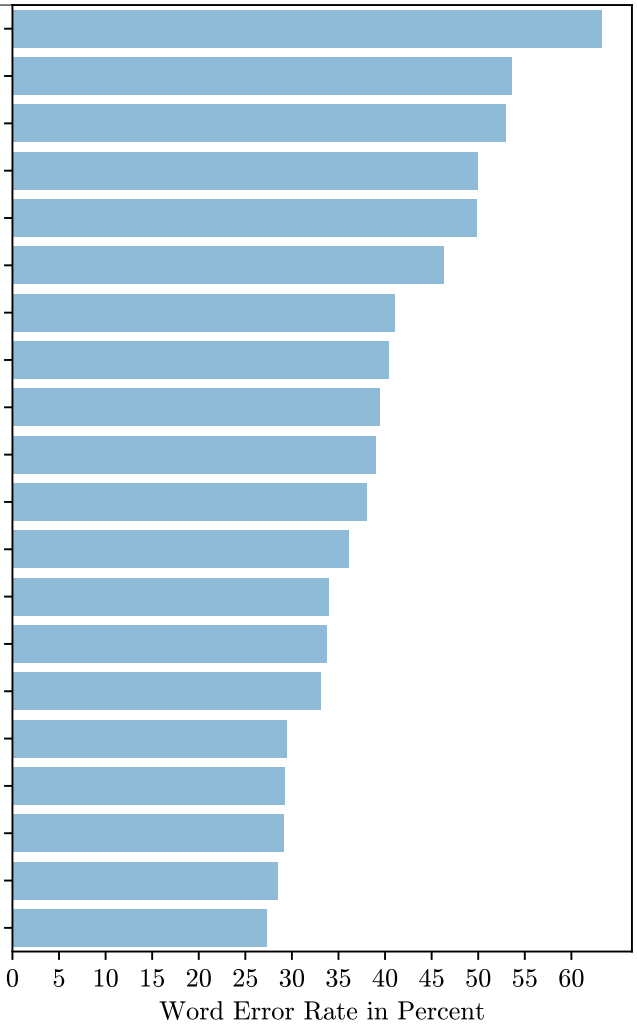


Figure 5.3: Performance regarding Word Error Rate on the noisy KTTR data set.

5.5 Hyperparameter Optimization for the Semantic Logic Predicate Recognizer

To find fitting hyperparameters for our model, we use *hyperopt* (Bergstra et al., 2015) with a *Tree of Parzen Estimator* iterating for 200 trials. The hyperpa-

rameters to find are the dropout probability after the *Max Pooling* layer and the dropout probability after the concatenation using a uniform distribution between 0 and 1. Also, we searched for a good *batch size* between 1 and 401. The size of the dense hidden layer was searched between 500 and 5000 neurons. To find a reasonable *learning rate*, we chose a uniform distribution between 0.01 and 0.0001. We performed a 5-fold cross-validation for each iteration. The metric to optimize measures whether an output sample was correctly predicted. For each subvector of the output, a winner-takes-all is performed, setting the winner to 1 and the rest to 0. This vector is compared to the validation sample. If the two samples are equal, the sample is correctly classified. This way, we calculated the error of the validation set, doing this once for each fold, and determined the average error rate for an iteration. The found hyperparameters are given in Table 5.2.

Table 5.2: Hyperparameters found for the KTTR data set.

Hyperparameter	Found Value
Dropout Probability 1	8.215 %
Dropout Probability 2	75.078 %
Batch Size	7
Hidden Size	757
Learning Rate	0.0001

5.6 Semantic Logic Predicate Recognition on Clean Speech

To evaluate the performance of our system for lab and real-world applications, we measured the performance of our system on the 909 test sentences (Gold) and the ASR hypotheses produced by our ASR systems. We tested different combinations of acoustic models, internal language models and external language models to generate the hypotheses. The hypotheses are then fed to

a *SemaPred Recognizer* that is producing *SemaPreds*. The quality of these *SemaPreds* is analysed in this experiment. Figure 5.4 shows the results for the clean KTTR data set.

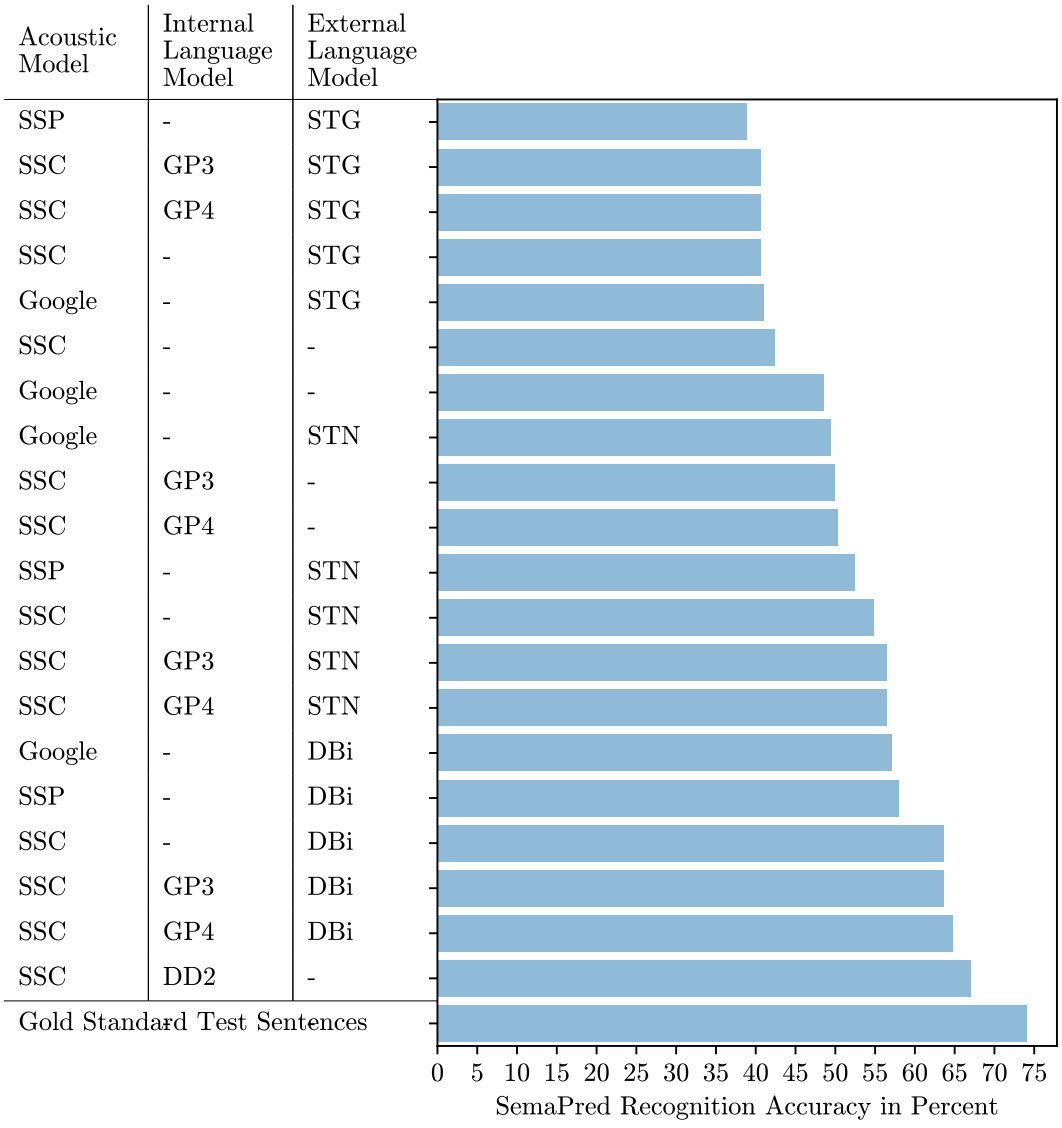


Figure 5.4: Performance regarding *SemaPred* recognition accuracy on the clean KTTR data set.

When trained on the 2,500 training sentences and training *SemaPreds*, the model achieves an accuracy of 74.147 % for the 909 test samples (Gold

Standard Test Sentences). A *SemaPred* output consists of up to two actions, up to two relations, and up to 25 attributes. If all *SemaPreds* for a sentence are classified correctly, the output is considered as correct. The accuracy is calculated on the number of correctly classified samples. Partly incorrect outputs are considered as incorrect, making it a strict metric.

The best accuracy when using ASR outputs as inputs to the system is achieved by the SSC-DD2 model. The other n-gram-based models achieve a slightly lower performance, except the SSP-DBi achieving an obviously lower performance. The performance of the STN-based models is even lower. The best STN-based approaches use an internal general-purpose language model (Google, SSC-GP3, SSC-GP4). Then, the models with general-purpose language models follow, afterwards, the approaches using STG language models. Comparing our best approach (SSC-DD2) with the performance of a *SemaPred Recognizer* taking test sentences as input, the gap in performance is not high (67.107 % vs. 74.147 %). This means that our best approach achieves around 90.5 % of the performance that could theoretically be achievable. If we compare our best approach against Google's ASR together with a *SemaPred Recognizer*, the performance gap is much larger with 67.107 % vs. 48.625 %, which is an absolute improvement of 18.482 % or a relative improvement of around 138 %.

5.7 Semantic Logic Predicate Recognition on Noisy Speech

This time, we want to measure the quality of the produced *SemaPreds* under noisy speech conditions. The same experiment as in Section 5.6 is performed, again the *SemaPred Recognizer* is trained on the 2,500 training samples. This time, the noisy KTTR audio test set is used. We produce speech hypotheses using different combinations of acoustic and language models. These hypothe-

ses are fed to the *SemaPred Recognizer*. Figure 5.5 contains the results of this experiment.

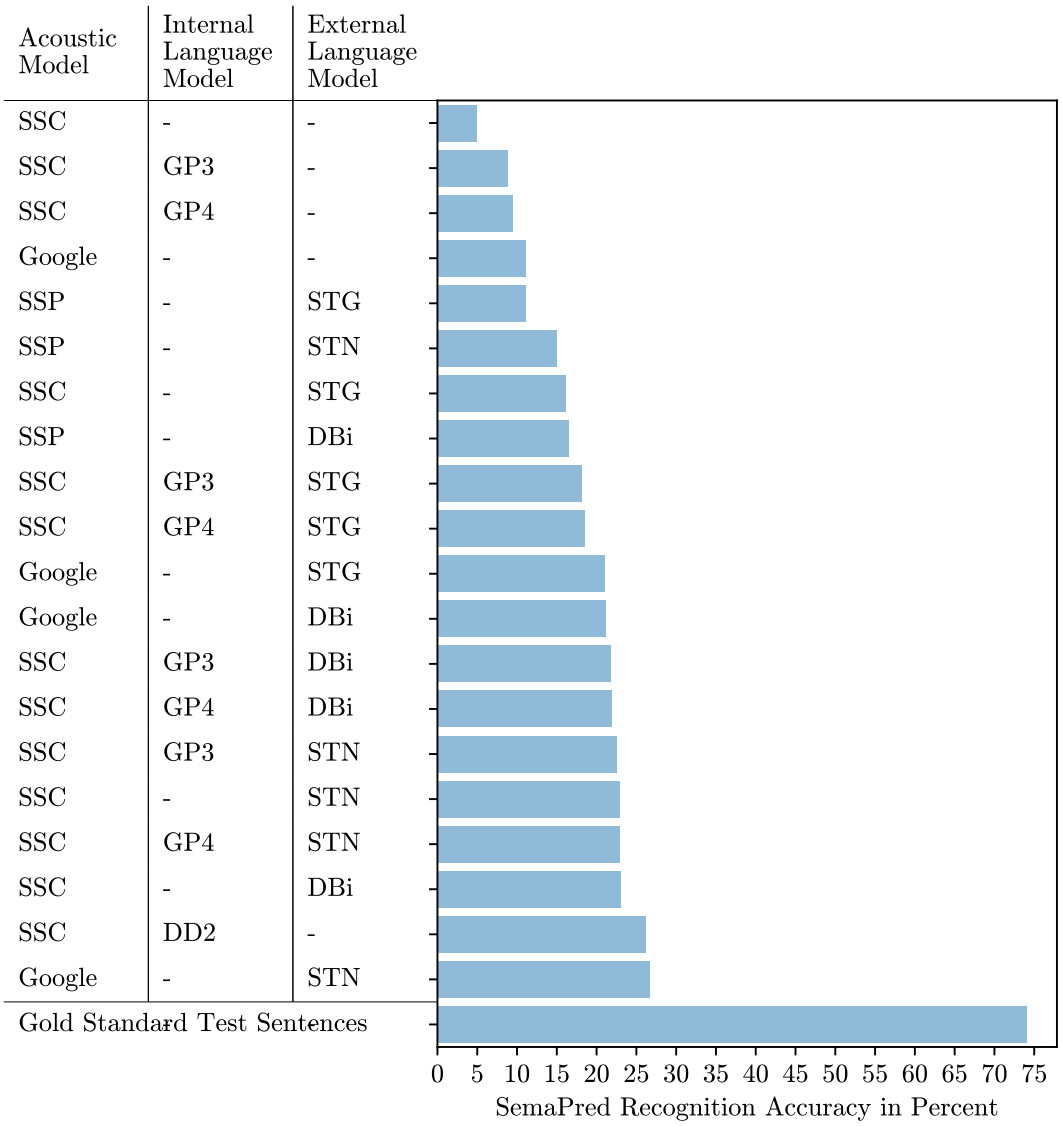


Figure 5.5: Performance regarding *SemaPred* recognition accuracy on the noisy KTTR data set.

The overall performance is obviously lower for all approaches. The best performing combination is the STN language model taking Google's ASR hypotheses as input. It is followed by SSC-DD2 and the other STN language

models using character input. Then, there are the DBi models using character input, and the STG-based models. The lowest performance is achieved when using general-purpose ASR systems. Under noisy conditions, the performance of the STN-based models is almost equal or better than the performance of other external language models. If we compare the accuracy of our best approach Google-STN against Google without an external language model (26.733 % vs 11.111 %), we observe an absolute improvement of 15.622 % or a relative improvement of 240.6 %. When we compare to our best SSC-based model (SSC-DD2), the absolute improvement is 15.072 %, and the relative improvement is 235.649 %.

