Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Tell your robot what to do: Evaluation of natural language models for robot command processing

Erick Jesus Romero Kramer

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
M.Sc. Argentina Ortega Sainz
M.Sc. Alex Mitrevski

January 15, 2019

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Erick Jesus Romero Kramer

Acknowledgements

With this message, I would like to express my sincere and deep gratitude to my R&D advisors, Prof. Dr. Paul G. Ploeger, M.Sc. Argentina Ortega Sainz, and M.Sc. Aleksander Mitrevski for the helpful comments and indications throughout the development of this work.

I would also like to thank the B-it-bots RoboCup@Home team for the support during the research and development of the models that will be used for the GPSR competition.

Abstract

The language used by us, humans, is denominated natural language and it has been our main way of communication since the very beginning of humanity. When we want to indicate a task to another person, we use a special kind of sentences denominated natural language commands or instructions. Robotics researchers have explored different ways to replicate this form of communication in order to improve the Human Robot Interaction (HRI) and use it in the context of domestic service robots. The implementation of domestic service robots capable of understanding natural language commands provides a great improvement in the life quality of people.

In this project, we are presenting an evaluation of the state of the art of Natural Language Understanding (NLU) models and approaches for robot command processing. Additionally, a comparative analysis of chosen open source models is performed in the context of domestic service robots. The comparative analysis reflects the different features each of the models has, as well as the requirements and the process to train and use the models.

Furthermore, we evaluate the performance of the models to understand domestic service robot commands, by recognizing the actions and complementary information in them, in three use cases: GPSR category 1, GPSR category 2, and Ropod.

Contents

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**

1.1 Motivation 1

1.2 Challenges and difficulties 2

1.3 Problem statement 3

2 Background Knowledge **5**

2.1 Natural Language Processing 5

2.1.1 Part-of-speech tagging 6

2.1.2 Named entity recognition 6

2.1.3 Natural Language Understanding 7

2.2 Word embeddings 9

2.2.1 Models 9

2.3 Frame Semantics 11

2.4 Neural Networks 12

2.4.1 Recurrent Neural Networks 12

3 State of the art **15**

3.1 Surveys 15

3.1.1 Benchmarking 16

3.2 Deep learning models 17

3.2.1 Object manipulation 17

3.2.2 Navigation 19

3.2.3 Navigation and object manipulation 20

3.3 Probabilistic graphical models 22
3.3.1 Navigation and object manipulation 22
3.3.2 Navigation 25
3.3.3 Object manipulation 26
3.4 Semanic parsers 28
3.4.1 Navigation 29
3.4.2 Object manipulation 29
3.4.3 Navigation and object manipulation 29
3.5 Statistical models 30
3.6 Named entity recognition 30
3.7 Embodied Construction Grammar 31
3.8 Frame semantics 32
3.9 Limitations of the previous work 32

4 Comparative analysis **35**
4.1 Mbot 37
4.1.1 General description 37
4.1.2 Structure 37
4.1.3 System requirements 40
4.1.4 Implementation process 42
4.2 Rasa 45
4.2.1 General description 45
4.2.2 Structure 45
4.2.3 System requirements 50
4.2.4 Implementation process 50
4.3 LU4R 52
4.3.1 General description 52
4.3.2 Structure 53
4.3.3 System requirements 55
4.3.4 Implementation process 56
4.4 ECG 61
4.4.1 General description 61
4.4.2 Structure 63

4.4.3 System Requirements . . . 65
4.4.4 Implementation process . . . 65
4.5 Features . . . 67

5 Methodology **69**
5.1 Setup . . . 69
5.2 Use cases . . . 70
5.2.1 General Purpose Service Robot . . . 70
5.2.2 Ropod . . . 71
5.3 Experimental design . . . 71
5.3.1 Datasets . . . 71
5.3.2 Evaluation metrics . . . 73
5.3.3 Experiments description . . . 74
5.3.4 Assumptions . . . 74
5.3.5 Experimental procedure . . . 75
5.3.6 Technical details . . . 76

6 Solution **77**
6.1 Training and adaptation . . . 77
6.1.1 Mbot . . . 77
6.1.2 Rasa NLU . . . 79
6.1.3 ECG . . . 79

7 Results **81**
7.1 Action classification . . . 81
7.1.1 Precision . . . 82
7.1.2 Recall . . . 82
7.1.3 F1 score . . . 82
7.2 Full command recognition . . . 83
7.3 Run-time . . . 83

8 Conclusion **87**
8.1 Contributions . . . 87
8.2 Lessons learned . . . 88

8.3 Future work 88

Appendix A Acronyms **91**

References **93**

List of Figures

2.1 NLU vs. NLP. Figure source: [MacCartney [46]] 8

2.2 CBOW and Skip-gram models. Figure source: [Mikolov et al. [56]] . . 10

2.3 Figure (a) represents the structure of a RNN with a stacking approach, while (b) represents a gated-feedback approach that creates a deep architecture. The dots presents between the layers represent global reset gates. Figure source: [Chung et al. [20]] 13

2.4 Structure of a LSTM block. Figure source: [Greff et al. [34]] 13

3.1 Scheme of the first approach. Figure source: [Martins et al. [49]] . . 22

3.2 Scheme of the second approach. Figure source: [Martins et al. [49]] . 22

3.3 Parse tree of the sentence *go to the kitchen that is down the hall.* Figure source: [Chung et al. [19]]. 26

3.4 HDCG vs DCG. Figures source: [Chung et al. [19]] 27

4.1 Pipeline of Mbot. Figure created based on the diagrams shown in [Martins et al. [49]] and [Martins [50]]. 39

4.2 Structure of the SLU part of LU4R. Figure adapted from: `http://sag.art.uniroma2.it/lu4r.html` 55

4.3 Extract of a semantic specification for the command *PR2, bring the soda can to the dining table.* Figure source: [Eppe et al. [28]] 62

4.4 Structure of the ECG2 framework. Figure source: [Raghuram [66]] . . 64

5.1 Lucy is our local B-it-bots Human Service Robot developed by Toyota. Image obtained from the team's Facebook page. 69

7.1 Results of the 4 NLU models for our use cases concerning single action natural language robot commands. 85

7.2 Results of the 4 NLU models for our use cases concerning multiple actions natural language robot commands. 86

List of Tables

4.1 Example of the slot filling output for the sentence ”textitdeliver a coke to peter at the living room. This table was obtained from [Martins [50]] 38
4.2 System requirements needed to train Mbot 40
4.3 System requirements needed to use Mbot 40
4.4 Features of the chosen NLU models. ☑ represents yes, ☒ represents no. 68

5.1 Experiment results table format 74

7.1 Results table for the experiments done with single action sentences dataset files. The values in blue represents the best values obtained for each metric on each dataset file. 84
7.2 Results table for the experiments done with multiple actions sentences dataset files. The values in blue represents the best values obtained for each metric on each dataset file. 84

1 Introduction

In this project, we are presenting an evaluation of the state of the art of Natural Language Understanding (NLU) models and approaches for robot command processing. Additionally, a comparative analysis of set of chosen open source models is performed in the context of domestic service robots. Furthermore, we evaluate the performance of the models to understand domestic service robot commands, by recognizing the action and complementary information in them, in three use cases: GPSR category 1, GPSR category 2, and Ropod.

1.1 Motivation

The language used by us, humans, is denominated natural language and it has been our main way of communication since the very beginning of humanity. When we want to indicate a task to another person, we use a special kind of sentences denominated natural language commands or instructions. A natural language command is composed of at least one action and a set of arguments that provide additional context to the indicated action. Robotics researchers have explored different ways to replicate this form of communication in order to improve the Human Robot Interaction (HRI) and use it in the context of domestic service robots.

But, what is exactly a domestic service robot?. "*A service robot is a robot that operates semi- or fully autonomously to perform services useful to the well being of humans and equipment*". This definition was presented by the authors of [de Almeida and Fong [25]] based on the definition given by the International Federation of

Robotics (IFR). The IFR divided service robots into two categories: for professional use and for domestic use.

The implementation of robots to perform domestic tasks has been an incredible option to improve the quality of life of people (mostly seniors and people with disabilities). But, this has been a quite challenging task. One of the problems is developing a proper interaction between the user and the robot. One way to implement this could be using a joystick or a Graphical User Interface (GUI), but doing so would add another level of complexity to our problem, which is training the user to interact with the robot in a proper way. It is because of this, that robotics researchers have focused mostly on HRI relying on natural language communication, as this does not require any training of the user because the communication with the robot would resemble the communication with another human being.

1.2 Challenges and difficulties

Developing a system capable of understanding natural language commands is not trivial. One of the major challenges is dealing with the abstraction that is present in the way people speak in normal basis. We normally ignore grammar rules, use words in different order, use new words, or in some cases we just skip some words. But, even having all of these issues, it is still possible for us to understand what the other person is saying, by taking into account the environment we are in, or the context of the current conversation based on the previous sentences used. The work here presented shows great progress towards this goal. But, there is still much to be done to reach the same level of understanding we have.

Another challenge regarding the actual implementation of a commanded task is developing the proper sequence of actions required to fulfill it. There are commands that involve primitive actions such as move, grasp, or tell, which do not require a planning to execute them. But, there are commands that require a certain sequence of actions, which are not explicitly mention. For instance, if a person says to a robot "*make me a strawberry smoothie*", the robot would need to infer that it needs to go to the kitchen, pick up a strawberry, throw it into a blender, pour milk or water into the blender, mix everything up, pour the smoothie into a glass, and deliver the smoothie to the person. To the best of our knowledge it is still not possible to do this, without taking the robot a long time to execute such commands.

1.3 Problem statement

Due to the benefits of working with natural language commands, a great number of researchers and companies have been working on systems that use Natural Language Processing (NLP) techniques to develop a communication system between humans and robots. A quick search in Google Scholar using the keywords *natural language instructions for domestic robots* can get you more than 20,000 references showing the tremendous amount of work that researchers have been doing in this field. Because of this, and the rapidly growth this field has faced, up to date comparative analysis of the state of the art models is missing. This lack of comparative analysis makes it really complicated for new robotics developer to know which approach should be followed, or which model should be selected, when trying to create a system that works with natural language commands.

With this work, we are providing a survey of the current state of the art of natural language models for robot command processing and presenting open source options that can be used to develop such systems. We are performing a comparative analysis of a selected set of open source options and evaluate their effectiveness on three use cases, GPSR category 1, GPSR category 2, and ROPOD, using standard metrics like precision, recall, F-measure, and accuracy. The major objective of this work is to be a guideline for new robotics developer to select a proper model to understand robot commands. Also, this work will be used by our local *Robocup@home* team to compete in the upcoming robotic competitions.

2

Background Knowledge

2.1 Natural Language Processing

Natural Language Processing (NLP), as defined in [Socher [70]], is the field of study that "*resides at the intersection of computer science, artificial intelligence, and linguistics*". NLP is not a new field, as some people might believe due to the popularity that voice controlled devices have recently obtained. In fact, it has been researched since 1950, with Alan Turing being one of the pioneers of the field with his work [Machinery [47]].

NLP focuses on the study of the human language, more formally known as natural language, which can be defined as a "*discrete, symbolic, and categorical signaling system*" [Socher [70]]. This system is composed of a person sending a message to an entity, which can be, for instance, a person, a program, or a robot, in order to express a specific meaning with it. The person sending the message is known as the communicator, while the entity receiving the message is known as the receptor. There are many different kinds of sentences that compose the natural language, such as questions, answers, or commands, being the latter the one that is particularly interesting for us. These sentences are used to indicate a task desired to be executed. The entity that receives the command needs to extract the action and complementary information from it in order to correctly infer the meaning and execute the task. For example the command *go to the kitchen*, would indicate a robot to perform a displacement action from its current location to the desired location

2.1.1 Part-of-speech tagging

Part-of-speech tagging (POS-tag) is a NLP task that consists of labeling words based on their corresponding "*syntactic role*" [Collobert et al. [21]] in the speech, i.e. if the word is either a noun, an adjective, a preposition, etc. In order to label the words appropriately, most of the approaches take into consideration the context of the words by looking at the words around the one being currently labeled.

2.1.2 Named entity recognition

Named Entity Recognition (NER) is a NLP task that was used for the first time in the Sixth Message Understanding Conference (MUC-6) [Grishman and Sundheim [35]] and it was used to identify the names of people, organizations, places, temporal expressions and numerical expressions [Sharnagat [69]]. Basically, the idea behind this task is to find special words, known as entities, in a sentence or text and label them based on a specific category the words could belong to.