5.8 Semantic Logic Predicate Interpretation on Clean Speech

In this experiment, we measure the *SemaPred* interpretation accuracy under clean conditions using the clean KTTR dataset. The *SemaPreds* are produced using the *SemaPred Recognizer*. As input to the *SemaPred Recognizer*, we use the speech hypotheses produced by the different ASR combinations. The speech hypotheses are the same ones as in experiment 5.6 and the produced *SemaPreds* are the same as in experiment 5.6. The *SemaPreds* are then processed by the *SemaPred Interpreter* (SPI). The dataset consists of 909 samples and scenes. Each scene consists of an initial layout and a subsequent layout, where an entity and the gripper is moved to a different position in the grid. After processing the *SemaPreds*, we receive a source and a destination position of the gripper. We simulate a move by changing the gripper position to the destination position and the position of the entity at the source position to the destination position. Finally, we compare the positions of all entities of the changed layout to the subsequent layout of the dataset. If the layouts are equal, the sample is considered as interpreted correctly, otherwise as incorrectly.

Figure 5.6 shows the results of the experiments. The SPI almost reaches 100% accuracy when processing gold standard *SemaPreds*. If we use the gold standard test sentences to produce *SemaPreds*, the accuracy is 79.538 % which should be the maximum performance achievable when using ASR hypotheses as input. The best ASR model achieves 73.157 % accuracy (SSC-DD2). It is followed by the DBi-based models and then STN-based models using SSC as input. The best system without a domain-dependent language model is SSC-GP4, with 56.986 %. When using Google's ASR without a domain-dependent language model, the accuracy is 56.436%. Comparing our best model (SSC-DD2) against Google, we observe an absolute improvement of 16.721 % and a relative improvement of 28.377 %.

5.9 Semantic Logic Predicate Interpretation on Noisy Speech

To investigate the performance of the *SemaPred Interpreter* under very noisy audio conditions (see Sec. 5.2.2), perform a similar experiment as in section 5.8. This time, we choose the noisy KTTR data set to produce the ASR hypotheses, which are the same as in experiment 5.4, and use the *SemaPred Recognizer* to recognize *SemaPreds*. These *SemaPreds* are the same ones as in experiment 5.7. The *SemaPreds* are processed by the *SemaPred Interpreter* (SPI). The produced source and destination coordinates are employed to simulate a gripper movement like in experiment 5.9. Under very noisy conditions, the accuracy of the best approach is 32.233 %. Those are SSC-DD2 and Google-STN. Compared to the clean conditions, the STN-based approaches perform superior compared to the DBi-based approaches in most cases. The weakest performance is achieved by SSC when using no language model at all. Google's accuracy is only 15.622 %. When comparing Google to our best approaches (SSC-DD2, Google-STN), we observe an absolute improvement of 16.611 % and a relative improvement of 106.90 %.

Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor
SSP	-	STG	SPI
SSC	GP3	STG	SPI
SSC	GP4	STG	SPI
SSC	-	STG	SPI
Google	-	STG	SPI
SSC	-	-	SPI
Google	-	STN	SPI
Google	-	-	SPI
SSC	GP3	-	SPI
SSP	-	STN	SPI
SSC	GP4	-	SPI
SSC	-	STN	SPI
SSC	GP3	STN	SPI
SSC	GP4	STN	SPI
Google	-	DBi	SPI
SSP	-	DBi	SPI
SSC	-	DBi	SPI
SSC	GP3	DBi	SPI
SSC	GP4	DBi	SPI
SSC	DD2	-	SPI
Gold Standard Test Sentences			SPI
Gold Standard Test SemaPreds			SPI

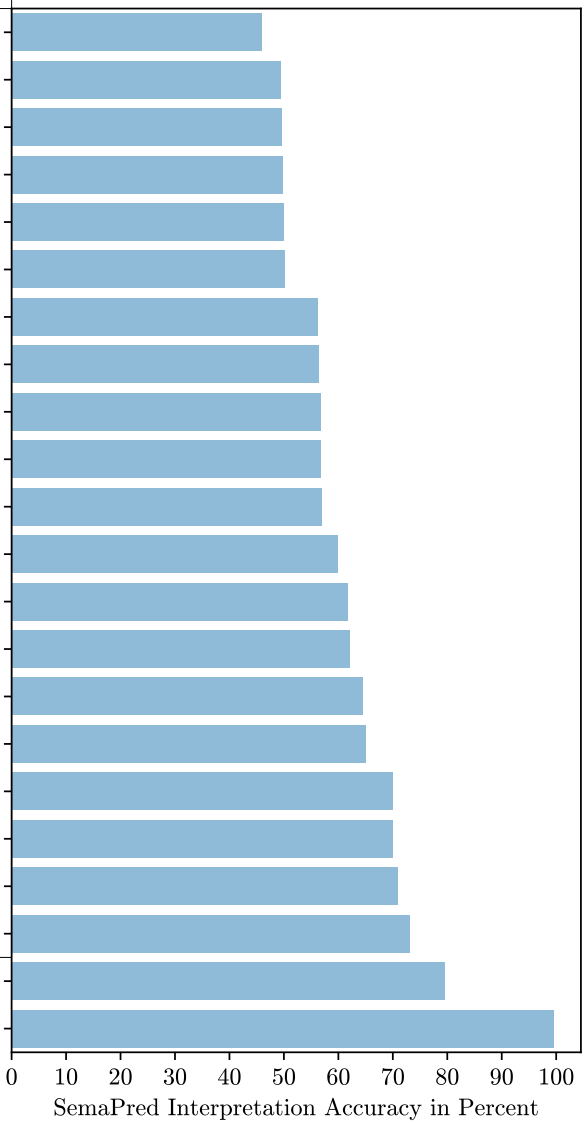


Figure 5.6: Performance regarding *SemaPred* interpretation accuracy on the clean KTTR data set.

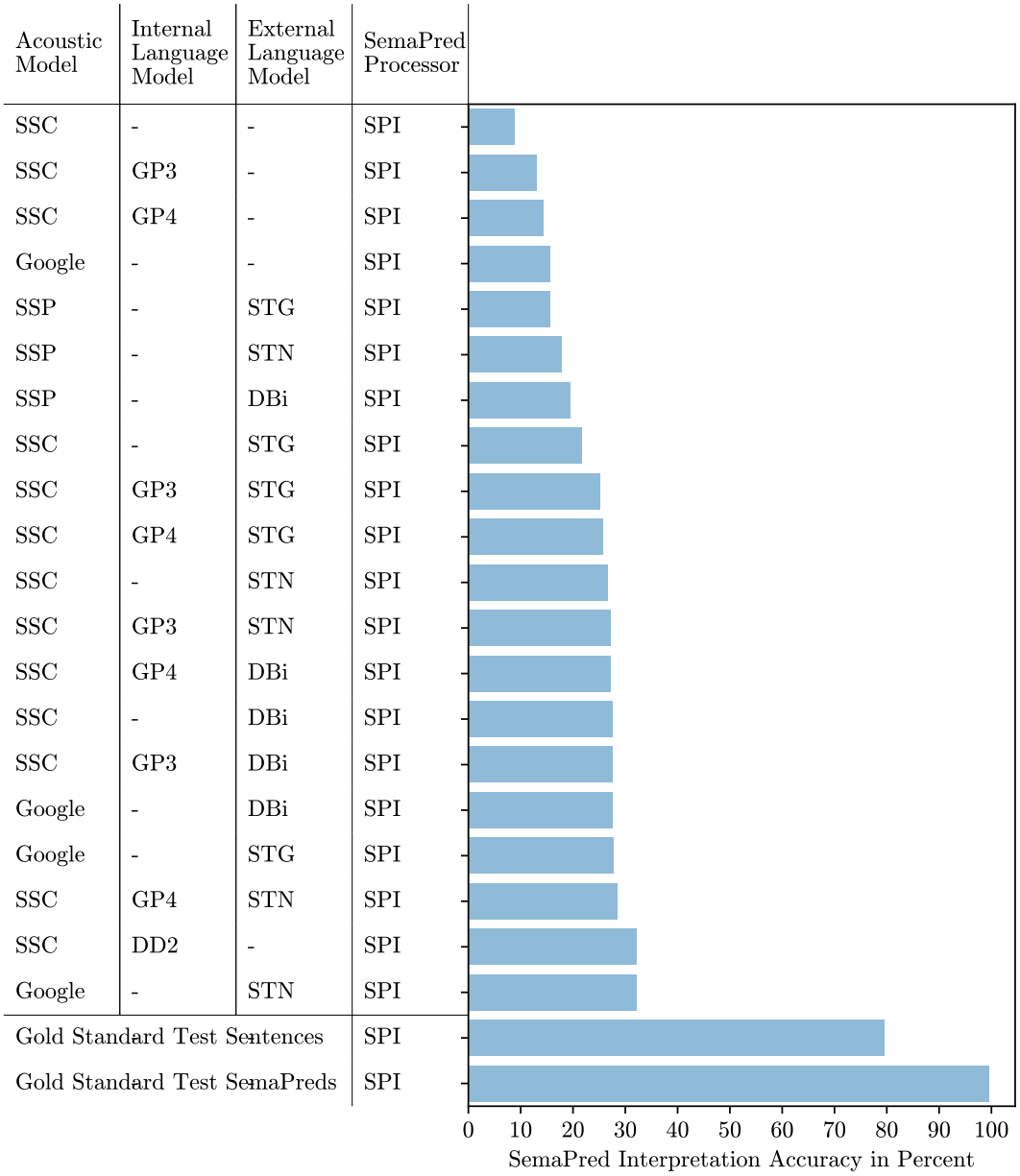


Figure 5.7: Performance regarding *SemaPred* interpretation accuracy on the noisy KTTR data set.

5.10 Crossmodal Correction on Clean Speech

In this experiment, we want to measure the influence of the *Crossmodal Corrector* (CC) on the *SemaPred* interpretation accuracy. We base this experiment

on experiment 5.8. Again, the *SemaPreds* are produced using different variants of ASR systems and the *SemaPred Recognizer*. As a data set, we also chose the clean KTTR . We employ the *SemaPred Interpreter* (SPI) to produce source and destination gripper positions and simulate movements like in experiment 5.8. This time, we use the CC to attempt to correct implausible inputs to the SPI. Figure 5.8 shows the outcomes of the experiment and also contains the *SemaPred* interpretation accuracy measured in experiment 5.8 (rows having only SPI as *SemaPred* processor).

Again, the best accuracy (99.56%) is achieved when interpreting the gold standard *SemaPreds* from the test set. When using the SPI+CC instead of only using the SPI only, the accuracy is improved for all approaches. Even when feeding the gold standard sentences of the test set to the *SemaPred Recognizer*, the accuracy can be improved from 79.538% to 82.728%. The best approach (SSC-DD2) can be improved from 73.157% to 77.228%, which is close to the accuracy of using gold standard test sentences without CC (79.538%). Again, SSC-DD2 is followed by the DBi-based approaches, the STN-based approaches, and the models using a general-purpose language model. Google achieves 59.406% accuracy when employing the CC. When comparing our best system against Google using only SPI, the accuracy is 77.228% (SSC-DD2-SPI+CC) versus 57.096% (Google-SPI).

5.11 Crossmodal Correction on Noisy Speech

This experiment investigates the influence of the *Crossmodal Corrector* (CC) on the *SemaPred* interpretation accuracy under very noisy conditions. We produce speech hypotheses using the different ASR combinations on the noisy KTTR dataset like in experiment 5.4. The speech hypotheses are processed by the *SemaPred Recognizer*, and the *SemaPreds* are processed using the *SemaPred Interpreter* (SPI) in combination with the CC. To be able to measure the influence of the CC, we compare the performance with the SPI without

CC. Figure 5.9 shows the results of this experiment.

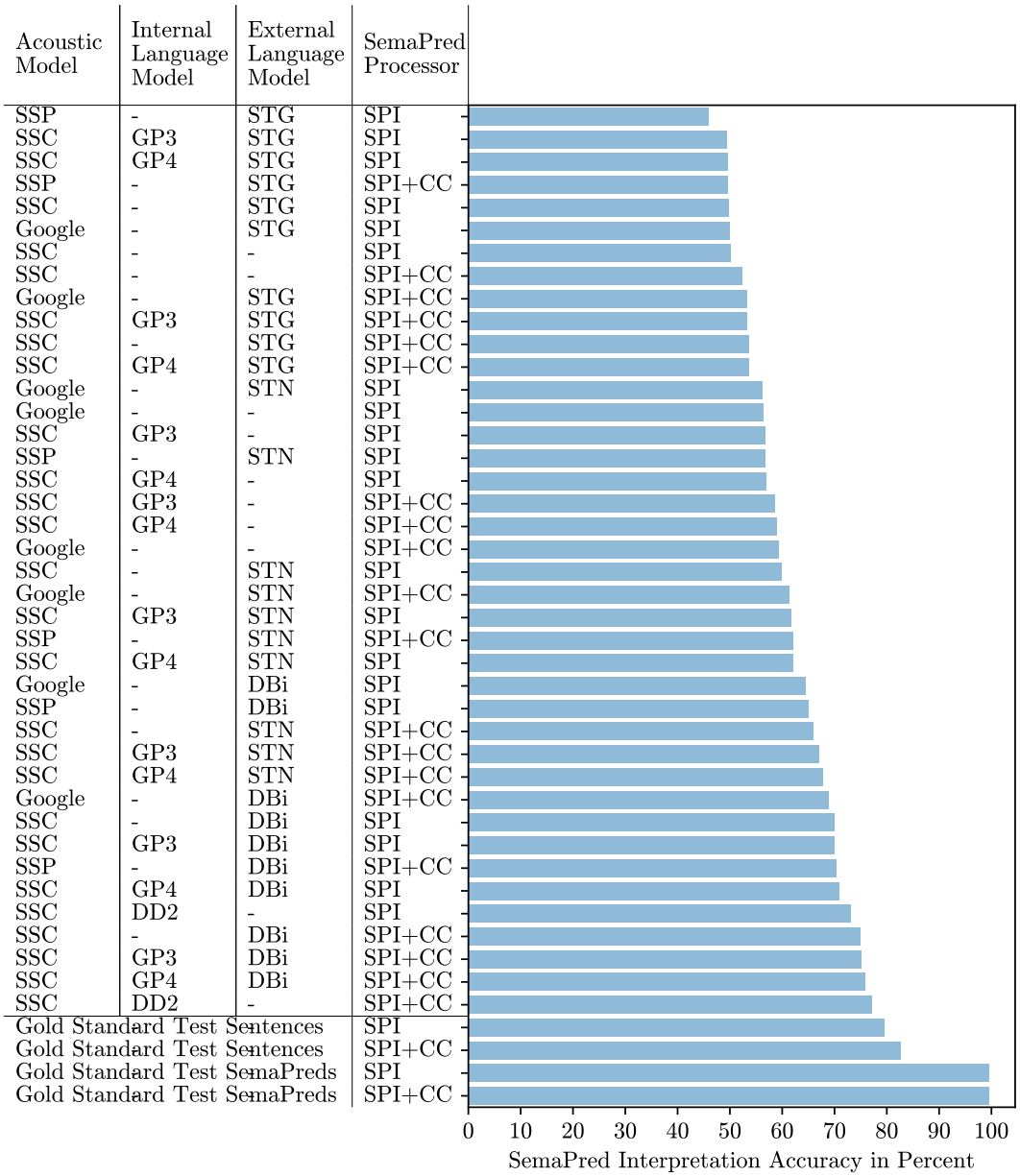


Figure 5.8: Performance regarding *SemaPred* interpretation accuracy on the clean KTTR data set.