Approaches

Some of the approaches that have been implemented in NER, based on the work presented by [Sharnagat [69]], are:

- Supervised Methods:
  - Hidden Markov Models (HMM) [Bikel et al. [9]]:
    HMM is one of the first models that were implemented to solve NER problems. It was constructed as a generative probabilistic classifier that implemented a system called "*IdentiFinder*" that assigned a single label to a word, based on its context, i.e. every single word in the text was labeled with either a class or with a labeled saying "*NOT-A-NAME*". The problem of HMM was that it failed to use the "*dependence between words*" [Chesworth et al. [18]].
  - Maximum Entropy Markov Models (MEMM) :
    MEMM is a discriminative model capable of learning weights of discrim-

  inative features used for classification. The objective of MEMM is to maximize the entropy of the data and with that generalize as much as possible. MEMM resulted a better option than HMM, but they had problems to "*factor future observations*" [Chesworth et al. [18]].

  - Support Vector Machine (SVM) [Cortes and Vapnik [22]]:
    SVM was a model developed to learn a linear hyperplane capable of separating positive examples from negative ones by "*a large margin*". SVM proved to be useful to perform NER with the work done by [McNamee and Mayfield [53]], in which they labeled words using "*8 classes, B- (Beginning) and I- (Inside) tags for person, organization, location, and misc entities*".
  - Conditional Random Fields (CRF) [Lafferty et al. [44]]:
    CRF is a statistical approach that has been used for pattern recognition and structure prediction. It was developed as a discriminative probabilistic graphical classifier capable of "*factor in future observations*" [Chesworth et al. [18]] and perform a global normalization with the aid of additional features. These made CRF more suitable for NER than the previous models here mentioned.

- Semi-supervised methods:

  - Bootstrapping based:
    This approach was developed to provide a solution to the problem of having few labeled datasets. AdaBoost, presented in [Carreras et al. [17]], was used to perform NER by labeling entities using BIO format (B- Beginning, I- Inside, and O- outside), using three binary classifiers, one for each tag.

2.1.3 Natural Language Understanding

Natural Language Understanding (NLU), also known as Natural Language Interpretation (NLI), [Mykowiecka [59]] is a sub-field of NLP that is used to ground sentences to a simpler representation that expresses their corresponding meaning. NLP is mostly used for processing application of natural language, for instance, part-of-speech tagging (POS-tag), machine translation, Named Entity Recognition (NER), etc. NLU is used in applications where the main goal is to know and understand the message conveyed in the sentences, such as sentiment analysis, dialogues, question answering, semantic parsing, command understanding, etc. This difference is better reflected in Figure 2.1. Another characteristic that differentiates between NLP from NLU systems is that NLP systems rely on a set of predefined set of keyword-based input templates to interpret a sentence. As mentioned in [Eppe et al. [28]] this approach has been implemented in most of the current commercial products, such as Apple's Siri, Microsoft's Cortana, Google Assistant, and Amazon's Alexa.

One of the first models developed to understand natural language sentences was SHRDLU. It was a computer program presented in [Winograd [76]] in which the user had a conversation with the program requesting to move objects, name collection of objects and query the state of the simple "*blocks world*". The program used hand-written rules to understand the sentences provided, which is not a reliable approach as it requires the enumeration of all possible grammar rules.

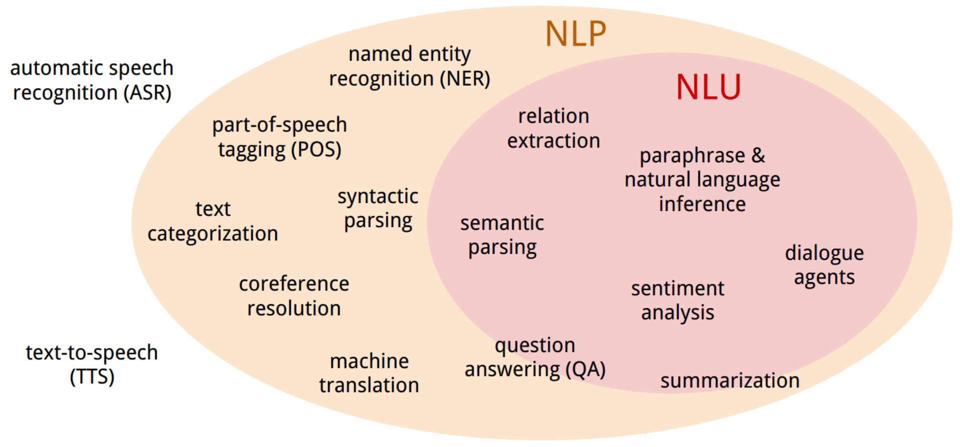


Figure 2.1: NLU vs. NLP. Figure source: [MacCartney [46]]

Almost any NLU approach has two important stages, one where the words in the sentence are analyzed identifying special words and taking into account the order of them, and another where the sentence is grounded into a form that represents the meaning of the sentence. The last stage is known as symbol grounding.

The symbol grounding problem was formally introduced in [Harnad [36]] and since then, many projects have been developed to solve it; however, it is still believed to be an open issue. One of the reasons why this has been such a complex challenge is the diversity of words and order we can use to formulate a sentence, not obeying grammar rules, or using generic words, such as this, that, etc. to refer to object in our current environment.

2.2 Word embeddings

Word embeddings have been useful to capture the semantic similarity between words and improved the understanding capacity of NLU models. It is possible to measure how similar two words are using techniques like cosine similarity. Even though it is possible to capture similar words using word embeddings, it is still necessary to adapt them to the specific domain they are going to be used to improve the performance of the models.

One of the first word representation techniques was one-hot vector representation, which used vectors containing 0s and a single 1 corresponding to the position of the word in a vocabulary. Some of the major weaknesses of this approach are that it cannot generalize well across the words in the vocabulary and it cannot capture synonyms, meaning that two similar words require a completely different vector in order to be identified. In order to improve this, new vector representations using different kind of features were developed. Word embeddings represent words using feature vectors to express the meaning of the words. The features used to represent the words could be gender, age, food, size, etc. The vector space of word embeddings could be visualized using t-SNE, which is a technique developed top visualize high dimensional data.

2.2.1 Models

Bengios' model

The work done by [Bengio et al. [7]] was one of the first neural language models that implemented word embeddings, as it was stated by [Martins [50]]. Bengio developed a Feed-Forward Neural Network (FFNN) capable of predicting words in a sequence by implementing a layer containing feature vectors, named the embedding layer.

Word2Vec

Word2Vec is one of the most well-known word embeddings models. Word2Vec was implemented following two different architectures: Continuous Bag-of-Words (CBOW) [Mikolov et al. [56]] and Skip-gram [Mikolov et al. [57]]. CBOW was developed based on the FFNN language model implemented by [Bengio et al. [7]]. It used a fixed-size context window containing words that are before and after the word currently being predicted. The words in the context window are represented using one-hot vectors and the value for the predicted word is the conditional probability calculated by averaging the context words in the projection layer and using a *softmax* activation function. Differently to the CBOW, the Skip-gram model predicts the words that appear before and after the word that is currently being analyzed. The output layer of the model contained several nodes, one for each previous and following context word. The values of these nodes are calculated using a *softmax* activation function and corresponds to the conditional probability of the context word given the current word. The structure of the two architectures can be seen in the Figure 2.2

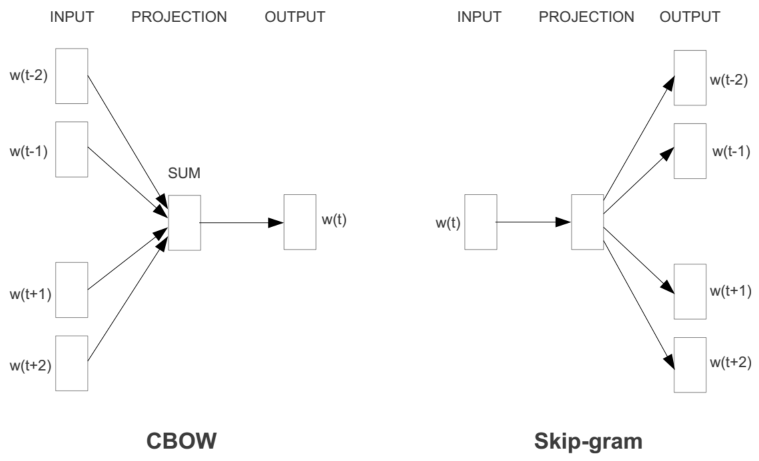


Figure 2.2: CBOW and Skip-gram models. Figure source: [Mikolov et al. [56]]

GloVe

GloVe is an unsupervised learning algorithm used to compute word representations. It was developed by Stanford [1] and formally introduced in [Pennington et al. [64]]. GloVe is a global log-bilinear regression model that includes global matrix factorization and local context window methods. The intuition behind this model is that the ratios of word-word co-occurrence probabilities contain the meaning of the words, as it was stated in [Pennington [63]]. This word representation has outperformed other word embedding approaches on word analogy, word similarity, and NER tasks.

Fasttext

FastText is an open-source library that can be used for word representation and sentence classification. It was developed by Facebook [2] and formally introduced in [Bojanowski et al. [14]]. The idea behind this model is that taking into account the morphology of words becomes important when computing word representations, overcoming the problems of having large vocabularies and rare words. The library support both CBOW and Skip-gram models. One of the major characteristics of Fasttext is that it trains much faster than previous word representations, even when very large corpora are given. Furthermore, the model can compute word representations for words that were not present in the training dataset and does not require preprocessing or supervision during training.

2.3 Frame Semantics

Frame semantics is a linguistic theory developed by [Fillmore [33]]. It has been implemented to construct relations between linguistic representations of actions and robot behaviors, as presented in [Croce [24]]. Based on this theory, a frame can be used to express a real-world situation describing actions and events. Therefore, a sentence is analyzed by recognizing all the words that "*evoke*" a frame and associating a representation that helps to express the semantics of the sentence.

[1] `https://github.com/stanfordnlp/GloVe`
[2] `https://fasttext.cc/`

2.4 Neural Networks

2.4.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were developed a long time ago, but they became popular due to the work done by Juergen Schmidhuber, Sepp Hochreiter, and Alex Graves. RNNs implement a built-in feedback loop that allows the network to behave like a "*forecasting engine*" [DeepLearning.TV [26]]. Furthermore, RNNs use a transition function to compute the values of the internal hidden states for each of the values in the input sequence, taking into account the previous hidden states values and the current input. The output of the network is computed based on the current input and the history of inputs the model has received in the past. RNNs are trained using backpropagation through time, which causes the vanishing gradient problem that was also present in Multi-Layer Perceptron (MLP) networks. To solve this problem, several techniques have been developed, but the most used one is the implementation of gates in the network [Chung et al. [20]]. Figure 2.3 shows two different RNN structures, the first one using the common stacking approach without gates, and the second one shows a RNN with a gated-feedback approach. One of the main advantages of RNNs is that they can receive a sequence of values that can change through time as input and produce a single, or a sequence of values as output.

Long-Short Term Memory

Long-Short Term Memory (LSTM) networks are RNNs that contains LSTM blocks. The elements contained in each LSTM block are "*a memory cell, an input gate, a forget gate, an output gate*". They were developed by [Hochreiter and Schmidhuber [37]] to improve the learning of long-term dependencies. As defined in [Chung et al. [20]], each of the elements in the LSTM block plays an important role: The memory cell stores the memory content of the LSTM unit, the input gate controls the amount of new content that is memorized, while the forget gate regulates the amount of old content that is forgotten, and the output gate helps to regulate the exposure of the memory content. The memory content gets updated by performing a weighted sum of the new content and the previous memory content restricted by

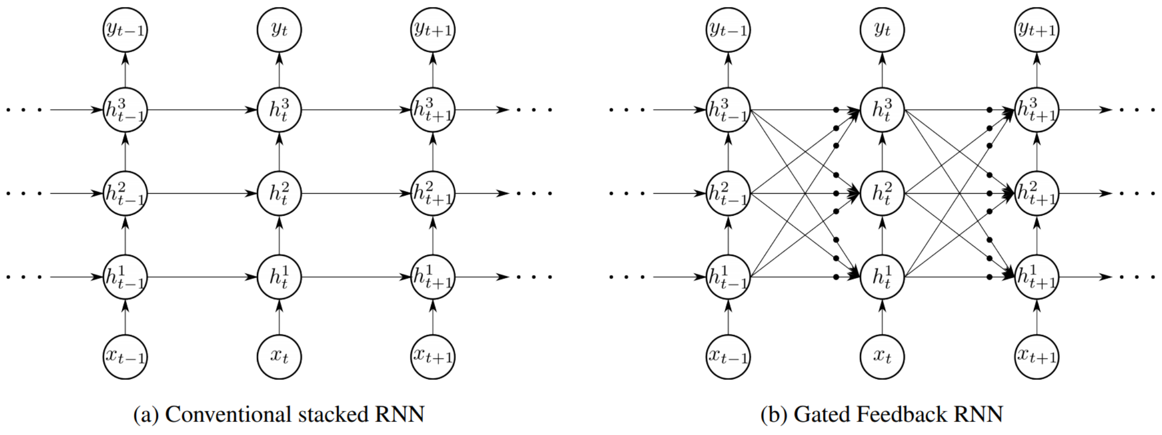


Figure 2.3: Figure (a) represents the structure of a RNN with a stacking approach, while (b) represents a gated-feedback approach that creates a deep architecture. The dots presents between the layers represent global reset gates. Figure source: [Chung et al. [20]]

the input and forget gates. The structure of a LSTM block can be appreciated in Figure 2.4.

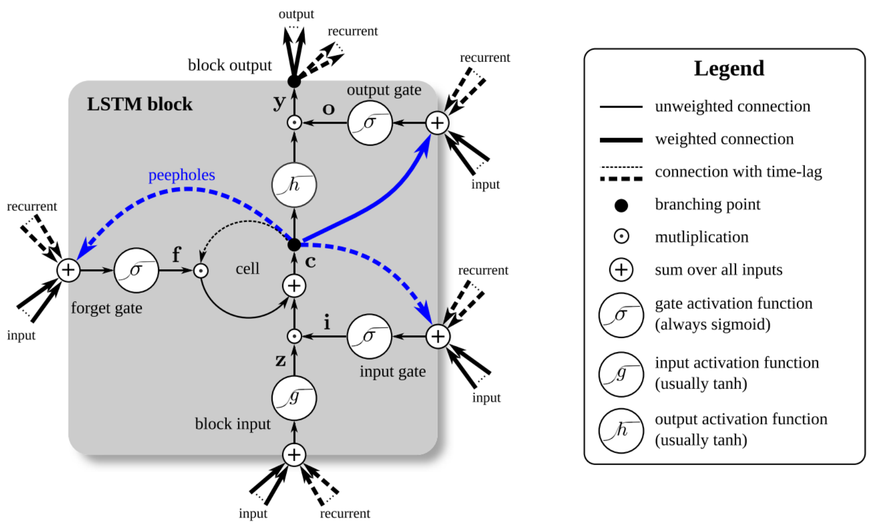


Figure 2.4: Structure of a LSTM block. Figure source: [Greff et al. [34]]

‘

3

State of the art

NLP has been a widely studied field with constant development of new models that improve the processing and understanding capabilities of natural language sentences. In the field of AI, using natural language commands to control an agent has been a long-standing goal. As it was mentioned in [Bisk et al. [11]], the continuous advances in the fields of robotics, language, and vision have been quite important in the development of better systems capable of understanding natural language instructions and performing the requested actions. Most recently, researchers have been working in the development of systems capable of dealing with the multiple levels of abstractions that are normally present in the way humans speak. The idea behind this is that identifying linguistic abstraction can lead to an improvement in the interpretation of the sentences. Furthermore, it would remove certain constraints in the instruction the robot can understand.

With this work, we want to provide an overview of the current state of the art in the field of natural language understanding in the context of commands used to indicate tasks to robots in a domestic service environment.

3.1 Surveys

The work done by [Wiriyathammabhum et al. [77]] presented a well-explained introduction of the integration of computer vision techniques and NLP models for multimedia and robotics application. The authors focused mostly on creating an unified view of both fields. They stated that the use of NLP techniques can be useful for robots to provide more control and improve the accuracy of the reasoning when "*low-level*" computer vision processes are executed. Additional to the collection of multiple approaches done in both fields, the authors presented an approach that combines both vision and language using distributional semantics. The authors proposed possible future research directions, such as event understanding and testing the integration of vision and language.

The authors of [Liu and Zhang [45]] presented several methodologies that have been implemented for HRI using natural language. They covered areas concerning NLU, natural language-based execution plan generation, and knowledge world mapping. A detailed analysis of the methods implemented in each of the areas and their applications was presented including advantages and disadvantages of them. As future work, it was proposed to look into the cognitive process of humans when using natural language instructions and to use information from the web to reduce the cost of learning.

A review of deep learning methods that have been applied in NLP was presented in [Otter et al. [61]]. The authors explained the benefits using deep learning approaches provided to the field, in comparison to the rule-based models and manually defined features that were used in the past. The work focused on the areas of language modeling, morphology, parsing, and semantics. Additionally, they presented the state of the art models for information extraction, text classification, summarization, question answering, machine translation, and image and video captioning. The thing that was notorious in most of the models presented was the use of word embeddings to represent the words in the vocabulary and that modified versions of RNNs are the type of neural networks that have obtained better results for the different tasks of NLP.

3.1.1 Benchmarking

A "*functional benchmark*", named Functional Benchmark on Speech Understanding (FBM3), was presented in [Vanzo et al. [75]]. Their goal was to measure and evaluate the performance of different architectures in the context of service robots operating in a home environment and on service robot competitions. The architectures evaluated were expected to interpret natural language robot commands, identifying the action behind the commands and the complementary information. The commands were selected from a limited set of semantic frames, such as motion, searching, taking, and bringing. The format of the output expected from the models, named Command Frame Representation (CFR), was inspired by frame semantics 2.3.

The analysis done in this benchmark focused more on comparing different speech recognition model rather than comparing different NLU models. They used the following evaluation metrics:

- Word error rate and speech recognition accuracy:
  They were used to measure the recognition ability of the automatic speech recognition approaches.
- Action Classification (AC):
  AC was measured using precision ("*percentage of correctly tagged frames among all the frames tagged by the system*"), recall ("*percentage of correctly tagged frames with respect to all the gold standard frames*"), and F-measure ("*harmonic mean between precision and recall*").
- Full Command Recognition (FCR):
  FCR was measured using accuracy (percentage of commands correctly understood).

3.2 Deep learning models

3.2.1 Object manipulation

Grounding spatial operations, such as on, in, next to, behind, etc., is key when dealing with object manipulation tasks. In order to provide a solution for this matter, the authors of [Bisk et al. [11]] developed an end-to-end model that receives as input a natural language instruction and a 3D representation of the world and produces the selection of the desired block to move, based on the grounded information. It is composed of a Bi-LSTM, to encodes the sentences, a module to apply the spatial operation to the block and a module to predict the final coordinate in the simulated space. The model is an improved version of their previous work on [Bisk et al. [10]].

It is capable of understanding complicated instructions containing actions like rotate or mirror a block and place it in different positions. The model was only tested in a simulated scenario and it was compared against their previous implementation and a model presented by other authors to solve the same task, using accuracy as the main evaluation metric for the comparison. The model can be found on [1].

A neural- and statistical based natural language approach to understand and ground natural language robot commands was presented by [Zhong et al. [79]]. Basically, the authors presented an implementation of two methods, one using Word2vec and another one using GloVe as a pre-trained tool and applied transfer learning for multimodal interaction for robots. The multimodal interaction was performed using two multiple time-scales recurrent learning models, Multiple Time-scale Recurrent Neural Network(MTRNN) and Multiple Time-scale Gated Recurrent Units (MTGRU). They evaluated the models using iCub robot [Metta et al. [55]] indicating simple commands involving object manipulation like *touch the ball* and more complicated ones like *icub touch the ball please.*

The work presented by [Sugiura and Kawai [71]] consisted of a grounded instruction understanding method capable of inferring the correct action given an instruction and situation. They focused on short sentences where verbs were missing. To do this, they used Generative Adversarial Networks (GAN). One of the characteristics of this method is that the user does not need to directly specify the desired action in the sentence, because it gets estimated by the model. For instance, the model would understand what the user means by saying *Bottle, please.* They evaluated the effectiveness of their model against two baselines models. They created their own dataset because no standard dataset existed for object manipulation instruction understanding. The dataset was constructed based on a subset of the standard visual genome dataset used for visual question answering. They used the accuracy of the model to understand the commands given, as their only evaluation metric.

The interactive text2pickup model, developed by [Ahn et al. [1]], is a method implemented for human-robot collaboration using natural language sentences containing a certain level of ambiguity. The model requires as input an image of the current environment and the command desired to be executed. It generates two heatmaps, one position heatmap with a 2D distribution of the object inferred by the

[1] `https://groundedlanguage.github.io/`

model, and one uncertainty heatmap with a 2D representation of the uncertainty in the grounding. The model copes with uncertainty in the command by interacting with the user and asking a question based on the kind of uncertainty in the received sentence. The system is composed of an Hourglass network, to process the input image by encoding a set of image features, a RNN, to encode the information embedded in the sentence, and a CNN-RNN, for the question generation stage. The authors compared it against a baseline NN model reporting the accuracy of the grounding. The model can be found on [2].

3.2.2 Navigation

The work done by [Mei et al. [54]] presented a sequence-to-sequence model for a direction following task. It uses an encoder-decoder model composed of LSTM-RNN networks to ground natural language instructions into a sequence of actions depending on the world state representation. The model has the characteristic of not requiring specialized linguistic resources like parsers or task-specific annotations methods. It was evaluated on a benchmark route instruction dataset obtaining the best result at that time for sentences containing a single action.

The Grounded Semantic Mapping Network (GSMN) is a neural network architecture developed to follow high-level navigation instructions. It was presented in [Blukis et al. [12]]. The model receives images, instructions, and pose estimates and from that information, it generates continuous low-level velocity commands. When the model receives multiple instructions, it processes them in a sequential manner. The model uses a RNN and a LSTM to perform sentence embedding. After the sentences are processed, the model constructs two maps, a relevance map indicating landmarks mentioned in the sentence, and a goal map indicating the desired goal location. The major quality of the approach here presented is the modularity of the system. It was evaluated in a simulated scenario in a fully known environment, i.e. no exploration was required.

FollowNet is an end-to-end differentiable neural architecture developed by [Shah et al. [68]]. It was used for learning multi-modal navigation policies. It maps natural language instructions in combination with visual and depth inputs to locomotion

[2] `https://github.com/hiddenmaze/InteractivePickup`

primitives. An attention mechanism was implemented to focus on the visual and depth aspects of the input received when processing the instructions. The attention mechanism gives the agent the ability to track the instruction command, focusing on the important parts while the agent explores the environment. Additionally, motion primitives, sensory observations, and parts of the instructions are paired with a reward received to cope with new instructions. The model constructed using a Deep Q-Network (DQN) and it was evaluated on a dataset composed of complex natural language navigation instructions and against a model without attention mechanisms, measuring the number way-points the agent reached correctly. The results showed that the model with attention mechanisms performed much better than the model without them.

3.2.3 Navigation and object manipulation

One way of approaching the symbol grounding problem is the implementation of a Markov Decision Process (MDP) to map natural language commands to robot actions. One example of this is the model developed by [Arumugam et al. [3]]. The authors treated the language grounding problem as a decomposition into the task inference and task execution components. This resulted in a grounding process where they extracted what they called "*callable units*" and arguments from the sentences. For instance, the sentence *go to the red room* would result in a callable unit *agentInRoom* and an argument *roomIsRed*. After grounding the sentence, a planner maps this information into a sequence of actions corresponding to the command indicated. The model is composed of an Object Oriented Markov Decision Process (OO-MDP), used to model the robot's environment and actions, a single RNN for the language model, and a planner to get the actions that the robot needs to perform. It was evaluated against other NN models on navigation and object manipulation tasks. The way the sentences are grounded into this form of representation makes the approach not easy to adapt to different environments. In the work presented, it was not clear how to define new callable units so that the model could understand new commands. The model can be found on [3].

[3]`https://github.com/h2r/GLAMDP`

Mbot is a model developed by [Martins et al. [49]]. It was developed based on the idea that a natural language command can be interpreted by finding the action indicated in the command and perform a "*slot filling*" method to collect the arguments that provide additional information necessary to fully understand the commands, such as locations, objects, etc. When designing the model, the authors explored two different approaches to interpret instructions. In the first approach 3.1, the instruction is received and divided into phrases. Then, the words in the commands are converted into feature vectors and send to the action detection and slot filling networks. The action detection result is analyzed to determine if the action belongs to the set of possible actions, if not the result of the slot filling network is erased and the action is classified as *Other*. The main aspect of this approach is that it uses the same slot filling model for all the actions. In contrast, for the second approach 3.2, a different slot filling model is implemented for all the actions that were detected, which is chosen based on the result obtained from the action detection network.