For all combinations of ASR models, the performance is improved when using a CC compared to only using an SPI alone. The best performance is

achieved by the STN model in combination with Google's ASR as an acoustic model (36.634 %) and SPI with a CC. It is followed by the SSC-DD2 with the SPI and the CC (35.204 %). When comparing to Google's ASR with SPI and CC, it performs weaker with 15.952 % accuracy. Google with SPI and without CC achieves 15.622 % accuracy.

5.12 Processing Time

In this experiment, we want to analyse the processing and response time of our different models. For this purpose, we measured the different processing times for each stage of the processing chain. Those are the time between taking the audio and producing the first text hypotheses. These are created using Google's ASR or using our SSC or SSP acoustic model and an internal language model like DD2, GP3, or GP4 or not using an internal language model at all. When using an external language model like STG, STN, or DBi, this step is also measured separately. Afterwards, we measure the time to create *SemaPreds* from the speech hypotheses. Thereafter, the time to interpret the *SemaPreds* using the SPI is noted. Finally, we measure the processing time when using a CC to improve the results. The measurement is started, when a module receives the input data and stopped after a module produced the output.

Figure 5.10 shows the results of this experiment. The fastest ASR system is SSC with 0.35s. When using an internal language model like GP4, GP3, or DD2, the response time rises to 0.461s, 0.459s and 0.436s. This is very fast compared to Google's response time of 1.271s. When creating *SemaPreds* from a speech hypothesis, the processing time is around 0.012s to 0.017s. The processing time for the external language models varies depending on the connected ASR system. DBi takes 0.334s to 0.435s, STN's processing time is between 0.651s to 1.407s.

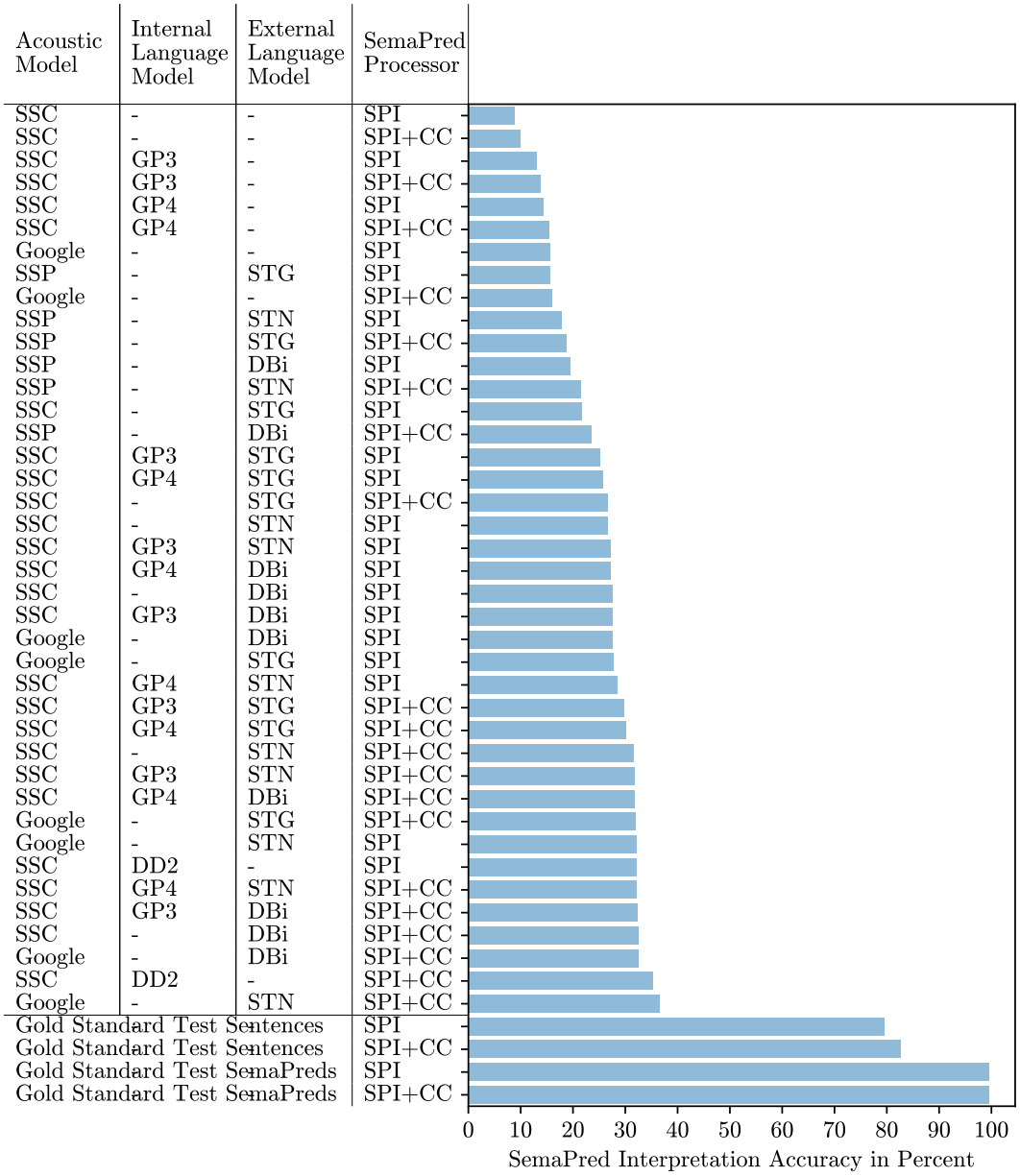


Figure 5.9: Performance regarding *SemaPred* interpretation accuracy on the noisy KTTR data set.

The processing time of the SPI varies between 0.216s and 0.524s. For the CC, the processing time is only measured if it was really used. This was the case when the SPI did not deliver a unique solution. Otherwise, the processing

time of CC is 0. The average processing time for CC is between 0.051s and 0.128s.

Regarding the whole processing chain from the acoustic model to the CC, the fastest approach is SSC-GP4 with 0.785s followed by SSC (0.842s), SSC-GP3 (0.891s) and SSC-DD2 (0.96s). When defining an average response time of one second as the limit for realtime capability, these are the only approaches achieving this. When using Google's ASR together with the *SemaPred Recognizer* and SPI with CC, the processing time is 1.638s, without CC, it is 1.549s, which is still not realtime capable using our definition.

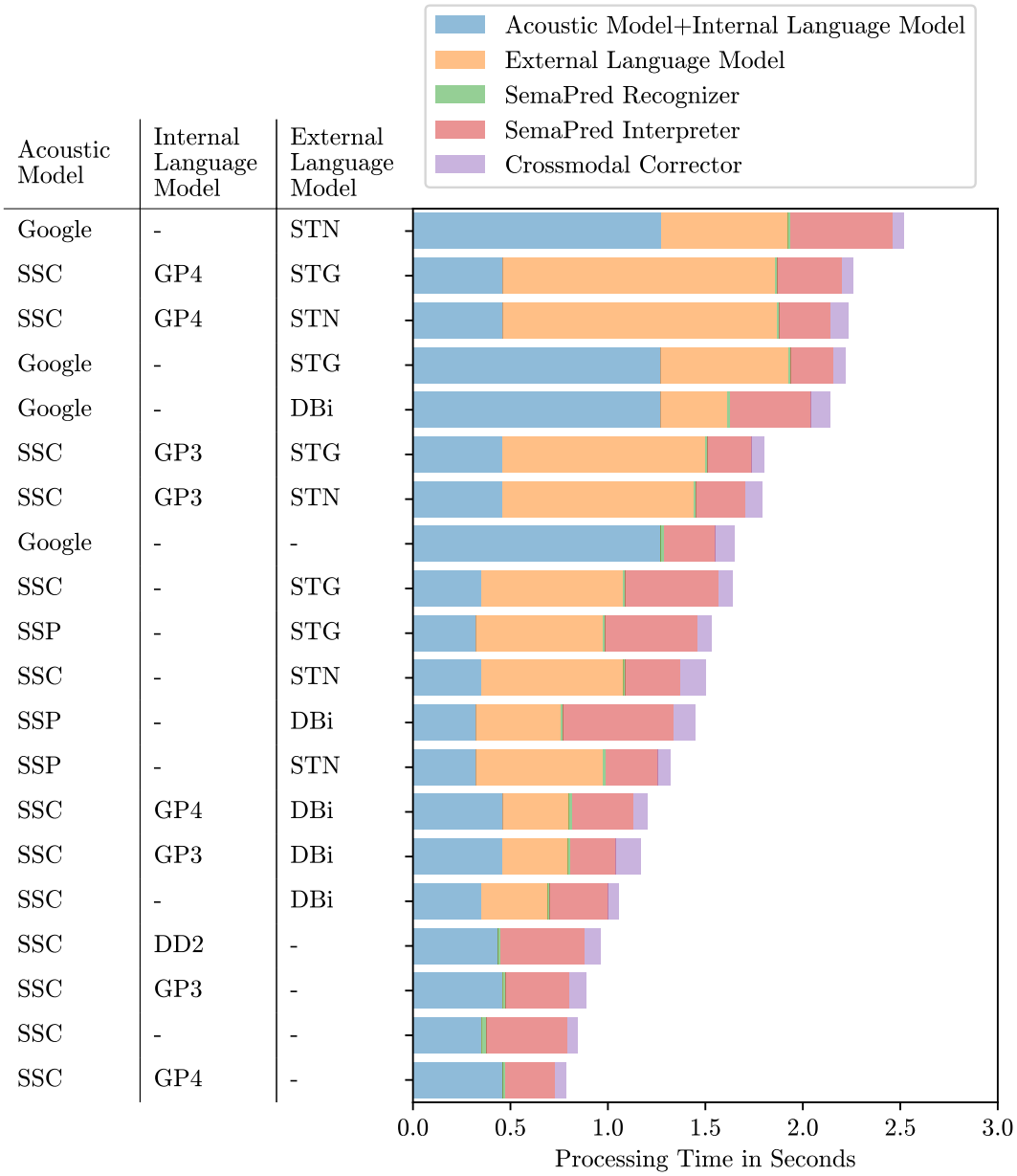


Figure 5.10: Average processing time per utterance on the clean KTTR data set.

Chapter 6

Discussion and Conclusion

This chapter contains the outcomes of this thesis. In the first subsection, we mention discuss the main findings of our experiments. Then, we provide concrete answers to the research questions we derived. After that, we give an overview of the limitations and possible future work that could be performed using the contributed knowledge contained in this thesis. Finally, we provide conclusions.

6.1 Discussion

This section contains a discussion of the main findings of our experiments. In experiment 5.3, we investigated the performance of the acoustic model of our *SlimSpeech* ASR system. On the TIMIT benchmark data set, the model achieved a Word Error Rate (WER) of 17.575% when using a general-purpose quadrogram (GP4) language model, while Google's ASR achieved 25.083%. These results show that our system can be considered as an alternative to Google's ASR, at least under clean audio conditions.

In next experiment (5.4), we analysed the performance of the different combinations of acoustic models, internal language models and external language models on clean and noisy audio data. On the clean and the noisy *Knowledge Technology Train Robots* data set, *SlimSpeech* with a domain-dependent bigram (DD2) language model achieved a WER of 3.85% (clean) and 27.349% (noisy) which is vast improvement compared to Google's ASR with 15.854% (clean) 49.902% (noisy).