Additionally, different RNN with LSTM architectures were explored for both action detection and slot filling. The networks changes in the number of layers and LSTM cells. They were evaluated for both the action detection and slot filling process, reporting a comparison of the performance for each configuration. Furthermore, the authors compared the implementation of one-hot vectors and two kinds of word embeddings algorithms, Word2vec using skip-gram model and GloVe, both trained using the Wikipedia 2014 corpus [4] and a vocabulary set of 50000 most frequent English words. The experiments evaluated the performance of the models in the context of domestic service robots implementing two datasets, GPSR 5.2.1 and FBM3 [5]. The results showed that the first approach had better accuracy and it was computationally lighter than the second approach. The word representation that showed better results was the model using GloVe word embeddings. For all the experiments, they used accuracy as their evaluation metric.

[4] `https://corpus.byu.edu/wiki/`

[5] `http://thewiki.rockinrobotchallenge.eu/index.php?title=Datasets`

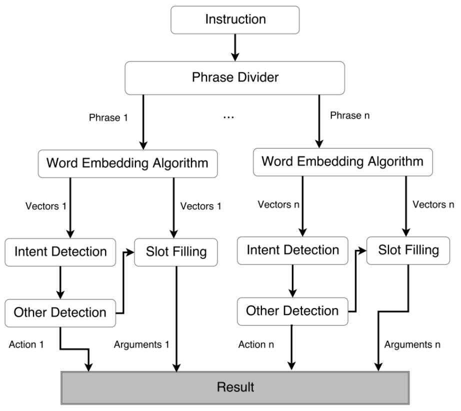


Figure 3.1: Scheme of the first approach. Figure source: [Martins et al. [49]]

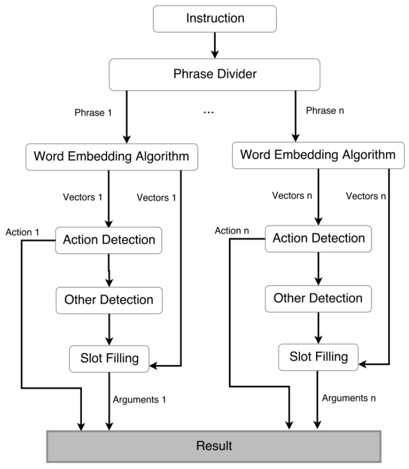


Figure 3.2: Scheme of the second approach. Figure source: [Martins et al. [49]]

3.3 Probabilistic graphical models

3.3.1 Navigation and object manipulation

One of the first probabilistic graphical models that were developed to ground natural language robot commands was the Generalized Grounding Graph (G^3). It was presented by [Tellex et al. [72]], with a more recent adaptation presented in [Kollar et al. [43]]. The model is a combination of their previous work presented in [Kollar et al. [42]] and [Huang et al. [39]]. The model constructs a probabilistic graphical model dynamically taking into account the linguistic parse structure of the natural language command received. The grounding graph is constructed using Spatial Directional Clauses (SDCs) [Kollar et al. [42]]. Each SDC corresponds to a linguistic constituent that is in the command and represents a figure, a relation or a variable number of landmarks in the environment. The model was trained following a supervised approach using a corpus composed of commands paired with the respective grounding in the form of a tuple containing the 3D shape of the object, the trajectory through space, and a set of symbolic perceptual tags. The corpus contained commands in the context of controlling a forklift, a mobile manipulator, a robotic wheelchair, and a robotic micro-air vehicle. The authors reported the precision of the model to understand commands performing minor modifications to the structure of the model. Based on the results, they claimed that the approach could generalize to many different domains where you could use linguistic constituents paired with the corresponding action and a set of world features. As it was highlighted in [Pillai and Matuszek [65]], the approach requires a manual cleaning process due to the noise that is present in language itself.

The Distributed Correspondence Graph (DCG) [6] was presented in [Howard et al. [38]]. It was built to address the difficulty of directly inferring the most likely constraints (distance, orientation, and contact), objective functions, and model dynamics that was presented when using the G^3 model. It uses the conditional independence assumptions that were already presented in the G^3 and considered the components of a grounding to be also conditionally independent with a finite and well-known space of constituents. One of the main characteristics of this model is that it can infer the most likely set of planning constraints from natural language instructions. Then, having completed the grounding process, a planner is used to compute a sequence of actions to fully execute the instruction requested. Basically, it identifies the action encoded by the language by detecting the individual steps of the planning constraint inference and motion planning. The authors explored the use of both the Stanford Parser [Klein and Manning [41]] and the Cocke-Kasami-Younger algorithm [Younger [78]] to create the parse trees for the commands. When multiple

[6]It was named DCG because "*it distributes conditionally independent elements of phrase groundings across multiple factors*".

commands are given to the model, it represents them as "*a single parse tree with an implicit root phrase to connect the individual sentences*", as mentioned in [Arkin et al. [2]]. Due to the fact that when the DCG model was created, the authors did not know how the linguistics constituents correlated to the phrases, the run-time for the model tended to increase linearly with the number of constituents, as it was shown in [Chung et al. [19]]. The model can be found in [7].

One of the most recent probabilistic graphical models that have been developed to process natural language robot commands is the Monologic Distributed Correspondence Graph (MDCG). It was presented in [Arkin et al. [2]] with the goal of creating a model capable of understanding monologues composed of multiple commands. It was created as an extension of the DCG model. MDCG segments the monologues into a sequence of sentences in order to improve the efficiency of the grounding for each of the instruction. The sentences that are in the monologue are treated as an individual labeled parse tree and, using the trees, the meaning behind each instruction is collected, but taking into consideration the context of previously inferred groundings from the previous sentences. The major improvement this model had over the normal DCG was that it is possible to initiate the execution of the first sentence, while inferring the meaning of the remaining sentences in the monologue. The authors evaluated the model using two corpora consisted of simulated robot behaviors labeled using human user interactions through Amazon Mechanical Turk [8] and implementing the model in a Baxter robot for manipulation commands.

The Probabilistic Action Cores (PRAC) is a probabilistic knowledge base interpretation model developed by Daniel Nyga. In the most recent publication [Nyga et al. [60]], the authors focused on grounding natural language robot commands that corresponded to high-level tasks. For instance, the instruction *make me a banana milkshake* is a high-level task in the sense that it is not specified each subtask that is required to fulfill the instruction. PRAC is capable of inferring a plan to fulfill the requested high-level task, based on semantic co-associations learned from textual task descriptions. The probabilistic relationships between semantic concepts are modeled using a Markov Logic Network (MLN). The model grounds a sentence by calculating a set of predicates that represents observations of the task specification

[7]`https://github.com/tmhoward/h2sl`

[8]`https://worker.mturk.com`

and knowledge carried in the instruction. This is done by using a log-linear model, mapping phrases in the instruction with semantic concepts derived from the partially observed workspace. Two kinds of evaluation were performed. The first evaluation consisted of a quantitative evaluation where they compare the proposed model against a baseline model for the scenarios of workplace context, partially known workspace, and acquiring missing knowledge. The second evaluation was a qualitative evaluation where they implemented the model in a Baxter robot and a simulated PR2 robot performing clearing, assembly, and recipe preparation tasks. The results showed that the model is indeed capable of creating a plan with better accuracy than the baseline model. As the major downside that we could find for this model was the run-time. It takes a considerably long time to process an instruction and to generate the plan of actions to execute it. The model can be found in [9].

3.3.2 Navigation

Another adaptation of the DCG model was presented in [Chung et al. [19]]. The model is called Hierarchical Distributed Correspondence Graph (HDCG). It is capable of learning a set of rules from a language command excluding the highly unlikely symbols from the search space and speeding up the grounding process. Furthermore, the model overcame the limitation of the previous probabilistic graphical models when dealing with large and complex symbolic representations. It uses parse trees similar to the ones shown in Figure 3.3 and models of the environment to learn the structure of the sentences and infer the meaning based on that. One of the main difference between this model, DCG, and G^3 is that it assumes that the space of the linguistic constituents as a function of the utterance in the context of the environment. Using this assumption and applying the learned set of rules to exclude unlikely symbols the model reduces the number of active constituents, as it can be appreciated in Figure 3.4. The authors compared the performance of the HDCG against the DCG and G^3 getting navigation commands and reported the accuracy and run-time for the models. The core code of the model can be found in [10].

[9] `http://www.actioncores.org/`

[10] `https://github.com/tmhoward/h2sl`

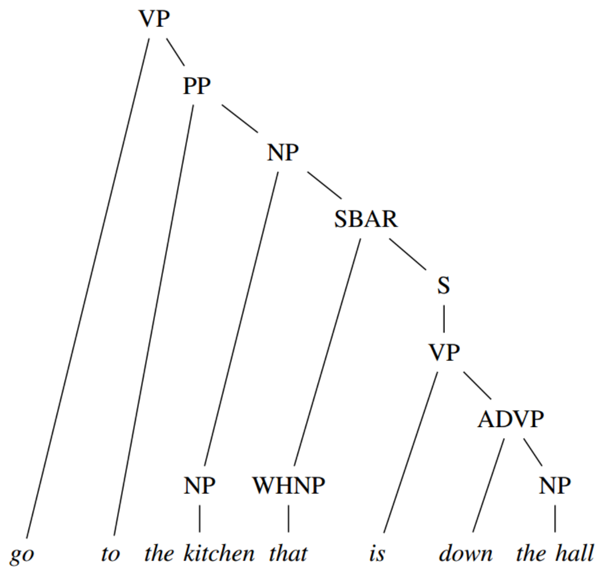
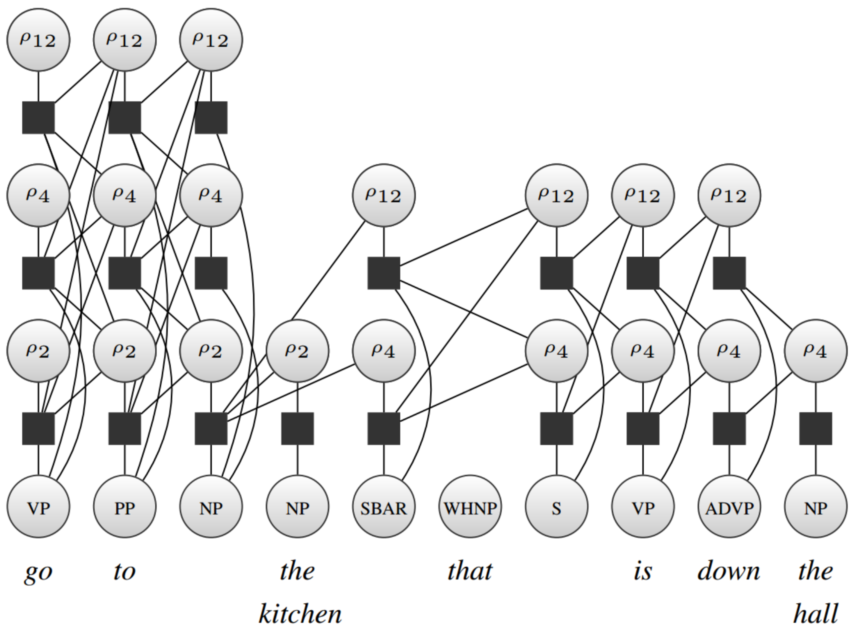


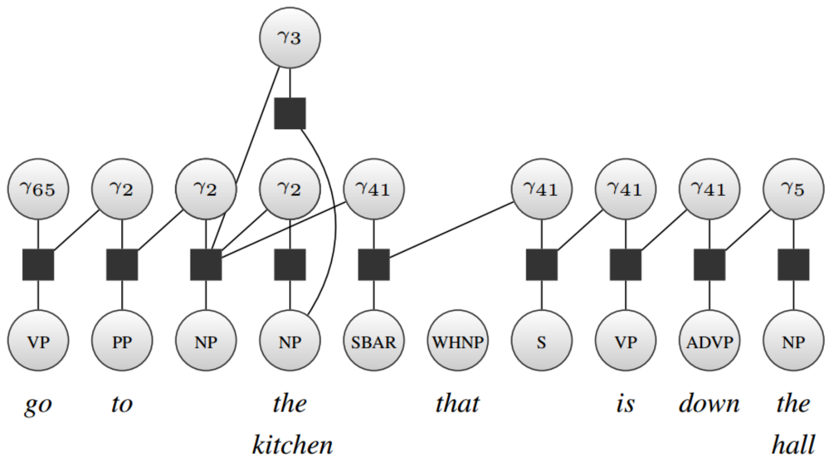
Figure 3.3: Parse tree of the sentence *go to the kitchen that is down the hall*. Figure source: [Chung et al. [19]].

3.3.3 Object manipulation

Improving the work that was1 done for DCG and HDCG models, the authors of [Paul et al. [62]] developed the Adaptive Distributed Correspondence Graph (ADCG). It is a probabilistic graphical model that created a space of abstract spatial concepts, notions of cardinality (one, two, three, etc.) and ordinality (second, third, etc.). In this model, abstractions are represented as hierarchical groundings and are linked to linguistic constituents in a probabilistic manner. The concrete and abstract symbol spaces form the generalized space of groundings. It was shown that the space of abstract groundings was exponentially large, which could be a problem when the robot would be operating in a complex environment with a large number of objects with diverse spatial layouts. The inference process for this model was posed as a tree-structured search over possible correspondences between phrases and groundings. The model was evaluated against the DCG model reporting an improvement in the accuracy of the grounding and the run-time when dealing with manipulation commands.



(a) Active constituents using the DCG model for the sentence *go to the kitchen that is down the hall.*



(b) Active constituents using the HDCG model for the sentence *go to the kitchen that is down the hall.*

Figure 3.4: HDCG vs DCG. Figures source: [Chung et al. [19]]

An implementation of the DCG model for object manipulation was presented in [Broad et al. [16]]. The authors developed a natural language interface capable of correcting the interpretation the model generates when receiving a command. The model uses a grammar and a set of features to relate linguistic constituents to "*corrective actions*" and to the manipulator trajectory. Corrections can be provided to the model by interrupting the current motion of the robot and indicate new information about the speed and orientation of the end effector, or about the spatial constraints during the trajectory. The model was evaluated by implementing on a physical robot manipulator and performing studies with human users with motor impairments to see the actual usefulness of their approach when facing real-life problems. The results showed the need to collect more data to improve the language model to solve conflicts between the data it was used to train and the inputs the user provided to correct the previous interpretations.

3.4 Semanic parsers

Encouraged by the idea of providing a solution to the problem of having restricted vocabulary on semantic parsers for end-user programming, the authors of [Sales et al. [67]] proposed a semantic parser capable of mapping natural language commands to actions. The commands concerned actions like currency exchange, translation, image localization, creating issues in Github, and some others. The model consisted of a "*distributional semantic parsing method with a semantic pivoting heuristic*". The parser relies on frame embeddings, which represents actions and parameters as a distributional vector space, to identify the action from the sentence and the complementary parameters. The semantic parsing is done in four steps: (1) The sentence is grounded into a representation having an action descriptor and the set of objects mentioned in the command. (2) A set of actions is selected based on the action descriptor. (3) The action candidates are represented based on a feature set. (4) A classifier runs over the action candidates and ranks them to show them to the user. The authors developed different architectures changed the models to perform step 2 and step 4, and evaluated them over 185 natural language commands.

3.4.1 Navigation

The work done by [Matuszek et al. [52]] differs from previous implementations of semantic parsers as their approach learns the language based on the situated context of the received natural language commands. The parser can maps commands to actions and create a distribution over possible robot control sequences. They trained the parser using English commands paired with control language expressions. It was evaluated using navigation commands consisted of short and long paths and using precision, recall, and F-measure as the evaluation metrics.

3.4.2 Object manipulation

The work done by [Dukes [27]] presented the SemEval-2014 task to encourage semantic parsing research. It was a "*high-quality*" annotated dataset to compare and evaluate approaches. The task consisted of contextual parsing of robot commands in the context of manipulators. For the evaluation stage, six teams submitted a model using both rule-based and statistical methods. The results showed that understanding of robot commands remained a challenge, even with a fixed domain.

3.4.3 Navigation and object manipulation

Another framework using a semantic parser implemented to ground natural language robot commands is Flipper, which was presented in [Boldt et al. [15]]. As defined by the authors, "*It is a natural language interface for describing high levels task specifications for robots that are compiled into robot actions*". Flipper was designed as a system for end-users to program mobile robots using robot commands. It has a back-end doing the semantic parsing, which is a custom version of Sempre [Berant et al. [8]], a front-end, which is a GUI, and a planer, which is an adapted version of $A*$. The semantic parser translates the commands into internal representations for programs. When it fails to do so, it is possible for the user to declare the command failed to understood as a new one and implement a sequence of already understood actions in required to execute the new command. By using a naturalization process, the grounding capabilities of the model can be improved by constant user interactions. As it was designed as a based framework, it currently deals with single instruction sentences with basic actions, such as visit certain locations and picking or dropping tasks. The implementation of the model can be seen in [11].

3.5 Statistical models

The authors of [Misra et al. [58]] presented a statistical method to ground natural language instructions in the context of manipulation tasks. They created their own dataset of task descriptions pairing commands with the corresponding grounded task-logs. The major characteristic of their model was that it was capable of handling missing or incomplete commands. The environment and task context is encoded into an energy function over a CRF grounding instructions into environment tasks. They used two evaluation metrics for their model: (1) Instruction edit distance to measure the distance between two instruction sequences comparing the model's output against the ground-truth. (2) Environment edit distance to measure the distance between the ground-truth sequence of environments and the predicted sequence of environments.

3.6 Named entity recognition

The authors of [Chesworth et al. [18]] implemented a NER model, using JCarafe [12], with a CRF to process military mission commands. The entities inside the mission files were classified and labeled appropriately in order to be used in a second stage, which was not implemented by the authors. The second stage was supposed to process the entities found in the files in order to provide that information to an robot to fulfill the requested mission.

Another project that used a NER model to process important information for robots is the work presented by [Costa et al. [23]]. They extracted entities from instructions manuals for assembly operations. Their idea was to create a model that could gather information from assembly manuals and use that information to teach industrial robots to perform some tasks by demonstration. Furthermore, they provided a detailed analysis of the fine tunning process for the Stanford named entity recognizer [13]. They performed 91 tests changing a single parameter for each test,

[11] `flipper.mpi-sws.org`
[12] `https://github.com/wellner/jcarafe`
[13] `https://nlp.stanford.edu/software/CRF-NER.shtml`

based on the recommended configuration, and analyzed the impact that parameter had using precision, recall, and F-measure as their evaluation metrics.

Rasa is a language understanding and dialogue management open-source python library that was presented in [Bocklisch et al. [13]]. It is composed of two modules, Rasa NLU and Rasa Core. The authors developed the library to simplify the implementation of dialogue management and language understanding models for chatbots and assistant programs. Rasa NLU is a module used to identify the intention behind natural language sentences and detect the entities that are in them. It uses a set of pre-defined pipelines to process sentences by performing a set of steps, such as tokenization, POS-tagging, and word representation. This module is used to indicate tasks to the assistant programs. The dialogue management is implemented by Rasa Core and is capable of handling complex conversations with users. The library can be found in in [14].

3.7 Embodied Construction Grammar

ECG2 is a framework introduced by [Eppe et al. [29]], which is an improved version of the framework presented in [Feldman et al. [31]]. It was designed to infer the meaning of sentences using combinations of words and constructions. The framework was constructed based on the neural theory of language [Feldman et al. [31]] and cognitive linguistics [Fillmore [32]]. The framework contains formalisms and technical notations to describe the grammar implements and the meaning of the sentences. It uses construction grammars to understand natural language sentences, as they are capable of describing language at different levels, such as phonology, morphology, syntax, and semantics. The natural language commands are processed by encoding the inferred information as a JSON structure called n-tuple. In the work presented, the authors provided three different application examples of the framework, one using a Morse robotics environment to simulate two robots obeying navigation commands, one implementing the framework into ROS to prove the ability of the model to understand navigation and object manipulation commands, and one consisted of an agent playing a strategy computer game in real-time. Also, in their work done previously, [Khayrallah et al. [40]] [Eppe et al. [28]][Trott et al. [74]],

[14]`https://github.com/RasaHQ/`

the authors demonstrated how ECG could be implemented for autonomous systems, robotic applications, and multi-agent systems.

3.8 Frame semantics

The Language Understanding chain For Robots (LU4R) was presented by [Bastianelli et al. [6]]. It has a language understanding part, which was based on the theory behind frame semantics [Fillmore [33]], to give a linguistic and cognitive basis to the interpretation of the actions encoded in a natural language command. It combines discriminative learning and distributional semantics. The understanding process of the model was combined with perceptual information from the environment using semantic maps where real-world objects were related to multiple lexical labels expressing the name of the entities and extracting the spatial relations between them. It was evaluated in the context of domestic service robots. Three different configurations of the model were reported: (1) No perceptual information was considered. (2) Perfect grounding information was assumed. (3) Grounding information was based on the optimized grounding function. The experiments measured the performance of the three configurations to perform action detection, argument identification, argument classification, and the complete interpretation of the sentence. For each of the tasks, precision, recall, and F-measure were used as the evaluation metrics. The results showed that including perceptual information into the language understanding improved the understanding of the commands.

The work done by [Evangelista et al. [30]] showed an application of the LU4R model. In here the authors implemented the model in an industrial scenario, more specifically in an assisted environment exploration and a collaborative part assembling. They stated that they used LU4R as their NLU model because it was specifically developed for handling language ambiguity in HRI. They implemented and evaluated their whole system for tasks in the context of the *Robocup@Work* competition. They used the success rate, time execution mean, and time execution standard deviation as their evaluation metrics for repeating a set of tasks 10 times each.

3.9 Limitations of the previous work

The surveys that we have encountered were not able to provide a comparative analysis and evaluations of the current state of the art. It was complicated to clearly appreciate the benefits and drawbacks each of the approaches has to offer. Additionally, some of the evaluations that have been developed, compared models that where following the same approach, and did not cover models from a different ones. For instance, grammar based models against deep learning models.

For the models that were following deep learning approaches, it is still a problem the huge datasets required to train the models. Also, it is required to explore new word embeddings approaches to look for improvements when facing words that were not present in the training corpus for the word embeddings.

The uncertainty that is present in the way we speak remains a challenge. It is still complicated to properly interpret sentences that uses demonstrative articles [15] or the pronoun *it* to refer to objects. Furthermore, the complete process of getting a natural language command, generating an interpretation, and creating the sequence of actions required to fulfill the command is still not optimal. Probabilistic graphical models seem the most promising approach for this matter, but there are still issues that needs to be solved, such as reducing the exploration time of the solution space, and generating actions plans for high-level actions, not just primitive actions.

[15]The words this, that, these, and those

4 Comparative analysis

In this chapter, we are presenting a comparative analysis of the state of the art models that are open source and can be used to generate interpretation of natural language robot commands. We did not include any semantic parser implementation because none of the models that we reviewed are open source nor provided a good guideline on how to recreate the models and adapt them to different domains. In regards to the probabilistic graphical models, we reviewed the possibility to implement DCG [Howard et al. [38]] and PRAC [Nyga et al. [60]] to our use cases. But, we decided to not include them due to the following reasons:

1. Using the open source version of DCG, we were only able to obtain a probabilistic graph and a sentence containing POS-tags when giving a natural language command. We were not able to obtain a higher-level interpretation of the sentences that could be used to perform a comparison with the other NLU models in our survey
2. Similar to DCG, we could not get a higher-level interpretation of natural language commands using PRAC. Additionally, after testing the online version of it [1], we found out that the model requires a long time to analyze a sentence and generate the sequence of actions that are required to execute it.

The models that we have selected are the following:

[1]`http://prac.open-ease.org/`

- Mbot is one of the models that follows a deep learning approach. It has already been used for one of our use cases, which is GPSR category 1. It is an open source model with reasonably well-documented code that helps adapt it to new domains and use it.
- Rasa was selected as a model capable of performing not just NER, but also identifying the intention behind sentences. It is open source python library and it has decent documentation that provide guidance on know how to create a model and train it on a completely new domain without many complications. Even though its focus is mainly in chatbots and assistance programs, we saw feasible the option of developing a Rasa model to process natural language robot commands.
- LU4R was selected as a model that relies on frame semantics. We were able to obtain the compiled version of the model by requesting it to the authors, not allowing us to retrain it or adapt to new commands.
- ECG was selected as a model implementing construction grammars. It is an open-source model, which has very detailed documentation of the components of the model.