In experiments 5.6 and 5.7, we investigated the behaviour of our *SemaPred Recognizer*Regarding the performance of the *SemaPred Recognizer*, we observed an accuracy of 74.147% when processing text sentences from the KTTR test set. This indicates, that the *SemaPred Recognizer* contains the central limitations of our architecture and leaves room for improvements. We analysed the errors made by the system to identify possible reasons. For this, we measured the coverage rate of test sentences inside the training set. The whole test set consists of 909 sentences, 353 of them were already contained in the training set, which results in 556 completely unknown sentences (61.16%). In the experiment, the recognizer recognized (partially) wrong *SemaPreds* for 235 test sentences; all of them are from the unknown portion. We decide between outputs that possess the correct number of *SemaPreds* and outputs that have too many or not enough *SemaPreds*. 158 (67.23%) of the 235 incorrect outputs contained the correct number of *SemaPreds*. 85 of them had only one errors, meaning one word inside the list of *SemaPreds* was confused; 43 had 2 errors; 18 had 3 errors. The most common error was confusing `region` with `edge` (46) and `nearest` with `above` (22). Other prominent errors were confusing numbers like, `one` with `two`, `two` with `one`, `three` with `one` etc. The reasons for this behavior we believe to be found in overfitting and a weak coverage of the domain in the training set. As there are often only few or one error inside the output we believe that the approach can be improved in the future. The error cases are not evenly distributed leading to the hypothesis, that there could be a systematic error which could be identified. It may result from a lack of coverage of the training data. Perhaps data augmentation may

help in this case. Our *Sentence Template* language models could be used to generate novel sentences and training data for this purpose.

When the input of the *SemaPred Recognizer* came from an ASR system, the performance decreased from 74.147% accuracy to 67.107% on the clean *Knowledge Technology Train Robots* data set. The best performing ASR system is *SlimSpeech* (SSC) with a domain-dependent bigram (DD2) language model. The performance of 67.107% is good, regarding the fact that the performance on clean text is only around 7% higher, while using the text from Google's ASR, a lower accuracy of 48.625% could be achieved. Under noisy audio conditions, the best-performing model was Google's ASR together with out *Sentence Template N-gram* language model with 26.733% and SSC-DD2 with 26.183%. Google's ASR achieved 11.111%.

In experiments 5.8, 5.9, 5.10 and 5.11, we analysed the *SemaPred* interpretation accuracy on text generated by ASR systems on the clean and the noisy *Knowledge Technology Train Robots* data set. When using clean text, the *SemaPred Interpreter* achieved an accuracy of 79.538%; the best ASR approach is *SlimSpeech* with a domain-dependent (DD2) language model and the *Crossmodal Corrector*. It achieved an accuracy of 77.228%, which is marginally lower compared to using clean text. Google's ASR with the *SemaPred Interpreter* achieved 56.436 %. Under noisy conditions, the best performing approaches were Google with a *Sentence Template N-gram* language model (36.634%) and *SlimSpeech* with a DD2 language model (35.204%). Both employed the *Crossmodal Corrector*. Google'ASR together with the *SemaPred Interpreter* achieved an accuracy of 15.952%.

We analysed the response time of the different models in experiment 5.12. The fastest configuration when applying ASR, *SemaPred Recognizer*, *SemaPred Interpreter* and *Crossmodal Corrector* is *SlimSpeech* with a general-purpose quadrogram (GP4) language model with a response time of 0.785s for the whole processing chain. *SlimSpeech* with a domain-dependent bigram language model achieved a response time of 0.96s for the whole processing chain.

The ASR itself achieved 0.45s. Google's ASR with a *SemaPred Recognizer* and a *SemaPred Interpreter* achieved a response time of 1.549s.

6.2 Answers to Research Questions

The following research questions could be answered:

Question 1: How is it possible to develop a local ASR system that is usable in realtime and achieves similar performance as state-of-the-art speech recognition models?

We successfully showed that it is possible to develop such a system. Experiment 5.4 shows that the performance on an ASR benchmark dataset is similar to a state-of-the-art approach. In experiment 5.12 we show that the system is realtime capable (for our definition) with a processing time of around 0.450s against 1.271s (Google).

Question 2: Are there better alternatives for speech applications than taking a black-box ASR together with a domain-dependent NLP system?

We showed that it is possible to develop a local realtime capable speech recognition system. The acoustic model of the system is not adapted to the domain, because an acoustic model is a model for speech in general. What actually helps in improving the performance is a domain-dependent language model. The language models we use in experiment 5.4 perform, almost all, better than the reference cloud-based solution without a domain-dependent language model. In case there is no data available, the local ASR can use a general-purpose language model, which achieves a similar performance compared to a cloud-based solution. The produced text hypotheses could be recorded and corrected to be used as training data for a domain-dependent

language model. For the presented scenario, only 2,500 training sentences where enough to achieve a vast improvement in performance.

Question 3: Is it possible to define a novel NLP representation that can be corrected if parts of it are incorrect? Is the novel representation suitable to be interpreted to compute concrete execution parameters?

We defined the novel *Semantic Logic Predicates* (*SemaPreds*) representation to meet this requirement. They are less complex than dependency or RCL trees, and can be represented as logic representations, which can be interpreted by a declarative programming language like *Answer Set Programming*. If *SemaPreds* represent an instruction, an interpreter can compute concrete execution parameters like source and destination coordinates in our scenario. In other scenarios, other parameters are possible. The *SemaPred Recognizer* presented in this thesis is able to produce the novel representation from natural language. To be able to repair incorrect parts, we developed a Crossmodal Corrector (CC) that matches the output against the sentence input and identifies semantically implausible parts. These parts can be replaced with different parts that are consistent with the context, using the interpreter. The wildcard system of the interpreter makes it possible for the interpreter to find consistent variable allocations for the incorrect parts independently.

Question 4: Is the developed system also working under very noisy conditions?

Under very noisy conditions, the performance of the cloud-based ASR system is weak. When using the best variant of our local ASR system, we achieve a relative reduction in word error rate by around 54%. When connecting an NLP module (*SemaPred Recognizer*) to the Cloud-based ASR, the accuracy in *SemaPred* recognition is very weak (11.221%). We developed an external language model (Sentence Template N-Grams) that can be connected to the Cloud-based ASR and boosts the performance relatively by around

230%. When using the best variant of our local ASR system, we achieve a relative boost of around 213%. The overall performance of the system is still relatively low (25.853% accuracy), which is not surprising regarding the massive amount of noise added to the audio test data.

6.3 Future Work

The ASR system is based on DeepSpeech 2, which was state-of-the-art when we trained our system. The system we trained is not as powerful as DeepSpeech 2, because it possesses fewer layers and used less training data. In the meantime, there appeared other ASR systems which achieved a slightly better performance than DeepSpeech 2 on different benchmark test sets like LibriSpeech (Panayotov et al., 2015), and Wall Street Journal '92 and '93 (Garofolo et al., 2007; Consortium et al., 1994). For LibriSpeech, these are transformer-based models (WER: 2.3 %) (Wang et al., 2019), hybrid models (WER: 2.3 %) (Lüscher et al., 2019), self-attention (WER: 2.2. %) (Han et al., 2019), while DeepSpeech 2 achieved a WER of 5.15 %. There are also other approaches performing better than DeepSpeech 2 on this data set. For the WSJ '92 data set, only a lattice-free maximum mutual information approach achieved a better performance (WER: 2.9 %) (Hadian et al., 2018) than DeepSpeech 2 (WER: 3.1 %). In the future, we could adapt one of those infrastructures, measure the response time, and optimize the architecture or retrain a thinner architecture, and replace our SSC system.

The internal language model, which is based on HMMs, could be replaced by a Gated-CNN-based approach (Dauphin et al., 2017). This approach could also be used to extend or replace the *SemaPred Recognizer*. We showed that a domain-dependent language model provides a considerable boost in performance. Often, there is not much data available for a task (2,500 training samples for our scenario). This problem needs to be taken into account when developing a different approach for the internal language model and the

SemaPred recognition system. So far, the performance of our CNN-based system is acceptable, preliminary results with LSTMs and GRUs showed no comparable results, possibly because of the lack of training data.

The performance of the SPI is excellent for the chosen task and possibly for other tasks. A disadvantage is that it only works in discrete worlds so far, hence, we explored possibilities to discretize the real-world to a 2*2 grid when working with the NICO robot. A possible extension would be a visual system that is able to abstract discrete 3D representations from real-world scenarios. The input could be either 2D using a camera or 3D using a Kinect, and the output could be a 3D grid representation like in our scenario. The system is not limited to spatial tasks; also other tasks that can be solved using logic like question answering could be tackled with our ASR-to-execution chain.

Our external language models work on text input and are partly written in pure python. They could be optimized to work with C-based libraries to speed up processing. Also, they could be integrated with the acoustic model and to direct the decoding. When comparing the performance of DD2 and DBi language models, which are both using the same bigram language model, the performance is better for DD2 in any of our experiments. This is because DD2 works with character distributions as input, while DBi only uses phonemes. Adapting STN and STG to also work with character or phoneme distributions could lead to a boost in performance. The data set we chose for our experiments is diverse and rich of different phrasings. This is not helpful for ST-based language models, as they can only recognize known phrasing structures. If the test data contained more phrasing structures already found in the training data, the performance is expected to be better. This leads to the conclusion that ST-based language models are especially useful when there are not many variations expected in the test data.

6.4 Conclusion

In this work, we proposed different approaches for acoustic modelling, language modelling, natural language processing and a new language representation called *SemaPreds*. We evaluated the performance of our presented acoustic model and general-purpose ASR system on the commonly used *TIMIT Core Test Set*, achieving better performance than Google's cloud-based ASR. The aim of the presented approaches is to provide an alternative bidirectional processing chain from ASR to execution for future real-world applications like HRI scenarios. Depending on the results of our experiments (see Section5.3) we suggest employing our ASR system for real-world projects instead of the commonly used unidirectional pipeline approach.

The results of the experiments on clean audio (see Section 5.4, Figure 5.2) indicate that the use of a domain-dependent language model provides a clear improvement in performance (Google 15.854 % WER, SSC-DD2 3.850% WER). For low noise environments, we suggest to use a domain-dependent bigram language model (DD2 or DBi). DD2 performs better, as the decoding is performed on distributions of characters, while DBi uses only one-hot vector information.

For the type of ASR, the choice should clearly be character-based ASR (SSC, Google) instead of phoneme-based ASR models (SSP), as they perform better for each type of language model. SSC is the better option, as it allows decoding over distributions of characters (DD2), which performs best. When using external language models, we suggest using Google's ASR as an acoustic model, as the performance is slightly better than SSC with external language models. The response time for Google is much higher than SSC, which would probably not justify the slight improvement in WER and suggest SSC.

When testing on a noisy dataset (see Section 5.4, Figure 5.3), the results indicate that the whole dataset is challenging (SSC 63.335% WER, Google

49.902% WER). Using a general-purpose model (GP4, GP3) helps (SSC-GP4 52.992% WER, SSC-GP3 53.626% WER) to improve the performance, but is still lower than Google's performance. We interpret this behaviour as Google's ASR being more robust to noise. The reason could be that our acoustic models are mainly trained on clean audio data. Using a domain-dependent language model improves the performance drastically (SSC-DD2 27.349 % WER vs. Google 49.902 % WER). Again, DBi performs marginally worse than DD2 due to the reason already mentioned. Even the very restricted STG language models perform better than the general-purpose language models when using characters as input. We strongly suggest using a domain-dependent language model if training data from the domain is available.

The performance of the *SemaPred Recognizer* is acceptable (see Section 5.6, Figure 5.4) when receiving clean test sentences as input (accuracy 74.147%) regarding the accuracy to be a strict metric (see explanation in Section 5.6). When performing general-purpose ASR (Google) on clean speech data and *SemaPred* recognition is performed afterwards, the performance decreases drastically (accuracy 48.625 %). Even though the ASR performance of Google's ASR showed a WER of 15.854%, its performance is still not good enough to achieve acceptable NLP performance. By employing a DD2 language model, the accuracy can be improved to 67.107 %. This is already very good, as the performance of clean text input from the test data is only 7.040 % better. Also, the DBi language model performs similarly well, especially the SSC-GP4-DBi and SSC-DBi perform better than Google-DBi, which makes the use of Google's ASR obsolete here. The performance of the STN-based language models is lower but still obviously better than using general-purpose language models alone.

When performing experiments on very noisy data (see Section 5.7, Figure 5.5), one can observe a huge decrease in NLP performance. When performing *SemaPred* recognition on Google's text output, the accuracy is only 11.221%, which is far from acceptable. The best-performing language model is STN in

combination with Google as an acoustic model (25.853 %). SSC-DD2 is performing second best (23.872%). Then, the other character-based STN models follow. STN models provide stability to the structure of the output hypothesis. All character-based STN models perform better than all character-based DBi models, which indicates that the provided stability of the STN models obviously helps when the speech input is very noisy. For very noisy speech input, we suggest using the STN language models with Google's ASR or the faster SSC-DD2.

The results of the experiments on *SemaPred* interpretation accuracy on clean audio data (see Section 5.6) indicate that the *SemaPred Interpreter* (SPI) performs very well (99.56 %) when it processes Gold Standard *SemaPreds* from the test set. When processing test sentences and recognizing *SemaPreds* using the *SemaPred Recognizer* the accuracy of 79.538 % is still very high. When processing real audio data from the test set using our best approach (SSC-DD2), the performance drops only marginally to 73.157 %. Compared to Google (56.436 %), this is a large improvement. For clean audio conditions, we suggest using our SSC-DD2 model.

Under very noisy audio conditions, the performance in *SemaPred* interpretation accuracy drops for all approaches (see Section 5.9). Our best approaches (SSC-DD2, Google-STN) achieve a relative accuracy improvement of 106.90 % compared to Google. We also showed that all domain-dependent language models helped to improve performance.

In this thesis, we also presented a Semantic Evaluator and a Crossmodal Corrector (CC). Our experiments (see Section 5.10) show that the use of a CC is helpful under clean audio conditions. When processing text sentences from the test set, recognizing *SemaPreds* with the *SemaPred Recognizer*, and process them with the SPI, the performance could be improved by utilizing the CC (79.538 % vs. 82.728 % *SemaPred* interpretation accuracy). This shows that the CC is able to compensate both errors from the ASR and the *SemaPred Recognizer*. The experiments performed under very noisy conditions (see Sec-

tion 5.11) indicate that the usage of the CC also improves the performance for all approaches. The best performing approach is Google-STN (36.634 %), the second-best is SSC-DD2 (35.204 %). Under noisy conditions we suggest to use the configuration Google-STN-SPI+CC for the best performance.

In our experiments, we also measured the processing time of the different approaches. The fastest acoustic model is SSC with 0.35s. When using internal language models (GP4, GP3, DD2), the response time rises to 0.461s, 0459s, and 0.436s. Compared to Google's ASR (1.271s), the response time is much shorter. One needs to consider that Google's ASR is cloud-based, and the higher response time may be caused by the network traffic. Our aim was to provide a local and faster solution, which could be achieved. The response time for the slowest system (Google-STN 2.519s) is relatively slow. For noisy environments, we suggest using the system nevertheless, if there are no time constraints. Otherwise, we suggest employing SSC-DD2.

In summary, we suggest using domain-dependent language models in any case. In this work, we show that a self-trained ASR can perform similar to cloud-based Google ASR when using general-purpose language models. When using a self-trained ASR, together with domain-dependent language models, the performance is drastically better for the ASR itself but also for connected NLP models. The performance of our NLP system is acceptable, making it a baseline model for future *SemaPred* recognition tasks. This work also showed that *SemaPreds* can be successfully recognized from speech. The SPI achieves almost 100 % *SemaPred* interpretation accuracy when working on clean input, which raises hope that it is also useful for other scenarios. We suggest employing our CC whenever time constraints make this possible. As the main outcome of this thesis, we contribute the fast (0.842s per command) ASR-to-execution chain (SSC-DD2-SPI+CC) which achieves excellent performance under clean audio conditions and acceptable performance under noisy audio conditions. We also demonstrated that the developed bidirectional processing chain can be used in simulated and real human-robot interaction scenarios.

Appendix A

Nomenclature

NLP natural language processing

NLU natural language understanding

ASR automatic speech recognition

HRI human-robot interaction

SemaPred Semantic Logic Predicates

CNN Convolutional Neural Network

NICO Neuro-Inspired COmpanion

GRU Gated Recurrent Unit

LSTM Long Short-Term Memory

RNN recurrent neural network

HMM Hidden Markov Model

CTC Connectionist Temporal Classifcation

DNN deep neural network

MFCC Mel Frequency Cepstral Coefficients

ASP Answer Set Programming

RCL Robot Control Language

CSV comma-separated values

SSC SlimSpeech Chars

SSP SlimSpeech Phonemes

WER word error rate

PER phoneme error rate

ST Sentence Template

STG Sentence Template Grammar

STN Sentence Template N-gram

SW semantic word

TB terminal bag

SPR SemaPred Recognizer

SPI SemaPred Interpreter

XML Extensible Markup Language

SE Semantic Evaluator

GP3 general-purpose trigram language model

GP4 general-purpose quadrogram language model

DD2 domain-dependent bigram language model

DBi DOCKS bigram language model

Appendix B

Complete List of Logic Declarations

```
% We work with a 8x8x8 world.
size(8).
```

```
%   Coordinates are X,Y,Z
%   Direction-relations left, right are
%      from the robots perspective
%   Direction-relation front is far away
%      from the robot, while back is near the robot
%   From perspective of the viewer, X is left, right
%   From perspective of the viewer, Y is front, back
%   From perspective of the viewer, Z is height

% Define a grid with the existing size.
grid(X,Y,Z) :- X = 0..S-1, Y = 0..S-1, Z = 0..S-1,
                        size(S).
```

```
% Each point on the board is a reference point.
ref(X,Y,Z) :- grid(X,Y,Z), Z=0.
ref(X,Y,Z) :- shape(X,Y,Z).
ref(X,Y,Z+1) :- shape(X,Y,Z).
ref(X,Y,Z) :- gripper(X,Y,Z).
```

```
% define colors
color(white;red;yellow;blue;green;gray;cyan;magenta).
```