4.1 Mbot

4.1.1 General description

Mbot is a NLU model developed by [Martins et al. [49]], based on the idea that robot commands can be processed by performing action detection (also known as intent detection) and slot filling, to find the action requested and the arguments that complement the action. As an example, if we give the sentence *navigate to the kitchen and take the bottle from the table* to the model, it would produce as output the interpretation *go destination kitchen take object bottle source table*, in which we can identify two intentions: *go* and *take*, and three slots *destination, object, and source* with the respective values of *kitchen, bottle, and table.*

The model uses GloVe word embedding algorithm to represent words in a dictionary and to capture synonyms. It was trained using the Wikipedia 2014 corpus [2], which contains 1.6 billion tokens. A pre-processing of the corpus was required in order to be used, t was tokenized and lowercased using NLTK [3] python package, punctuation marks and numbers were also removed. Additionally to the Wikipedia corpora, a vocabulary consisting of 50,000 "*most frequent English words was used*". The dimension of the words vectors was set to 300 with a symmetric window of 10 words.

Mbot is an open source model that can be found in [4]. It has been implemented in ROS, which eases the work of deploying the model in a robotic platform. Furthermore, it has been used for domestic service robot competitions and the repository of the model contains pre-trained classifiers that can infer actions like *go, take, find, answer, and tell* and slots like *object, destination, source, sentence, and person.*

4.1.2 Structure

For the action detection, a RNN containing a single layer with 500 LSTM cells is used to capture the meaning and connections between all the words in a command,

[2]`https://corpus.byu.edu/wiki/`

[3]`http://www.nltk.org/`

[4]`https://github.com/socrob/mbot_natural_language_processing`

having word vectors as input for the network and producing, as output, a set of values expressing the amount of certainty that the action in the command corresponds to one of the possible actions the robot can perform, selecting the one with the highest value. Due to the possibility of getting a command containing an action not possible to be performed by the robot, a SVM is used to classify if the action in the command does not belong to the set of possible actions. The action detection network was trained using Adam optimizer with different learning rates, minimizing a softmax cross-entropy loss function. The SVM, used to classify valid actions, was trained using a subset of the data used to train the action detection NN with the addition of commands that should be classified as *Other*.

The slot filling task is done by using a RNN with 2 layers of 500 LSTM cells. Similar to the action detection, word vectors were used as input. The output of the slot filling network is a tag for each of the words in the sentence following the IOB format [5]. In case that a word has a tag that does not fit with the set of arguments according to the action detected, the tag is replaced by the *O* tag. The slot filling network was trained in a similar fashion that the action detection network. An example of a slot filling output is shown in Table 4.1

Instruction	deliver	a	coke	to	peter	at	the	living	room
Slot tags	O	O	B- object	O	B- person	O	O	B- destination	I- destination

Table 4.1: Example of the slot filling output for the sentence ”textitdeliver a coke to peter at the living room. This table was obtained from [Martins [50]]

Pipeline

The pipeline of the model can be appreciated in Figure 4.1. In more details, the model receives a natural language command that gets divided into phrases, each containing an action [6]. For each of the phrases found in the command, a filtering step is done, removing undesired characters and converting the phrase to lowercase. Then, the indices of all the words in the phrase are found in a dictionary (Wikipedia

[5]B- Beginning word of the entity, I- Inside word of the entity, O- Out word

[6]Currently, the phrases are identified by finding commas and the word *and* in the given sentences. The authors explored the implementation of a tool called SyntaxNet, `https://opensource.google.com/projects/syntaxnet`, to improve this by identifying the verbs in the sentences and split them according to the verbs. Unfortunately, it is not used in the current version of Mbot.

2014 corpora plus the 50,000 most frequent English words) and stored in a feature vector, of size 15 [7].

The feature vectors obtained from each of the phrases are used as input for the action detection and slot-filling network, producing an interpretation of the given command containing the intentions behind it and the arguments necessary to accomplish them.

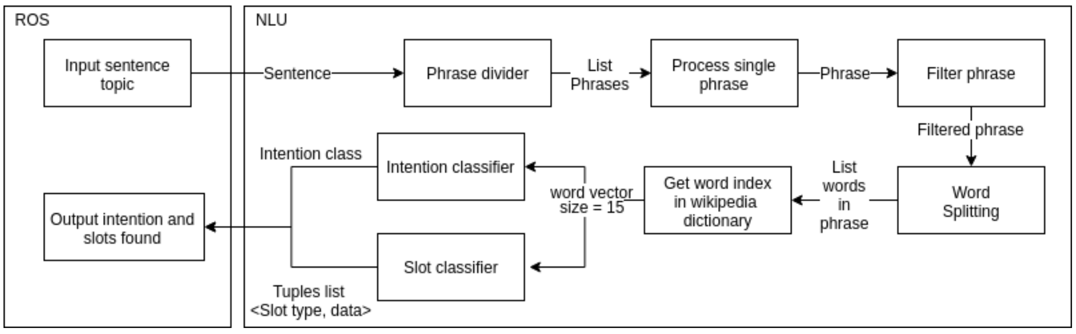


Figure 4.1: Pipeline of Mbot. Figure created based on the diagrams shown in [Martins et al. [49]] and [Martins [50]].

[7]The size of the feature vector was chosen by [Martins [50]] and is consistent with the feature vectors used during the training and the inferring process. Based on the information provided by the authors, it can be understood that the number 15 was chosen due to the implementation of negative sampling, as it was recommended in [Mikolov et al. [57]] having *k=15* sampled words for each vector.

4.1.3 System requirements

Training

Python 2.7 libraries	Python 3.5 libraries	Debian packages
asciitree	asciitree	bazel [a]
tensorflow 1.4.0 (or higher)	PyYAML	openjdk-8-jdk
numpy	numpy	swig
msgpack	scipy	python-mock
msgpack_numpy	scipy	graphviz
	scikit-learn	libgraphviz-dev
	msgpack	unzip
	msgpack_numpy	

Table 4.2: System requirements needed to train Mbot

[a] `http://storage.googleapis.com/bazel-apt`

Execution

Python 2.7 libraries	Python 3.5 libraries	Extras
asciitree	numpy	ROS Kinetic [a]
tensorflow 1.4.0 (or higher)	msgpack	
numpy	msgpack_numpy	
msgpack		
msgpack_numpy		

Table 4.3: System requirements needed to use Mbot

[a] The NLU part of Mbot is ROS independent.

In the model's repository, we can find an installation script that covers all the requirements previously mentioned. To start the installation process, it is necessary to execute the file *repository.debs*, which calls a set of shell files. The first shell file called by the *repository.debs* is the *nlu_setup.sh*. This file does the following:

1. Install the required libraries, for execution of the model, for both python 2.7 and 3.5.

2. Install the debian packages mentioned above (bazel, openjdk, etc.).

3. Clone some tensorflow models from SocRob's github repository [8].

4. Configure the cloned tensorflow models. During this process several paths and settings needs to be specified:

    4.1. Local python location and python library path.

    4.2. Optimization flags to use during compilation.

    4.3. If jeamlloc is going to be used as the malloc implementation.

    4.4. If Tensorflow is going to be built with Google Cloud Platform support, Hadoop file system support, XLA just-in-time compiler, OpenCL support, and/or CUDA support.

The second file that is executed is *nlu_training_setup.sh*. This file does the following:

1. Install required debian packages and python 3.5 libraries for training the model.

2. Downloads Wikipedia pre-trained vectors [9] to the nlu_training directory, keeping only the 300 dimension vectors (the zip file contains 50,200 and 100 dimension vectors).

3. Generate a dictionary using the file *gen_wikipedia_vectors.sh*

The last file to be executed is *download_classifiers.sh*. This file does the following:

1. Request the selection of one of the available pre-trained classifiers and downloads it (pedro_gpsr, mithun_gpsr, mithun_gpsr_robocup, or mithun_eegpsr_robocup).

[8]`https://github.com/socrob/models.git`.

[9]`http://nlp.stanford.edu/data/glove.6B.zip`

4.1.4 Implementation process

Training

The training process of the intention classifier starts with the creation of the inputs and outputs training files. To do so, the authors have provided several sample python scripts for three different domestic robot competitions: EEGPSR, ERL, and GPSR. Focusing in the latter, the file is named *gpsr_data_generator.py.* In this file it is necessary to specify the intentions desired to classify, as well as the number of sample sentences for each intention. Additionally, a list of objects, locations, names, sentences, and intros [10] is also defined. For each of the intentions desired, it is necessary to create a set of natural language commands and add labels, to indicates the intentions, at the end of the commands. It is necessary to have as many sentences as the number of samples previously defined, because this will allow us to have a balance dataset and improve the training procedure. An example of a valid training sentence would be *navigate to the kitchen - go.* The label is the word go.

Once all the different commands are created, a random sampling process is executed, adding intros to the commands, and removing the ones that have more than 15 words. Finally, the sampled commands are stored in an inputs file, while the labels are stored in an outputs file.

The process of creating the training files for the slot classifier is quite similar to the intention classifier, with the exception that the words that compose the lists of objects, locations, names, sentences, and intros are tagged using the IOB format. Also, the commands do not require the addition of the intention label at the end. The inputs file contains the commands randomly sampled without the tags, while the outputs file contains the tags of the slots in the sampled commands.

After the inputs and outputs files are created for both intention and slot classifiers, it is possible to execute the *train_nn_model.py* script corresponding to the training of intention classifier or the one to train the slots classifier. The file starts by reading the hyper-parameters of the network from a yaml file. Then, it imports the word vectors, the dictionary, and the training inputs and outputs files. The training of the RNN for the classifiers is done by randomly selecting a number of training and

[10] An intro is the beginning of a sentence similar to *could you ... or Hey lucy ...*

validation batches, from the training files, at each epoch and calculating the training loss, using softmax cross-entropy, and the validation accuracy of the model. To avoid having overfitting, the model performs early stopping based on the validation accuracy. The file to train the slots classifier is similar to the training file of the intention classifier , with the exception that it trains a different RNN model.

Execution

The process of executing the model is quite simple. First the ROS launch file is executed by calling 4.1. Then, we subscribe to the output recognition topic to see the intention and slots identified by the model by running 4.2. To send a command to the model, it is just necessary to publish to the input sentence topic by executing 4.3. The output produced by the model for this particular example would be 4.4.

The NLU part of Mbot is ROS independent, meaning that it is possible to use the model without the need of using ROS. In order to do so, we just need to use the file *mbot_nlu_common.py* that contains the *NaturalLanguageUnderstanding* class with all the methods necessary to initialize the model, process sententeces, find intentions and slots, and finalize the model.

Listing 4.1: Executing the launch file using the mithun gpsr robocup classifier

```
roslaunch mbot_nlu mbot_nlu.launch
        nlu_classifier:=mithun_gpsr_robocup
```

Listing 4.2: Subscribe to the output recognition topic

```
rostopic echo /hri/nlu/mbot_nlu/output_recognition
```

Listing 4.3: Publish a command to the input sentence topic

```
rostopic pub --once /hri/nlu/mbot_nlu/input_sentence
        std_msgs/String "data: 'go to the kitchen'"
```

Listing 4.4: Output of Mbot for the command go to the kitchen

```
sentence_recognition:
        intention: "go"
        slots:
                type: "destination"
                data: "kitchen"
---
```

4.2 Rasa

4.2.1 General description

Rasa is an open source python library designed to ease the work of building chatbots and AI assistants without the need of having linguistic knowledge. It was formally presented in [Bocklisch et al. [13]] and is available in [11].

The library is composed of two main parts: Rasa core and Rasa NLU. Rasa core is a framework that helps to build conversational software, i.e. chatbots. It contains tools to create systems capable of following conversations and keeping track of the desired information. On the other hand, Rasa NLU is used to infer the meaning behind the sentences given by the user. It is capable of performing intent classification and entity recognition of sentences. Rasa NLU allows the use of many different languages when creating the model. But, to do so, it is necessary to use either pre-trained word embeddings or train new word embeddings by providing a large dataset. In here, we only cover Rasa NLU in more details, as it is the one we are going to use to create a model to interpret natural language robot commands.

4.2.2 Structure

The Rasa NLU library presents different modules to perform entity recognition:

- *ner_crf*: Uses a CRF that allows the use of custom entities, i.e. entities that are not places, dates, people, nor organizations.
- *ner_spacy*: Uses an averaged perceptron to identify common entities, such as places, dates, people, and organizations.
- *ner_duckling_http*: It uses a context-free grammar model, providing pre-trained entities.
- *ner_mitie*: It uses a structured SVM that is useful for custom entities.