The ASP code snippets in this section are (partially) taken from Tobergte's Bacherlor's thesis (Tobergte, 2017). We developed the ideas and conceptions behind this approach, while Tobergte performed the implementation (under our supervision).

```
% define directional positions
position(left;right;center;back;front).
```

```
% define possible shapes
shape(cube;prism).
```

```
% other
extreme(leftmost;rightmost).
```

```
has_position(X,Y,Z, position(left)) :- left(X,Y,Z).
has_position(X,Y,Z, position(right)) :- right(X,Y,Z).
has_position(X,Y,Z, position(center)) :-
                        center(X,Y,Z).
has_position(X,Y,Z, position(front)) :- front(X,Y,Z).
has_position(X,Y,Z, position(back)) :- back(X,Y,Z).
```

```
% combine has_position + has_color predicates
%    to has_attr ( + one rule for individual)
has_attribute(X,Y,Z, ATTR) :- has_color(shape(X,Y,Z),
                            color(ATTR)).
has_attribute(X,Y,Z, ATTR) :- has_position(X,Y,Z,
                        position(ATTR)).
has_attribute(X,Y,Z, individual) :- individual(X,Y,Z).
```

```
% has shape predicate
has_shape(X,Y,Z, shape(cube)) :- cube(X,Y,Z).
has_shape(X,Y,Z, shape(prism)) :- prism(X,Y,Z).
```

```
% there is a robot at approx. 0,4,0
robot(X,Y,Z) :- X = -(S/2), Y = S/2, Z = 0, size(S).
```

```
% cubes and prisms are both shapes.
shape(X,Y,Z) :- cube(X,Y,Z).
shape(X,Y,Z) :- prism(X,Y,Z).
```

```
% there are 4 corners.
corner(0,0,-1 ; 0,7,-1 ; 7,0,-1 ; 7,7,-1).
```

```
% there are 4 edges, each having 8 coordinates
edge(X,Y,Z) :- X = 0, Y = 0..S-1, Z = -1, size(S).
edge(X,Y,Z) :- X = 0..S-1, Y = 0, Z = -1, size(S).
edge(X,Y,Z) :- X = S-1, Y = 0..S-1, Z = -1, size(S).
edge(X,Y,Z) :- X = 0..S-1, Y = S-1, Z = -1, size(S).
```

```
% the board is 'below' ground level
board(X,Y,-1) :- grid(X,Y,Z), Z = 0.
```

```
% robot, shapes, corners, edges and board are
%    pointers to locations
pointer(X,Y,Z) :- robot(X,Y,Z).
pointer(X,Y,Z) :- shape(X,Y,Z).
pointer(X,Y,Z) :- corner(X,Y,Z).
pointer(X,Y,Z) :- edge(X,Y,Z).
pointer(X,Y,Z) :- board(X,Y,Z).
```

```
% corner and edge are both on the outer rim
rim(X,Y,Z) :- corner(X,Y,Z).
rim(X,Y,Z) :- edge(X,Y,Z).
```

```
% each point in the world can be associated with
%    one or more region
% regions are implemented as points, 4 for each
%    direction plus 4 for the center
% the center has 4 points, because the board size
%    is divisible by two, so it has no exact center

% right region
region(X,Y,-3) :- X = (S/2),Y = 0, size(S).
% left region
region(X,Y,-3) :- X = (S/2), Y = S-1, size(S).
% front region
region(X,Y,-3) :- X = S-1,Y = (S/2), size(S).
% back region (near robot)
region(X,Y,-3) :- X = 0, Y = (S/2), size(S).
% center region
region(X,Y,-3) :- X = (S/2), Y = (S/2), size(S).
region(X,Y,-3) :- X = (S/2)-1, Y = (S/2), size(S).
region(X,Y,-3) :- X = (S/2), Y = (S/2)-1, size(S).
region(X,Y,-3) :- X = (S/2)-1, Y = (S/2)-1, size(S).
```

```
% a group is either a stack or a row, same applies
%    for colored groups
group(X,Y,Z) :- stack(X,Y,Z).
group(X,Y,Z) :- row(X,Y,Z).

group(X,Y,Z,H,C1) :- stack(X,Y,Z,H,C1).
group(X,Y,Z,H,C1) :- row(X,Y,Z,H,C1).

group(X,Y,Z,H,C1,C2) :- stack(X,Y,Z,H,C1,C2).
group(X,Y,Z,H,C1,C2) :- row(X,Y,Z,H,C1,C2).

group(X,Y,Z,H,C1,C2,C3) :- stack(X,Y,Z,H,C1,C2,C3).
group(X,Y,Z,H,C1,C2,C3) :- row(X,Y,Z,H,C1,C2,C3).
```

```
% a colored stack is also a simple stack
stack(X,Y,Z) :- stack(X,Y,Z,H,C1).
stack(X,Y,Z) :- stack(X,Y,Z,H,C1,C2).
stack(X,Y,Z) :- stack(X,Y,Z,H,C1,C2,C3).
```

```
% the lower block in a stack with height 2 is also
%    part of the stack
stack(X,Y,Z-1) :- stack(X,Y,Z,2,C1).
stack(X,Y,Z-1) :- stack(X,Y,Z,2,C1,C2).
stack(X,Y,Z-1) :- stack(X,Y,Z,2,C1,C2,C3).
```

```
% A stack has a top coordinate X,Y,Z, and a height H
%    and has to start on the ground
% A stack can also be part of a bigger stack
stack(X,Y,Z,H) :- shape(X,Y,Z), shape(X,Y,B), Z > B,
            H = 1+Z-B,
            ground(X,Y,B),
            shape(X,Y,M1), Z >= M1, M1 >= B.
```

```
% A colored stack has a top coordinate X,Y,Z,
%    a height H, and a number of Colors C1 .. CN
% no other colors may occur inside the stack,
%    and the stack has to start on the ground
stack(X,Y,Z,H,C1) :- shape(X,Y,Z), shape(X,Y,B),
            Z > B, H = 1+Z-B,
            ground(X,Y,B),
            color(C1), has_color(shape(X,Y,M1),
            color(C1)), Z >= M1, M1 >= B,
            #false: shape(X,Y,Z1), Z >= Z1,
                  Z1 >= B,
                  has_color(shape(X,Y,Z1),
                        color(CX)),
                  CX != C1.
```

```
stack(X,Y,Z,H,C1,C2) :- shape(X,Y,Z), shape(X,Y,B),
            Z > B, H = 1+Z-B,
            ground(X,Y,B),
            color(C1), has_color(shape(X,Y,M1),
                  color(C1)),
            Z >= M1, M1 >= B,
            color(C2), has_color(shape(X,Y,M2),
                  color(C2)),
            Z >= M2, M2 >= B,
            C1<C2,
            #false: shape(X,Y,Z1), Z >= Z1,
                  Z1 >= B,
                  has_color(shape(X,Y,Z1),
                        color(CX)),
                  CX != C1, CX != C2.
```

```
stack(X,Y,Z,H,C1,C2,C3) :- shape(X,Y,Z),
            shape(X,Y,B),
            Z > B, H = 1+Z-B,
            ground(X,Y,B),
            color(C1), has_color(shape(X,Y,M1),
                  color(C1)),
            Z >= M1, M1 >= B,
            color(C2), has_color(shape(X,Y,M2),
                  color(C2)),
            Z >= M2, M2 >= B,
            color(C3), has_color(shape(X,Y,M3),
                  color(C3)),
            Z >= M3, M3 >= B,
            C1<C2,C2<C3,
            #false: shape(X,Y,Z1), Z >= Z1,
                  Z1 >= B,
                  has_color(shape(X,Y,Z1),
                        color(CX)),
                  CX != C1, CX != C2,
                  CX != C3.
```

```
% the simplest row is two adjacent shapes
row(X,Y,Z) :- shape(X,Y,Z), adjacent(shape(X,Y,Z),
      shape(A,B,C)).
```

```
% a row along the x-axis with length H and color C1
row(X,Y,Z,H,C1) :- shape(X,Y,Z), shape(A,Y,Z),
      X > A, H = 1+X-A, color(C1),
      has_color(shape(M1,Y,Z), color(C1)),
      X >= M1, M1 >= A,
      #false: free(M4,Y,Z), X >= M4, M4 >= A;
      #false: shape(M2,Y,Z), X >= M2, M2 >= A,
            has_color(shape(M2,Y,Z), color(CX)),
            CX != C1.
```

```
% a row along the y-axis with length H and color C1
row(X,Y,Z,H,C1) :- shape(X,Y,Z), shape(X,B,Z),
      Y > B, H = 1+Y-B, color(C1),
      has_color(shape(X,M1,Z), color(C1)),
      Y >= M1, M1 >= B,
      #false: free(X,M4,Z), Y >= M4, M4 >= B;
      #false: shape(X,M2,Z), Y >= M2, M2 >= B,
            has_color(shape(X,M2,Z), color(CX)),
            CX != C1.
```

```
% dual-colored row along the x-axis with length H
%    and colors C1 and C2
row(X,Y,Z,H,C1,C2) :- shape(X,Y,Z), shape(A,Y,Z),
      X > A, H = 1+X-A, color(C1),
      has_color(shape(M1,Y,Z), color(C1)),
      X >= M1, M1 >= A, color(C2),
      has_color(shape(M2,Y,Z), color(C2)),
      X >= M2, M2 >= A, C1<C2,
      #false: free(M4,Y,Z), X >= M4, M4 >= A;
      #false: shape(M3,Y,Z), X >= M3, M3 >= A,
            has_color(shape(M3,Y,Z), color(CX)),
            CX != C1, CX != C2.
```

```
% dual-colored row along the y-axis with length H
%    and colors C1 and C2
row(X,Y,Z,H,C1,C2) :- shape(X,Y,Z), shape(X,B,Z),
      Y > B, H = 1+Y-B, color(C1),
      has_color(shape(X,M1,Z), color(C1)),
      Y >= M1, M1 >= B, color(C2),
      has_color(shape(X,M2,Z), color(C2)),
      Y >= M2, M2 >= B, C1<C2,
      #false: free(X,M4,Z), Y >= M4, M4 >= B;
      #false: shape(X,M3,Z), Y >= M3, M3 >= B,
           has_color(shape(X,M3,Z), color(CX)),
           CX != C1, CX != C2.
```

```
% triple-colored row along the x-axis with length H
%    and colors C1,C2 and C3
row(X,Y,Z,H,C1,C2,C3) :- shape(X,Y,Z), shape(A,Y,Z),
      X > A, H = 1+X-A, color(C1),
      has_color(shape(M1,Y,Z), color(C1)),
      X >= M1, M1 >= A, color(C2),
      has_color(shape(M2,Y,Z), color(C2)),
      X >= M2, M2 >= A, color(C3),
      has_color(shape(M3,Y,Z), color(C3)),
      X >= M3, M3 >= A, C1<C2,C2<C3,
      #false: free(M5,Y,Z), X >= M5, M5 >= A;
      #false: shape(M4,Y,Z), X >= M4, M4 >= A,
           has_color(shape(M4,Y,Z), color(CX)),
           CX != C1, CX != C2, CX != C3.
```

```
% triple-colored row along the y-axis with length H
%    and colors C1,C2 and C3
row(X,Y,Z,H,C1,C2,C3) :- shape(X,Y,Z), shape(X,B,Z),
      Y > B, H = 1+Y-B, color(C1),
      has_color(shape(X,M1,Z), color(C1)),
      Y >= M1, M1 >= B, color(C2),
      has_color(shape(X,M2,Z), color(C2)),
      Y >= M2, M2 >= B, color(C3),
      has_color(shape(X,M3,Z), color(C3)),
      Y >= M3, M3 >= B, C1<C2,C2<C3,
      #false: free(X,M5,Z), Y >= M5, M5 >= B;
      #false: shape(X,M4,Z), Y >= M4, M4 >= B,
           has_color(shape(X,M4,Z), color(CX)),
           CX != C1, CX != C2, CX != C3.
```

```
% Extra relation used for tiles, and cubes 'in' stacks
in(ref(A,B,C),ref(X,Y,Z)) :- ref(A,B,C), ref(X,Y,Z),
      A=X, B=Y, C=Z.
```

```
% Ref X,Y,Z is above A,B,C, if the altitude is
%    higher and they are on the same coordinates
%    otherwise.
above(ref(X,Y,Z),ref(A,B,C)) :- ref(X,Y,Z),
      ref(A,B,C), X = A, Y = B, Z > C.

% Special case: rim (corner+edge) has no actual
%    "level", so a reference on the ground is above
%    them if its X and Y coordinates match.
above(ref(X,Y,Z), ref(A,B,C)) :- ref(X,Y,Z),
      rim(A,B,C), X = A, Y = B.

% Above rule for regions, center is special
above(ref(X,Y,Z), ref(A,B,C)) :- region(A,B,C),
      shape(X,Y,Z),
   horizontaldistance(X,Y,Z,A,B,C,Dist), Dist <= 0,
   center(A,B,C).
above(ref(X,Y,Z), ref(A,B,C)) :- region(A,B,C),
      right(A,B,C), right(X,Y,Z).
above(ref(X,Y,Z), ref(A,B,C)) :- region(A,B,C),
      left(A,B,C), left(X,Y,Z).
above(ref(X,Y,Z), ref(A,B,C)) :- region(A,B,C),
      front(A,B,C), front(X,Y,Z).
above(ref(X,Y,Z), ref(A,B,C)) :- region(A,B,C),
      back(A,B,C), back(X,Y,Z).
```

```
% Reference X,Y,Z is ontopof reference A,B,C, if it
%    is directly on top of it.
ontopof(ref(X,Y,Z),ref(A,B,C)) :- ref(X,Y,Z),
      ref(A,B,C), X = A, Y = B, Z = C + 1.
% special case for board as it is no reference
%    point.
ontopof(ref(X,Y,Z),ref(A,B,C)) :- ref(X,Y,Z),
      board(A,B,C), X = A, Y = B, Z = C + 1.
% Same for shape and shape
ontopof(shape(X,Y,Z),shape(A,B,C)) :- shape(X,Y,Z),
      shape(A,B,C), X = A, Y = B, Z = C + 1.
% And for ref and shape
ontopof(ref(X,Y,Z),shape(A,B,C)) :- ref(X,Y,Z),
      shape(A,B,C), X = A, Y = B, Z = C + 1.
```

```
% Below is needed only for the gripper so far, so for
%    performance increase we can narrow down the
%    solution
below(ref(X,Y,Z), ref(A,B,C)) :- gripper(A,B,C),
      ref(X,Y,Z), X=A, Y=B, Z < C.
```

```
% center region is defined to match 4 region points
%    in the center
center(X,Y,Z) :- region(X,Y,Z), X <= (S/2),
      X >= (S/2)-1, Y <= (S/2),
      Y >= (S/2)-1, size(S).
```

```
% Reference X,Y,Z is left of reference A,B,C, if the
%    X coordinate of X,Y,Z is smaller than A,B,C's,
%    which in this case is Y > B, because the
%    perspective is rotated 3 hours clockwise
left(ref(X,Y,Z), ref(A,B,C)) :- ref(X,Y,Z),
      ref(A,B,C), Y = B+1, X = A.
```

```
% A pointer (excluding edge) is (on the) left if it
%    is on the left half of the world,
%    which in this case is Y >= boardsize/2 because
%    the perspective is rotated 3 hours clockwise
left(X,Y,Z) :-
   pointer(X,Y,Z), size(S), Y >= (S/2),
   #false: edge(X,Y,Z).
% A region / edge is (on the) left if it is located
%    on the leftmost location possible on board,
%    given the boardsize S.
left(X,Y,Z) :- edge(X,Y,Z), size(S), Y = S-1.
left(X,Y,Z) :- region(X,Y,Z), size(S), Y = S-1.
```

```
% Reference X,Y,Z is <Dist> tiles leftward to
%    reference A,B,C if B+Dist = Y and X and A match.
leftward(ref(X,Y,Z), ref(A,B,C)) :-
      leftward(ref(X,Y,Z), ref(A,B,C), 1).
leftward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), ref(A,B,C), Y = B+Dist, X = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
leftward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), rim(A,B,C), Y = B+Dist, X = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
```

```
% Same applies for right / rightward
right(ref(X,Y,Z), ref(A,B,C)) :- ref(X,Y,Z),
      ref(A,B,C), Y+1 = B, X = A.
right(X,Y,Z) :-
   pointer(X,Y,Z), size(S), Y <= (S/2),
   #false: edge(X,Y,Z).
right(X,Y,Z) :- edge(X,Y,Z), size(S), Y = 0.
right(X,Y,Z) :- region(X,Y,Z), size(S), Y = 0.
```

```
rightward(ref(X,Y,Z), ref(A,B,C)) :-
      rightward(ref(X,Y,Z), ref(A,B,C), 1).
rightward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), ref(A,B,C), Y = B-Dist, X = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
rightward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), rim(A,B,C), Y = B-Dist, X = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
```

```
% Same applies for front / forward
front(ref(X,Y,Z), ref(A,B,C)) :- grid(X,Y,Z),
      grid(A,B,C), ref(X,Y,Z), ref(A,B,C),
      X = A+1, Y = B.

front(X,Y,Z) :-
   pointer(X,Y,Z), size(S), X >= (S/2),
   #false: edge(X,Y,Z).

front(X,Y,Z) :- edge(X,Y,Z), size(S), X = S-1.
front(X,Y,Z) :- region(X,Y,Z), size(S), X = S-1.
```

```
forward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), ref(A,B,C), Y = B, X-Dist = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
forward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), rim(A,B,C), Y = B, X-Dist = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
```

```
% Same applies for back / backward
back(X,Y,Z) :-
   pointer(X,Y,Z), size(S), X <= (S/2),
   #false: edge(X,Y,Z).

back(X,Y,Z) :- edge(X,Y,Z), size(S), X = 0.
back(X,Y,Z) :- region(X,Y,Z), size(S), X = 0.
```

```
backward(ref(X,Y,Z), ref(A,B,C)) :-
      backward(ref(X,Y,Z), ref(A,B,C), 1).
backward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), ref(A,B,C), Y = B, X+Dist = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
backward(ref(X,Y,Z), ref(A,B,C), Dist) :-
      ref(X,Y,Z), rim(A,B,C), Y = B, X+Dist = A,
      Dist = 1..S-1, size(S),
      #false: gripper(X,Y,Z).
```

```
% A shape with coordinates X,Y,Z is individual if
%    there are no shapes on top of it and it is on
%    ground level
individual(X,Y,Z) :- shape(X,Y,Z), ground(X,Y,Z),
   #false: ontopof(shape(A,B,C), shape(X,Y,Z)).
```

```
% A shape is on the ground if its altitude is 0.
ground(X,Y,Z) :- shape(X,Y,Z), Z=0.
```

```
% Shapes X,Y,Z and A,B,C are adjacent, if the
%    distance between them is exactly 1, and they
%    are on the same plain.
adjacent(shape(X,Y,Z), shape(A,B,C)) :-
      shape(X,Y,Z), shape(A,B,C),
      |X - A| = 1, Y = B, Z = C.
adjacent(shape(X,Y,Z), shape(A,B,C)) :-
      shape(X,Y,Z), shape(A,B,C), X = A,
      |Y - B| = 1, Z = C.
adjacent(ref(X,Y,Z), ref(A,B,C)) :-
      ref(X,Y,Z), edge(A,B,C), X = A,
      |Y - B| = 1.
adjacent(ref(X,Y,Z), ref(A,B,C)) :-
      ref(X,Y,Z), edge(A,B,C),
      |X - A| = 1, Y = B.
```

```
% The horizontal distance between X,Y,Z and A,B,C
%    is calculated via the 2D euclidean distance,
%    but without the squareroot
horizontaldistance(X,Y,Z,A,B,C,Dist) :-
      pointer(X,Y,Z), pointer(A,B,C),
      Dist = (X - A) ** 2 + (Y - B) ** 2.
horizontaldistance(X,Y,Z,A,B,C,Dist) :-
      pointer(X,Y,Z), region(A,B,C),
      Dist = (X - A) ** 2 + (Y - B) ** 2.
```

```
% Pointers / regions X,Y,Z and A,B,C are different
%    if they are not the same
different(X,Y,Z,A,B,C) :- pointer(X,Y,Z),
      pointer(A,B,C),
      #false: X == A, Y == B, Z == C.
different(X,Y,Z,A,B,C) :- region(X,Y,Z),
      region(A,B,C),
      #false: X == A, Y == B, Z == C.
```

```
% A point is free, if there is no shape in it.
free(X,Y,Z) :- grid(X,Y,Z), #false: shape(X,Y,Z).
free(ref(X,Y,Z)) :- ref(X,Y,Z), #false: shape(X,Y,Z).
```

```
% A point is not_blocked, if the point itself, or
%    the point directly on top of it is free.
not_blocked(ref(X,Y,Z)) :- ref(X,Y,Z),
      free(ref(X,Y,Z)).
not_blocked(ref(X,Y,Z)) :- ref(X,Y,Z),
      free(A,B,C), C=Z+1, X=A, Y=B.
% or if is the topmost location possible in the grid.
not_blocked(ref(X,Y,Z)) :- ref(X,Y,Z),
      Z=S-1, size(S).
```

```
% A shape X,Y,Z can be moved to a point A,B,C, if
%    X,Y,Z is not blocked and A,B,C is free.
% A,B,C has to be on the ground,
move(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
      ref(A,B,C),
      not_blocked(ref(X,Y,Z)), free(ref(A,B,C)),
      C = 0.
% A shape can be moved into the gripper
move(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
      ref(A,B,C),
      not_blocked(ref(X,Y,Z)), free(ref(A,B,C)),
      gripper(A,B,C).
% or on top of another shape, but not on top of a
%    prism
% the shape cannot be put on top of itself.
move(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
      ref(A,B,C),
   not_blocked(ref(X,Y,Z)), free(ref(A,B,C)),
   ontopof(ref(A,B,C), shape(P,Q,R)),
   #false: P == X, Q == Y, R == Z;
   #false: prism(P,Q,R).
#show.
```

```
% A shape can be moved to a point X,Y,Z in general
%    only if it is free.
% Furthermore, there are 3 different cases for a
%    valid move:
%   1. move onto the ground:
valid_pos_for_move(ref(X,Y,Z)) :- ref(X,Y,Z),
      free(ref(X,Y,Z)), Z = 0.
%   2. move into the gripper:
valid_pos_for_move(ref(X,Y,Z)) :- ref(X,Y,Z),
      free(ref(X,Y,Z)), gripper(X,Y,Z).
%   3. on top of another shape, but not on top of a
%    prism:
valid_pos_for_move(ref(X,Y,Z)) :- ref(X,Y,Z),
      free(ref(X,Y,Z)), ontopof(ref(X,Y,Z),
      shape(P,Q,R)), #false: prism(P,Q,R).
```

```
% normal drop: move from gripper onto free position.
drop(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
      gripper(X,Y,Z), not_blocked(ref(X,Y,Z)),
   ref(A,B,C), valid_pos_for_move(ref(A,B,C)),
   shape_in_gripper(true),
   % but dropping the object ontop of itself is
   %    not possible:
   #false: X == A, Y == B, Z+1 == C.
% special drop: if no block in gripper, then drop
%    should work with an object from anywhere
drop(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
      ref(A,B,C),
   not_blocked(ref(X,Y,Z)), free(ref(A,B,C)),
   shape_in_gripper(false),
   % but dropping the object ontop of itself is
   %    not possible:
   #false: X == A, Y == B, Z+1 == C.
```

```
% normal take: from anywhere to gripper.
take(shape(X,Y,Z), ref(A,B,C)) :- shape(X,Y,Z),
      ref(A,B,C),
      not_blocked(ref(X,Y,Z)), free(ref(A,B,C)),
      gripper(A,B,C).
```

```
shape_in_gripper(true) :- gripper(X1,Y1,Z1),
      shape(X2,Y2,Z2),
      X1 == X2, Y1 == Y2, Z1 == Z2.
shape_in_gripper(false) :-
      not shape_in_gripper(true).
#show shape_in_gripper/1.
```

Appendix C

Complete Simulation Results

Table C.1: Word Error Rate on the clean *Knowledge Technology Train Robots* data set.

Internal Acoustic Model	Internal Language Model	External Language Model	Word Error Rate
SSP	-	STG	25.134 %
SSC	-	STG	23.544 %
SSC	GP3	STG	22.847 %
SSC	GP4	STG	22.731 %
Google	-	STG	22.186 %
SSC	-	-	21.275 %
Google	-	-	15.854 %
SSC	GP3	-	15.595 %
SSC	GP4	-	14.755 %
SSP	-	STN	12.755 %
SSC	GP4	STN	11.325 %
SSC	GP3	STN	10.897 %
SSC	-	STN	10.406 %
SSP	-	DBi	10.182 %
Google	-	STN	9.512 %
SSC	-	DBi	4.823 %
SSC	GP3	DBi	4.26 %
SSC	GP4	DBi	4.252 %
Google	-	DBi	4.216 %
SSC	DD2	-	3.85 %

Table C.2: *SemaPred* recognition accuracy on the clean *Knowledge Technology Train Robots* data set.

Internal Acoustic Model	Internal Language Model	External Language Model	SemaPred Recognition Accuracy
SSP	-	STG	38.834 %
SSC	GP3	STG	40.594 %
SSC	GP4	STG	40.594 %
SSC	-	STG	40.704 %
Google	-	STG	41.034 %
SSC	-	-	42.354 %
Google	-	-	48.625 %
Google	-	STN	49.395 %
SSC	GP3	-	49.945 %
SSC	GP4	-	50.385 %
SSP	-	STN	52.475 %
SSC	-	STN	54.895 %
SSC	GP3	STN	56.546 %
SSC	GP4	STN	56.546 %
Google	-	DBi	57.096 %
SSP	-	DBi	57.976 %
SSC	-	DBi	63.696 %
SSC	GP3	DBi	63.696 %
SSC	GP4	DBi	64.796 %
SSC	DD2	-	67.107 %
Gold Standard Test Sentences			74.147 %

Table C.3: *SemaPred* interpretation accuracy on the clean *Knowledge Technology Train Robots* data set.

Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor	SemaPred Interpretation Accuracy
SSP	-	STG	SPI	45.875 %
SSC	GP3	STG	SPI	49.505 %
SSC	GP4	STG	SPI	49.615 %
SSC	-	STG	SPI	49.835 %
Google	-	STG	SPI	50.055 %
SSC	-	-	SPI	50.165 %
Google	-	STN	SPI	56.216 %
Google	-	-	SPI	56.436 %
SSC	GP3	-	SPI	56.766 %
SSP	-	STN	SPI	56.766 %
SSC	GP4	-	SPI	56.986 %
SSC	-	STN	SPI	59.846 %
SSC	GP3	STN	SPI	61.716 %
SSC	GP4	STN	SPI	62.156 %
Google	-	DBi	SPI	64.466 %
SSP	-	DBi	SPI	65.017 %
SSC	-	DBi	SPI	70.077 %
SSC	GP3	DBi	SPI	70.077 %
SSC	GP4	DBi	SPI	70.957 %
SSC	DD2	-	SPI	73.157 %
Gold Standard Test Sentences			SPI	79.538 %
Gold Standard Test SemaPreds			SPI	99.56 %

Table C.4: *SemaPred* interpretation accuracy on the clean *Knowledge Technology Train Robots* data set.

Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor	SemaPred Interpretation Accuracy
SSP	-	STG	SPI+CC	49.615 %
SSC	-	-	SPI+CC	52.365 %
Google	-	STG	SPI+CC	53.245 %
SSC	GP3	STG	SPI+CC	53.355 %
SSC	-	STG	SPI+CC	53.575 %
SSC	GP4	STG	SPI+CC	53.575 %
SSC	GP3	-	SPI+CC	58.636 %
SSC	GP4	-	SPI+CC	58.966 %
Google	-	-	SPI+CC	59.406 %
Google	-	STN	SPI+CC	61.386 %
SSP	-	STN	SPI+CC	62.046 %
SSC	-	STN	SPI+CC	66.007 %
SSC	GP3	STN	SPI+CC	66.997 %
SSC	GP4	STN	SPI+CC	67.767 %
Google	-	DBi	SPI+CC	68.977 %
SSP	-	DBi	SPI+CC	70.407 %
SSC	-	DBi	SPI+CC	74.917 %
SSC	GP3	DBi	SPI+CC	75.138 %
SSC	GP4	DBi	SPI+CC	75.798 %
SSC	DD2	-	SPI+CC	77.228 %
Gold Standard Test Sentences			SPI+CC	82.728 %
Gold Standard Test SemaPreds			SPI+CC	99.56 %

Table C.5: Word Error Rate on the noisy *Knowledge Technology Train Robots* data set.

Internal Acoustic Model	Internal Language Model	External Language Model	Word Error Rate
SSC	-	-	63.335 %
SSC	GP3	-	53.626 %
SSC	GP4	-	52.992 %
SSP	-	STG	49.929 %
Google	-	-	49.902 %
SSP	-	STN	46.338 %
SSC	-	STG	41.05 %
SSP	-	DBi	40.372 %
SSC	GP3	STG	39.443 %
SSC	GP4	STG	39.05 %
Google	-	STG	38.067 %
SSC	-	STN	36.075 %
SSC	GP3	STN	33.985 %
SSC	GP4	STN	33.708 %
Google	-	STN	33.128 %
SSC	GP4	DBi	29.475 %
SSC	GP3	DBi	29.278 %
Google	-	DBi	29.153 %
SSC	-	DBi	28.51 %
SSC	DD2	-	27.349 %

Table C.6: *SemaPred* recognition accuracy on the noisy *Knowledge Technology Train Robots* data set.

Internal Acoustic Model	Internal Language Model	External Language Model	SemaPred Recognition Accuracy
SSC	-	-	4.95 %
SSC	GP3	-	8.801 %
SSC	GP4	-	9.461 %
Google	-	-	11.111 %
SSP	-	STG	11.111 %
SSP	-	STN	14.961 %
SSC	-	STG	16.172 %
SSP	-	DBi	16.502 %
SSC	GP3	STG	18.152 %
SSC	GP4	STG	18.592 %
Google	-	STG	21.012 %
Google	-	DBi	21.122 %
SSC	GP3	DBi	21.782 %
SSC	GP4	DBi	21.892 %
SSC	GP3	STN	22.552 %
SSC	-	STN	22.992 %
SSC	GP4	STN	22.992 %
SSC	-	DBi	23.102 %
SSC	DD2	-	26.183 %
Google	-	STN	26.733 %
Gold Standard Test Sentences			74.147 %

Table C.7: *SemaPred* interpretation accuracy on the noisy *Knowledge Technology Train Robots* data set.

Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor	SemaPred Interpretation Accuracy
SSC	-	-	SPI	8.801 %
SSC	GP3	-	SPI	13.091 %
SSC	GP4	-	SPI	14.411 %
Google	-	-	SPI	15.622 %
SSP	-	STG	SPI	15.622 %
SSP	-	STN	SPI	17.932 %
SSP	-	DBi	SPI	19.472 %
SSC	-	STG	SPI	21.672 %
SSC	GP3	STG	SPI	25.193 %
SSC	GP4	STG	SPI	25.743 %
SSC	-	STN	SPI	26.733 %
SSC	GP3	STN	SPI	27.283 %
SSC	GP4	DBi	SPI	27.283 %
SSC	-	DBi	SPI	27.613 %
SSC	GP3	DBi	SPI	27.613 %
Google	-	DBi	SPI	27.613 %
Google	-	STG	SPI	27.723 %
SSC	GP4	STN	SPI	28.493 %
SSC	DD2	-	SPI	32.233 %
Google	-	STN	SPI	32.233 %
Gold Standard Test Sentences			SPI	79.538 %
Gold Standard Test SemaPreds			SPI	99.56 %

Table C.8: *SemaPred* interpretation accuracy on the noisy *Knowledge Technology Train Robots* data set.

Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor	SemaPred Interpretation Accuracy
SSC	-	-	SPI+CC	9.901 %
SSC	GP3	-	SPI+CC	13.861 %
SSC	GP4	-	SPI+CC	15.512 %
Google	-	-	SPI+CC	15.952 %
SSP	-	STG	SPI+CC	18.812 %
SSP	-	STN	SPI+CC	21.452 %
SSP	-	DBi	SPI+CC	23.542 %
SSC	-	STG	SPI+CC	26.623 %
SSC	GP3	STG	SPI+CC	29.813 %
SSC	GP4	STG	SPI+CC	30.143 %
SSC	-	STN	SPI+CC	31.573 %
SSC	GP3	STN	SPI+CC	31.793 %
SSC	GP4	DBi	SPI+CC	31.793 %
Google	-	STG	SPI+CC	31.903 %
SSC	GP4	STN	SPI+CC	32.233 %
SSC	GP3	DBi	SPI+CC	32.343 %
SSC	-	DBi	SPI+CC	32.563 %
Google	-	DBi	SPI+CC	32.563 %
SSC	DD2	-	SPI+CC	35.204 %
Google	-	STN	SPI+CC	36.634 %
Gold Standard Test Sentences			SPI+CC	82.728 %
Gold Standard Test SemaPreds			SPI+CC	99.56 %

Table C.9: Processing time on the clean *Knowledge Technology Train Robots* data set.

Internal Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor	Processing Time
Google	-	STN	SPI+CC	2.519s (1.271s + 0.652s + 0.014s + 0.524s + 0.058s)
SSC	GP4	STG	SPI+CC	2.256s (0.461s + 1.398s + 0.013s + 0.329s + 0.056s)
SSC	GP4	STN	SPI+CC	2.231s (0.461s + 1.407s + 0.013s + 0.262s + 0.09s)
Google	-	STG	SPI+CC	2.219s (1.271s + 0.655s + 0.014s + 0.216s + 0.062s)
Google	-	DBi	SPI+CC	2.139s (1.271s + 0.339s + 0.014s + 0.417s + 0.098s)
SSC	GP3	STG	SPI+CC	1.802s (0.459s + 1.038s + 0.013s + 0.226s + 0.066s)
SSC	GP3	STN	SPI+CC	1.792s (0.459s + 0.979s + 0.013s + 0.253s + 0.087s)
Google	-	-	SPI+CC	1.651s (1.271s + 0.017s + 0.261s + 0.102s)
SSC	-	STG	SPI+CC	1.638s (0.35s + 0.726s + 0.012s + 0.479s + 0.07s)
SSP	-	STG	SPI+CC	1.53s (0.323s + 0.649s + 0.014s + 0.475s + 0.069s)

Table C.10: Processing time on the clean *Knowledge Technology Train Robots* data set.

Internal Acoustic Model	Internal Language Model	External Language Model	SemaPred Processor	Processing Time
SSC	-	STN	SPI+CC	1.501s (0.35s + 0.727s + 0.012s + 0.282s + 0.128s)
SSP	-	DBi	SPI+CC	1.448s (0.323s + 0.435s + 0.014s + 0.565s + 0.112s)
SSP	-	STN	SPI+CC	1.323s (0.323s + 0.651s + 0.014s + 0.268s + 0.067s)
SSC	GP4	DBi	SPI+CC	1.202s (0.461s + 0.339s + 0.017s + 0.314s + 0.072s)
SSC	GP3	DBi	SPI+CC	1.166s (0.459s + 0.334s + 0.013s + 0.232s + 0.128s)
SSC	-	DBi	SPI+CC	1.056s (0.35s + 0.339s + 0.012s + 0.3s + 0.055s)
SSC	DD2	-	SPI+CC	0.96s (0.436s + 0.014s + 0.431s + 0.079s)
SSC	GP3	-	SPI+CC	0.891s (0.459s + 0.016s + 0.328s + 0.087s)
SSC	-	-	SPI+CC	0.842s (0.35s + 0.025s + 0.415s + 0.051s)
SSC	GP4	-	SPI+CC	0.785s (0.461s + 0.014s + 0.254s + 0.057s)

Publications Originated from this Thesis

Xavier Hinaut and **Johannes Twiefel**. Teach your robot your language! trainable neural parser for modelling human sentence processing: Examples for 15 languages. *IEEE Transactions on Cognitive and Developmental Systems*, 2019.

Leyuan Qu, Cornelius Weber, Egor Lakomkin, **Johannes Twiefel**, and Stefan Wermter. Combining articulatory features with end-to-end learning in speech recognition. In *International Conference on Artificial Neural Networks (ICANN)*, pages 500–510, Rhodes, Greece, 2018. Springer.

Nikhil Churamani, Paul Anton, Marc Brügger, Erik Fliewasser, Thomas Hummel, Julius Mayer, Waleed Mustafa, Hwei Geok Ng, Thi Linh Chi Nguyen, Quan Nguyen, Marcus Soll, Sebastian Springenberg, Sascha Griffiths, Stefan Heinrich, Nicolás Navarro-Guerrero, Erik Strahl, **Johannes Twiefel**, Cornelius Weber, and Stefan Wermter. The impact of personalisation on human-robot interaction in learning scenarios. In *Fifth International Conference on Human Agent Interaction (HAI)*, pages 171–180, Bielefeld, Germany, 2017. ACM.

Hwei Geok Ng, Paul Anton, Marc Brügger, Nikhil Churamani, Erik Fliewasser, Thomas Hummel, Julius Mayer, Waleed Mustafa, Thi Linh Chi Nguyen, Quan Nguyen, Marcus Soll, Sebastian Springenberg, Sascha Griffiths, Ste-

fan Heinrich, Nicolás Navarro-Guerrero, Erik Strahl, **Johannes Twiefel**, Cornelius Weber, and Stefan Wermter. Hey robot, why don't you talk to me? In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 728–731, Lisbon, Portugal, 2017. IEEE.

Johannes Twiefel, Xavier Hinaut, and Stefan Wermter. Syntactic reanalysis in language models for speech recognition. In *Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 215–220, Lisbon, Portugal, 2017. IEEE.

Marian Tietz, Tayfun Alpay, **Johannes Twiefel**, and Stefan Wermter. Semi-supervised phoneme recognition with recurrent ladder networks. In *International Conference on Artificial Neural Networks (ICANN)*, pages 3–10, Alghero, Italy, 2017. Springer.