[11] `https://github.com/RasaHQ`

Regarding the classification of intents, the library provides different classifiers:

- Embedding classifier:

  The embedding intent classifier uses supervised embeddings combining user inputs and intent labels into the same space. The classifier is trained by maximizing the similarity between the inputs and the labels. It requires the implementation of a featurizer prior the classification of the sentence. The structure of the classifier was designed based on StarSpace model [12].

- Keyword classifier:

  This is a simple classifier that looks for special keywords in the sentence and classifies them depending of the words found. For instance, if the word *hello* or *hi* is found in the sentence, the classifier would return *greet* as the intention.

- Mitie classifier:

  Mitie classifier is constructed using MITIE framework [13] and requires the implementation of a Mitie NLP module and a feature extractor module.

- Sklearn classifier:

  Sklearn classifier is constructed using scikit-learn framework [14]. It uses a SVM with a linear kernel, trained by performing cross-validation to select the C parameter from a set of possible values and using F-measure as the evaluation metric.

Pipeline

All sentences that are received by the NLU model are analyzed by executing different processing modules. The processing modules desired and the order can be predefined by using one of the four pre-configured processing pipelines, or can be explicitly mention by creating a customized pipeline. The pre-configured processing pipelines are:

[12] https://arxiv.org/abs/1709.03856
[13] https://github.com/mit-nlp/MITIE
[14] https://scikit-learn.org/stable/

- Spacy sklearn:

  Spacy sklearn pipeline is shown in 4.6. It can be appreciated that the pipeline contains a text processing stage, an intent and an entity featurizer stage, a NER stage using *ner_crf*, and an intent classification module. It is also possible to choose this pipeline just by specifying the name of the pipeline, as shown in 4.5.

  This pipeline is recommended when the training data corpus is less than 1000 training sentences and a spacy model exists for the language desired. One of the major advantages it provides is the ability to interpret words that are similar to the ones presented in the training dataset without being explicitly mentioned in it, due to the use of word embeddings vectors. For example, if the training dataset contains sentences mentioning fruit objects, but none of the training examples contains *apple* as the desired object, it would be possible for the model to identify that object when receiving a command requesting it. This is due to the ability of word embeddings to group similar words in the vector space.

- Tensorflow embedding:

  Tensorflow embedding pipeline is shown in 4.8. It can be appreciated that the pipeline contains a tokenization module, a NER stage using *ner_crf*, an intent featurizer module, and an intent classification module. It is also possible to choose this pipeline just by specifying the name of the pipeline, as shown in 4.7.

  This pipeline is more suitable than the previous one when the training data corpus contains more than 1000 training sentences. This allows the model to create word vector representations that are particular to a specific domain.

- Mitie:

  Mitie pipeline is presented in 4.9. It contains modules to perform text processing by extraction features and tokenizing the sentences. It continues the process by performing NER using *ner_mitie*, and classifying the intention of the sentences. The authors do not recommend the use of this pipeline as it will be deprecated in the near future.

- Mitie Sklearn:

  Mitie Sklearn pipeline is presented in 4.10. It contains modules to perform text processing by extraction features and tokenizing the sentences. It continues the process by performing NER using *ner_mitie* and classifying the intention of the sentences using a sklearn classifier. The authors do not recommend the use of this pipeline as it will be deprecated in the near future.

Listing 4.5: Config file for Spacy sklearn pre-configured pipeline using template

```
language: en
pipeline: spacy_sklearn
```

Listing 4.6: Config file for Spacy sklearn pre-configured pipeline

```
language: en
pipeline:
- name: nlp_spacy
- name: tokenizer_spacy
- name: intent_entity_featurizer_regex
- name: intent_featurizer_spacy
- name: ner_crf
- name: ner_synonyms
- name: intent_classifier_sklearn
```

Listing 4.7: Config file for tensorflow embedding pre-configured pipeline using template

```
language: en
pipeline: tensorflow_embedding
```

Listing 4.8: Config file for tensorflow embedding pre-configured pipeline

```
language: en
pipeline:
- name: tokenizer_whitespace
- name: ner_crf
- name: ner_synonyms
- name: intent_featurizer_count_vectors
- name: intent_classifier_tensorflow_embedding
```

Listing 4.9: Config file for Mitie pre-configured pipeline

```
language: en
pipeline:
- name: nlp_mitie
  model: data/total_word_feature_extractor.dat
- name: tokenizer_mitie
- name: ner_mitie
- name: ner_synonyms
- name: intent_entity_featurizer_regex
- name: intent_classifier_mitie
```

Listing 4.10: Config file for Mitie sklearn pre-configured pipeline

```
language: en
pipeline:
- name: nlp_mitie
  model: data/total_word_feature_extractor.dat
- name: tokenizer_mitie
- name: ner_mitie
- name: ner_synonyms
- name: intent_entity_featurizer_regex
- name: intent_featurizer_mitie
- name: intent_classifier_sklearn
```

4.2.3 System requirements

As Rasa is a python library, it is possible to install it using pip 4.11 or manually through github 4.12

Listing 4.11: Command to install Rasa NLU through pip

```
pip install rasa_nlu
```

Listing 4.12: Rasa NLU installation process through github

```
git clone https://github.com/RasaHQ/rasa_nlu.git
cd rasa_nlu
pip install -r alt_requirements/requirements_full.txt
pip install -e
```

4.2.4 Implementation process

Training

In order to train the intention classifier and the entity recognition model, it is necessary to develop a training markdown file, similar to the one shown in 4.13. The intentions that correspond to the training sentences need to be specified, as well as the entities desired to be classified. Once the training file is created, it is necessary to develop a configuration yml file, similar to the ones shown in the pipeline section. To start training the model, the training script is executed by running the command shown in 4.14, specifying the path to the configuration yml file, the path of the dataset markdown file, and saving the trained model in the *./models/current/nlu* path relative to the current working directory.

Listing 4.13: Example of a training file for Rasa NLU

```
## intent:go
- go to the [kitchen](destination)
- move to the [bedroom](destination)
- navigate to the [living room](destination)
- move to the [stove](destination)
- navigate to the [corridor](destination)
## intent:take
- put the [noodles](object) on the [bookshelf](destination)
- take the [bottle](object) from the [table](source)
- give [me](person) the [soap](object)
- grasp the [tea](object) from the [bookshelf](source)
```

Listing 4.14: Running the training script for Rasa NLU

```
python3 -m rasa_nlu.train --config nlu_config.yml
                          --data nlu.md -o models
                          --fixed_model_name nlu
                          --project current --verbose
```

Execution

The process to use a trained Rasa NLU model is quite simple, as shown in 4.15. It just requires to import the *Interpreter* class from the *rasa_nlu.model*, import Json, load the trained model, send a sentence to the parse function, and collect the interpretation, which is in JSON format.

Listing 4.15: Rasa NLU demo

```
from rasa_nlu.model import Interpreter
import json
interpreter = Interpreter.load("./models/current/nlu")
message = "go to the kitchen"
result = interpreter.parse(message)
print(json.dumps(result, indent=2))
```

4.3 LU4R

4.3.1 General description

LU4R is the *adaptive Language Understanding chain For Robots* tool proposed in [Bastianelli et al. [6]]. It relies on the theory of frame semantics [Fillmore [33]] to obtain an interpretation of natural language sentences based on linguistic and cognitive basis. LU4R receives a natural language command as input and outputs an interpretation of the command using a frame semantic representation, similar to the one implemented in FrameNet[Baker et al. [4]]. This representation expresses actions as a set of pairs $\langle f^i, Arg^i \rangle$, where f^i is a semantic frame containing the action desired to be performed and Arg^i is the set of arguments in the sentences, known as frame elements.

LU4R has two modes of operations:

- Basic: It does not take into account the knowledge obtained through perception during the interpretation of the natural language commands.
- Simple:

  Uses perceptual information to perform context-sensitive interpretation of the command at the predicate level. This would allow the model to differentiate between similar semantic frames. For instance, when receiving a command like *take the book on the table* the frame identified by LU4R would be *taking* if there a book was perceived in a table, while the frame would be *bringing* if no book was identified.

It is possible to request the model from [15], but only the compiled version of the model is accessible. Therefore, no modifications can be done to the model, which limits its generalization capabilities. It was pre-trained using the HuRIC corpus [16].

[15] `http://sag.art.uniroma2.it/lu4r.html`

[16] HuRIC corpus was developed by [Bastianelli et al. [5]]. HuRIC is a labeled corpus for HRI using natural language commands oriented to domestic service robots. It contains three datasets: Grammar generated, S4R experiment, and Robocup@home 2013.

The current implementation of LU4R can capture the following frames [17]:

- Arriving (e.g. enter the bathroom).
- Attaching (e.g. connect to the station on your right).
- Being_in_category (e.g. this is a box).
- Being_located (e.g. the tea is in the kitchen).
- Bringing (e.g. bring me a bottle of water from the kitchen).
- Change_direction (e.g. turn left by 45 degrees).
- Change_operational_state (turn on the light).
- Closure (e.g. close the door).
- Cotheme (e.g. follow Alex to the kitchen).
- Giving (e.g. give me the cup).
- Inspecting (e.g. check the table).
- Locating (e.g. find the glasses on the bed).
- Manipulation (e.g. grasp the ball).
- Motion (e.g. go to the kitchen).
- Perception_active (e.g. look at Argentina).
- Placing (e.g. place the mug on the sink).
- Releasing (e.g. drop the bottle in the trash).
- Taking (e.g. take the bottle on the table).

4.3.2 Structure

The interpretation of natural language commands is performed in the Spoken Language Understanding (SLU) chain of the LU4R system. It uses a "*cascade of statistical classification steps modeled as sequence labeling tasks*" [Bastianelli et al. [6]].

The whole SLU chain is composed of four stages, which can be seen in Figure 4.2:

1. Morpho-syntactic analysis:

   POS-tagging and syntactic parsing is performed to every utterance in the received command. This allows to obtain the morphological and syntactic

[17]The examples were created based on the examples presented in the model's website.

information from the command. This stage is implemented using Stanford CoreNLP library [[48]].

2. Reranking:

   It is used to obtain the "*best transcription*" from a set of transcriptions of the command generated by a speech recognition. This is currently disabled for the release given by the authors.

3. Action detection:

   All the frames that are expressed in the command are detected in this stage. It creates a pair of the word that indicates the action with the corresponding semantic frame $\langle w_i, f\rangle$, e.g. $\langle take, Taking\rangle$. All the words that do not express an action are labeled as null (_), e.g. $\langle$*book*, _ $\rangle$. To classify the words during this step, a SVM combined with a Markovian formulation (SVM^{hmm}) is used relying on the features found during the Morpho-syntactic analysis stage.

4. Argument labeling:

   This stage is subdivided into two more steps.

   4.1. Argument identification:

   For each frame found during the action detection, all the argument spans and semantic heads are labeled following the IOB2 notation [18]. For example, the sentence *take the book on the table* would end up as *O-take B-the I-book I-on I-the I-table.* This step also implements a SVM combined with a Markovian formulation (SVM^{hmm}) using the features found during the Morpho-syntactic analysis stage.

   4.2. Argument classification:

   Each of the arguments identified during the step before is classified, taking into consideration the frame identified during the action detection.

[18]The IOB notation format assigns one of following tags to each token I- Internal, B- Begin, or O- Outer

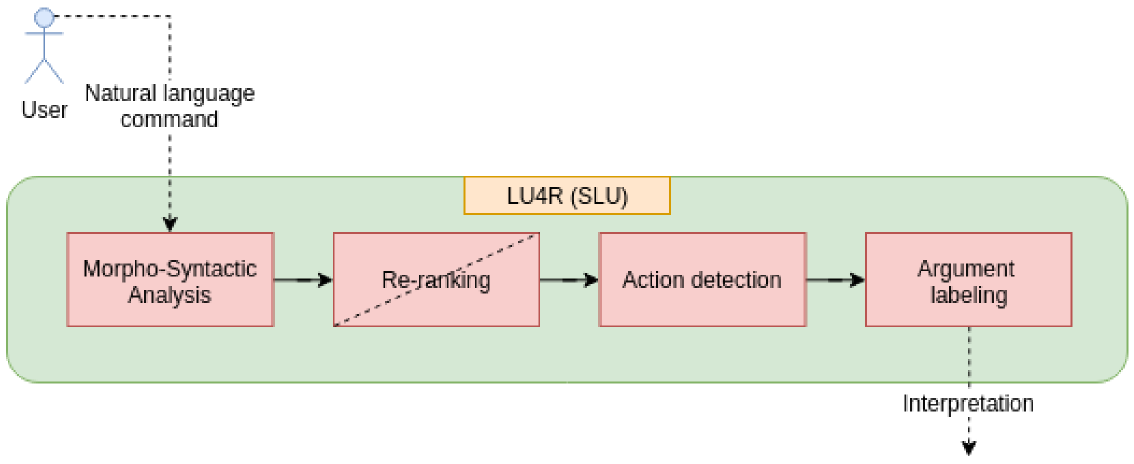


Figure 4.2: Structure of the SLU part of LU4R. Figure adapted from: `http://sag.art.uniroma2.it/lu4r.html`

4.3.3 System requirements

Java 1.8 or higher.