Francisco Cruz, German Parisi, **Johannes Twiefel**, and Stefan Wermter. Multi-modal integration of dynamic audiovisual patterns for an interactive reinforcement learning scenario. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 759–766, Daejeon, Korea, 2016. IEEE.

Iris Wieser, Sibel Toprak, Andreas Grenzing, Tobias Hinz, Sayantan Auddy, Ethem Can Karaoğuz, Abhilash Chandran, Melanie Remmels, Ahmed El Shinawi, Josip Josifovski, Leena Chennuru Vankadara, Faiz Ul Wahab, Alireza M. Bahnemiri, Debasish Sahu, Stefan Heinrich, Nicolás Navarro-Guerrero, Erik Strahl, **Johannes Twiefel**, and Stefan Wermter. A robotic home assistant with memory aid functionality. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 102–115, Klagenfurt, Austria, 2016a. Springer.

Iris Wieser, Sibel Toprak, Andreas Grenzing, Tobias Hinz, Sayantan Auddy, Ethem Can Karaoğuz, Abhilash Chandran, Melanie Remmels, Ahmed El Shinawi, Josip Josifovski, Leena Chennuru Vankadara, Faiz Ul Wahab,

Alireza M. Bahnemiri, Debasish Sahu, Stefan Heinrich, Nicolás Navarro-Guerrero, Erik Strahl, **Johannes Twiefel**, and Stefan Wermter. A robotic home assistant with memory aid functionality. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Video Session*, page 369, New York, USA, 2016b. IEEE.

Xavier Hinaut and **Johannes Twiefel**. Recurrent neural network sentence parser for multiple languages with flexible meaning representations for home scenarios. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Workshop on Bio-inspired Social Robot Learning in Home Scenarios*, Daejeon, Korea, 2016.

Xavier Hinaut, **Johannes Twiefel**, and Stefan Wermter. Recurrent neural network for syntax learning with flexible predicates for robotic architectures. In *Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 150–151, Daejeon, Korea, 2016. IEEE.

Johannes Twiefel, Xavier Hinaut, Marcelo Borghetti, Erik Strahl, and Stefan Wermter. Using natural language feedback in a neuro-inspired integrated multimodal robotic architecture. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 52–57, New York, USA, 2016a. IEEE.

Johannes Twiefel, Xavier Hinaut, and Stefan Wermter. Semantic Role Labelling for Robot Instructions using Echo State Networks. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 695–700, Bruges, Belgium, 2016b. Springer.

Francisco Cruz, **Johannes Twiefel**, and Stefan Wermter. Performing a cleaning task in a simulated human-robot interaction environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Work-*

shop An Open-source Recipe for Teaching/Learning Robotics with a Simulator, Hamburg, Germany, 2015a. IEEE.

Francisco Cruz, **Johannes Twiefel**, Sven Magg, Cornelius Weber, and Stefan Wermter. Interactive Reinforcement Learning through Speech Guidance in a Domestic Scenario. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Killarney, Ireland, 2015b.

Xavier Hinaut, **Johannes Twiefel**, Maxime Petit, Peter Dominey, and Stefan Wermter. A Recurrent Neural Network for Multiple Language Acquisition: Starting with English and French. In *Conference on Neural Information Processing Systems (NIPS), Workshop on Cognitive Computation: Integrating Neural and Symbolic Approaches*, Montréal, Canada, 2015a.

Xavier Hinaut, **Johannes Twiefel**, Marcelo Borghetti, Pablo Barros, Luiza Mici, and Stefan Wermter. Humanoidly speaking – learning about the world and language with a humanoid friendly robot. In *International Joint Conference on Artificial Intelligence (IJCAI), Video Competition*, Buenos Aires, Argentina, 2015b. https://youtu.be/FpYDco3ZgkU.

Jorge Davila-Chacon, **Johannes Twiefel**, Jindong Liu, and Stefan Wermter. Improving humanoid robot speech recognition with sound source localisation. In *Artificial Neural Networks and Machine Learning (ICANN)*, pages 619–626. Springer, Hamburg, Germany, 2014.

Johannes Twiefel, Timo Baumann, Stefan Heinrich, and Stefan Wermter. Improving Domain-independent Cloud-based Speech Recognition with Domain-Dependent Phonetic Post-Processing. In *28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1529–1536, Québec City, Canada, 2014.

Bibliography

Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., et al. (2015). Deep speech 2: End-to-end speech recognition in English and Mandarin. *CoRR*, abs/1512.02595.

Baral, C. (2003). *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, Cambridge, UK.

Bastianelli, E., Castellucci, G., Croce, D., Basili, R., and Nardi, D. (2014). Effective and robust natural language understanding for human-robot interaction. In *European Conference on Artificial Intelligence (ECAI)*, pages 57–62, Prague, Czech Republic.

Bastianelli, E., Castellucci, G., Croce, D., Basili, R., and Nardi, D. (2017). Structured learning for spoken language understanding in human-robot interaction. *The International Journal of Robotics Research*, 36(5-7):660–683.

Bechet, F., Nasr, A., and Favre, B. (2014). Adapting dependency parsing to spontaneous speech for open domain spoken language understanding. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Singapore.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

Bisani, M. and Ney, H. (2008). Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Buchner, J. (2017). Synthetic speech commands: A public dataset for single-word speech recognition. Available at: `https://www.kaggle.com/jbuchner/synthetic-speech-commands-dataset/`.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

Consortium, L. D. et al. (1994). Csr-ii (wsj1) complete. *Linguistic Data Consortium, Philadelphia, USA, vol. LDC94S13A*.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *34th International Conference on Machine Learning (ICML)*, pages 933–941, Sydney, Australia. JMLR.

Dukes, K. (2013a). Semantic annotation of robotic spatial commands. In *Proceedings of the Language and Technology Conference (LTC)*, Poznan, Poland.

Dukes, K. (2013b). Train robots: A dataset for natural language human-robot spatial interaction through verbal commands. In *International Conference on Social Robotics (ICSR), Embodied Communication of Goals and Intentions Workshop*, Bristol, UK.

Dukes, K. (2014a). Contextual Semantic Parsing using Crowdsourced Spatial Descriptions. *arXiv:1405.0145 [cs]*. arXiv: 1405.0145.

Dukes, K. (2014b). Semeval-2014 task 6: Supervised semantic parsing of robotic spatial commands. In *International Workshop on Semantic Evaluation (SemEval-2014)*, page 45, Dublin, Ireland.

Fakhruldeen, H., Maheshwari, P., Lenz, A., Dailami, F., and Pipe, A. G. (2016). Human robot cooperation planner using plans embedded in objects. *International Federation of Automatic Control (IFAC) PapersOnLine*, 49(21):668–674.

Garofolo, J., Graff, D., Paul, D., and Pallett, D. (2007). Csr-i (wsj0) complete. *Linguistic Data Consortium, Philadelphia, USA, vol. LDC93S6A*.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., and Pallett, D. (1993). Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1. *NASA STI/Recon technical report n*, 93.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., and Schneider, M. (2011). Potassco: The potsdam answer set solving collection. *AI Communications*, 24(2):107–124.

Goldberg, A. (1995). *Constructions: A construction grammar approach to argument structure*. University of Chicago Press.

Graves, A., Fernandez, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *23rd International Conference on Machine Learning (ICML)*, pages 369–376, Pittsburgh, USA. ACM.

Hadian, H., Sameti, H., Povey, D., and Khudanpur, S. (2018). End-to-end speech recognition using lattice-free mmi. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 12–16, Graz, Austria.

Han, K. J., Prieto, R., Wu, K., and Ma, T. (2019). State-of-the-art speech recognition using multi-stream self-attention with dilated 1d convolutions. *arXiv preprint arXiv:1910.00716*.

Hannun, A., Case, C., Casper, J., Catanzaro, B., and et al. (2014a). Deep Speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567.

Hannun, A., Maas, A., Jurafsky, D., and Ng, A. (2014b). First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs. *arXiv preprint arXiv:1408.2873*.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hinaut, X. and Dominey, P. F. (2013). Real-time parallel processing of grammatical structure in the fronto-striatal system: a recurrent network simulation study using reservoir computing. *PloS one*, 8(2):e52946.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Honnibal, M. and Johnson, M. (2014). Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, 2:131–142.

Kennedy, J., Lemaignan, S., Montassier, C., Lavalade, P., Irfan, B., Papadopoulos, F., Senft, E., and Belpaeme, T. (2017). Child speech recognition in human-robot interaction: evaluations and recommendations. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 82–90, Vienna, Austria.

Kerzel, M., Strahl, E., Magg, S., Navarro-Guerrero, N., Heinrich, S., and Wermter, S. (2017). Niconeuro-inspired companion: A developmental humanoid robot platform for multimodal interaction. In *26th IEEE Interna-*

tional Symposium on Robot and Human Interactive Communication (RO-MAN), pages 113–120, Lisbon, Portugal. IEEE.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. ACL.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lakomkin, E., Magg, S., Weber, C., and Wermter, S. (2018). Kt-speech-crawler: Automatic dataset construction for speech recognition from youtube videos. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 90–95, Brussels, Belgium. ACL.

Lamere, P., Kwok, P., Walker, W., Gouvea, E., Singh, R., and Wolf, P. (2003). Design of the cmu sphinx-4 decoder. In *8th European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 1181–1184, Geneva, Switzerland. ISCA.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, A. and Kawahara, T. (2009). Recent development of open-source speech recognition engine julius. In *Asia-Pacific Signal and Information Processing Association, Annual Summit and Conference (APSIPA-ASC)*, pages 131–137, Sapporo, Japan. APSIPA.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics – Doklady*, 10(8):707–710.

Lifschitz, V. (1999). Action languages, answer sets, and planning. In *The Logic Programming Paradigm*, pages 357–373. Springer.

Lüscher, C., Beck, E., Irie, K., Kitza, M., Michel, W., Zeyer, A., Schlüter, R., and Ney, H. (2019). Rwth asr systems for librispeech: Hybrid vs attention-w/o data augmentation. *arXiv preprint arXiv:1905.03072*.

Manzi, A., Fiorini, L., Esposito, R., Bonaccorsi, M., Mannari, I., Dario, P., and Cavallo, F. (2017). Design of a cloud robotic system to support senior citizens: The kubo experience. *Autonomous Robots*, 41(3):699–709.

Mermelstein, P. (1976). Distance measures for speech recognition – psychological and instrumental. In *Joint Workshop on Pattern Recognition and Artificial Intelligence*, pages 374–388, Hyannis, USA.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: an ASR corpus based on public domain audio books. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, Brisbane, Australia. IEEE.

Pleva, M., Juhar, J., Cizmar, A., Hudson, C., Carruth, D. W., and Bethel, C. L. (2017). Implementing english speech interface to jaguar robot for swat training. In *IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 105–110, Herlany, Slovakia. IEEE.

Rousseau, A., Deléglise, P., and Estève, Y. (2014). Enhancing the ted-lium corpus with selected data for language modeling and more ted talks. In *Ninth International Conference on Language Resources and Evaluation (LREC)*, pages 3935–3939, Reykjavik, Iceland.

Sainath, T. N., Vinyals, O., Senior, A., and Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584, Brisbane, Australia. IEEE.

Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 338–342, Singapore.

Sak, H., Senior, A., Rao, K., and Beaufays, F. (2015). Fast and accurate recurrent neural network acoustic models for speech recognition. *arXiv preprint arXiv:1507.06947*.

Sakaguchi, K., Post, M., and Van Durme, B. (2017). Error-repair dependency parsing for ungrammatical texts. In *55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 189–195, Vancouver, Canada.

Starke, S., Hendrich, N., Magg, S., and Zhang, J. (2016). An efficient hybridization of genetic algorithms and particle swarm optimization for inverse kinematics. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1782–1789, Qingdao, China. IEEE.

Tobergte, J. (2017). Improved interpretation of robot commands via cross-modal validation with a spatial planner. *Universität Hamburg, Dept. Informatik, Bachelor's Thesis*.

Tomasello, M. (2009). *Constructing a language*. Cambridge, MA: Harvard University Press.

Twiefel, J., Baumann, T., Heinrich, S., and Wermter, S. (2014). Improving domain-independent cloud-based speech recognition with domain-dependent phonetic post-processing. In *28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1529–1535, Québec City, Canada.

Twiefel, J., Hinaut, X., Borghetti, M., Strahl, E., and Wermter, S. (2016a). Using natural language feedback in a neuro-inspired integrated multimodal robotic architecture. In *25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 52–57, New York, USA. IEEE.

Twiefel, J., Hinaut, X., and Wermter, S. (2016b). Semantic role labelling for robot instructions using echo state networks. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, Bruges, Belgium.

Twiefel, J., Hinaut, X., and Wermter, S. (2017). Syntactic reanalysis in language models for speech recognition. In *2017 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 215–220, Lisbon, Portugal. IEEE.

Wang, Y., Mohamed, A., Le, D., Liu, C., Xiao, A., Mahadeokar, J., Huang, H., Tjandra, A., Zhang, X., Zhang, F., et al. (2019). Transformer-based acoustic modeling for hybrid speech recognition. *arXiv preprint arXiv:1910.09799*.

Winograd, T. (1972). Understanding natural language. *Cognitive Psychology*, 3(1):1–191.

Yoshikawa, M., Shindo, H., and Matsumoto, Y. (2016). Joint transition-based dependency parsing and disfluency detection for automatic speech recognition texts. In *Conference on Empirical Methods in Natural Language Processing*, pages 1036–1041, Austin, USA.

Zhou, G.-B., Wu, J., Zhang, C.-L., and Zhou, Z.-H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, 13(3):226–234.

Declaration of Authorship
Erklärung der Urheberschaft

Ich versichere an Eides statt, dass ich die Dissertation im Studiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ort, Datum Unterschrift

Permission for Publication
Erklärung zur Veröffentlichung

Ich erkläre mein Einverständnis mit der Einstellung dieser Dissertation in den Bestand der Bibliothek.

Ort, Datum Unterschrift