4.3.4 Implementation process

Adaptation

It is not possible to retrain nor adapt the model to a new domain, as it is only accessible as a compiled jar file.

Execution

The usage of LU4R involves two phases:

- Startup phase -

To launch the LU4R service, one needs to execute the following Bash command line 4.16, with the following parameters:

- type: Corresponds to two operating modes of LU4R:
  - Basic.
  - Simple.
- output: Sets the desired output format:
  - xml (default): eXtended Dependecy Graph format and XML compliant container.
  - amr: Abstract meaning representation format. It helps to easily see "*who is doing what to whom*" [19].
  - conll: Tabular representation similar to the one used for CoNLL shared task [Tjong Kim Sang and De Meulder [73]].
  - cfr: Command Frame Representation format used for *RoboCup@Home*[Matamoros et al. [51]].
- lang: It is the language on which LU4R is operating.
  - en: English

[19] `https://github.com/amrisi/amr-guidelines/blob/master/amr.md`

  - it: Italian

- port: Listening port for the http communication with the server
  - e.g. 9090

Listing 4.16: Command to execute the LU4R server

```
java -jar -Xmx1G lu4r-server-0.2.1 [type] [output] [lang] [port]

For example:
java -jar -Xmx1G lu4r-server-0.2.1 basic cfr en 9090
```

The startup process is completed once the message *Server launched: listening on port 9090* is prompted. The server then waits for any incoming command.

- Service phase -

To send a message to the server, it is necessary to execute a Bash command similar to the one shown in 4.17. The components of this type of message are:

- Transcription: The natural language command.
- Confidence: Confidence value assigned by an automatic speech recognition (ASR) module.
- Rank: Ranking position of the hypotheses produced by the ASR.
- Entities: List of objects populating the environment in which the robot is operating. For each entity is necessary to specify the following:
  1. Type: Indicates the class to which each specific entity belongs. E.g. table, book, chair, etc.
  2. PreferredLexicalReference: It is used to name a class of objects. Basically is a string that indicates how to call an object. For instance, a table could be referred to as a desk.
  3. AlternativeLexicalReference: It is used to provide additional labels to the entities.

4. Coordinates: Contains the pose of the objects that exist in the world the robot is working in, using planar coordinates, elevation, the orientation angle.

An example containing the previously mentioned information for the entities can be seen in 4.18.

Listing 4.17: Example of a natural language command request

```
curl -X POST -d "hypo={\"hypotheses\": [{\"transcription\":
\"bring me the book\", \"confidence\":0.9,\"rank\":0}]}&
entities={\"entities\":[]}"http://127.0.0.1:9090/service/nlu
```

Listing 4.18: Example of entities declaration

```
{"entities":[
{"atom":"book1",
"type":"book",
"preferredLexicalReference":"book",
"alternativeLexicalReferences":["volume","manual"],
"coordinate":{"x":"13.0","y":"6.5","z":"3.5",
"angle":"3.5"}},
...
{"atom":"table1",
"type":"table",
"preferredLexicalReference":"table",
"alternativeLexicalReferences":["bar","counter","desk",
"board"],
"coordinate":{"x":"12.0", "y":"8.5", "z":"0.0",
"angle":"1.6"}}]}
```

The interpretation that is generated by LU4R to our example command can be seen in 4.19.

Listing 4.19: Interpretation of the example command with different output formats

```
amr:
        (b1 / bring-Bringing
                :Beneficiary (m1 / me)
                :Theme (b2 / book)
        )

CoNLL:
        bring-VB        Bringing        _
        me-PRP   _      B-Beneficiary
        the-DT   _      B-Theme
        book-NN  _      I-Theme

cfr:
        BRINGING(beneficiary:"me", theme: "the book")
```

4.4 ECG

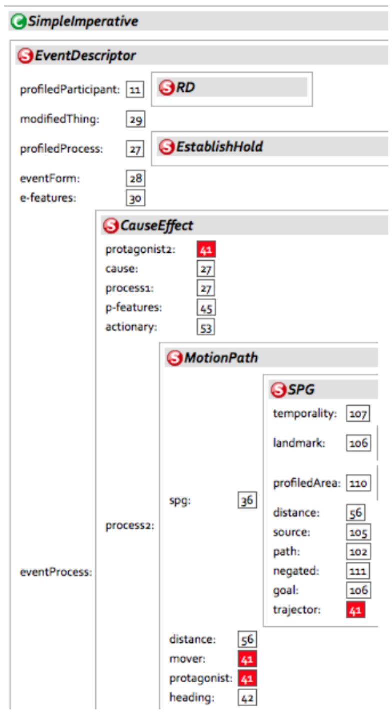
4.4.1 General description

Embodied Construction Grammar (ECG) is a NLU framework that resulted from the work done at UC Berkeley for over 30 years. The latest version, ECG2, was formally presented in [Eppe et al. [29]]. In there, the authors stated that Neural theory of language and cognitive linguistics were the pillars of the framework, and that it was developed as a solution to the problem of inferring the meaning of sentences taking into account the combinations of words and their constructions.

Using the basic principle of construction grammars, which is a way to describe language by creating a tuple of (form, meaning) for different levels, such as phonology, morphology, syntax, and semantics, the framework presents a different semantic formalism to express the meaning of sentences, denominated schemas. Schemas are, as defined by the authors, "*a symbolic description of putative conceptual universals of perception, action and thought*". They are used to join schema roles to grammatical constituents. The schemas are themselves the embodied construction grammars, which are "*a network of rules outlining form and function pairings*". The grammar used for the model is a collection of constructions, schemas, and ontology items that require domain-specific vocabulary to produce a reliable semantic analysis of natural language commands.

The framework uses the schemas to produce a detailed semantic representation denominated semantic specification or SemSpec. The main idea behind the creation of the semantic specifications is to identify the relations of the words in the sentence and from that derive the meaning of it. The analysis that obtaines the SemSpec is expected to be task-independent.

An extract of a SemSpec can be seen in Figure 4.3. The schemas conforming the SemSpec are generated due to certain words in the sentence. For instance, the word *bring*, which is a motion verb, provokes the *EstablishHold* schema, indicating that a grasping action is required. The construction *bring the soda can*, accompanied of the prepositional phrase *to the dining table*, causes the *CauseEffect* schema, containing the *MotionPath* and the *Source Path Goal (SPG)* schemas indicating that a certain trajectory from the current location to the goal location exist.



SimpleImperative

EventDescriptor

- profiledParticipant: 11 — RD
- modifiedThing: 29
- profiledProcess: 27 — EstablishHold
- eventForm: 28
- e-features: 30
- eventProcess: CauseEffect
  - protagonist2: 41
  - cause: 27
  - process1: 27
  - p-features: 45
  - actionary: 53
  - process2: MotionPath
    - spg: 36 — SPG
      - temporality: 107
      - landmark: 106
      - profiledArea: 110
      - distance: 56
      - source: 105
      - path: 102
      - negated: 111
      - goal: 106
      - trajector: 41
    - distance: 56
    - mover: 41
    - protagonist: 41
    - heading: 42

Figure 4.3: Extract of a semantic specification for the command *PR2, bring the soda can to the dining table.* Figure source: [Eppe et al. [28]]

The current version of the framework can interpret the following kind of sentences:

- Conceptual and constructional compositionality: Catching the meaning of the sentence based on how is constructed.
- Conditional sentences: *If there is a glass on the table, please bring it.*
- Clarification and dialogs: *A: Pick up the marker – B: Which marker?*
- Indirect assertions via relative sentences or appositions: *Please bring the bottle, which fell from the table, to the dishwasher.*
- Modalities: *Are you able to order pizza?*.

ECG2 can be obtained through the following sources:

- ECG_framework_code: [20] Contains the core framework code for the ECG model.
- ECG_robot_code: [21] Contains the code necessary to implement the ECG model to robotic applications
- ECG_grammars: [22] Contains the grammar files, associated ontologies, and preferences files.
- ECG_Workbench: [23] ECG workbench is a tool provided by the authors to ease the work of viewing, testing and modifying the grammars used in the framework. It was developed as an eclipse rich-client platform based project editor.

4.4.2 Structure

Figure 4.4 shows the structure of the ECG2 framework. The main modulesthat are used for NLU are:

1. UI-Agent:

   This module is responsible for the interaction with the user, receiving inputs from it and communicating the understood set of actions. It contains two submodules:

   1.1. ECG Analyzer:
   It is the one responsible to parse the inputs received from the user using a specific grammar and to generate the semSpecs. It uses a Bayesian best-fit approach to parse the sentences.

   1.2. Specializer:
   It is responsible of receiving a semSpec and creating action specifications as output in a format named n-tuple. Action specifications are feature

[20] `https://github.com/icsi-berkeley/ecg_framework_code`
[21] `https://github.com/icsi-berkeley/ecg_robot_code`
[22] `https://github.com/icsi-berkeley/ecg_grammars`
[23] `https://github.com/icsi-berkeley/ecg_workbench_release`

structures that contains the actions inferred from the commands. It uses a set of templates to find the relevant information embedded in the Semspec. Some of the templates currently implemented are mood templates (to deal with yes/no questions and wh questions), event templates (to deal with modality information, temporal information, and conditional commands), parameter templates (to deal with possession or spatial relations in the sentences), descriptor templates (to capture information such as gender, category or other object properties). An example of an action specification is shown in 4.20. In the action specification, we can find relevant information produced when inferring the command *Robot1, pick the marker!*, such as the schema Manipulation, with the predicate_type command, and the actionary pickup.

2. Problem Solver:

   It is in charge of unpacking and routing the n-tuples generated by the UI-agent. It requires a definition of the world in which the robot is going to be interacting on for calling the actions encoded in the n-tuple message. It contains different algorithms capable of performing tasks such as motion planning or action planning. We are not going to use this, as we are only interested in the interpretation aspect of the model.

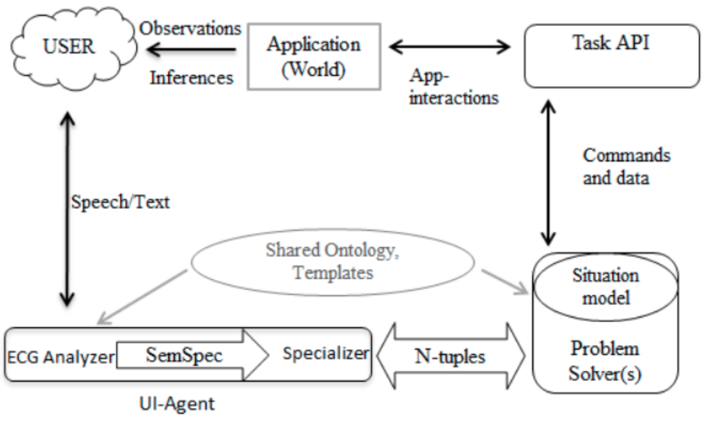


Figure 4.4: Structure of the ECG2 framework. Figure source: [Raghuram [66]]

Listing 4.20: Action specification of the sentence Robot1 pick the marker

```
{'eventDescriptor': {'eventProcess': {'manipulated_entity':
{'objectDescriptor': {'gender': 'genderValues',
'givenness': 'uniquelyIdentifiable', 'number': 'singular',
'type': 'marker'}}, 'schema': 'Manipulation',
'protagonist': {'objectDescriptor': {'gender': 'female',
'givenness': 'givennessValues', 'number': 'singular',
'referent': 'robot1_instance', 'type': 'robot'}},
'actionary': 'pickup', 'p_features': {'processFeatures':
{'voice': 'active'}}, 'template': 'Manipulation'},
'profiledParticipant': {'objectDescriptor': {'gender':
'female', 'givenness': 'givennessValues', 'number':
'singular', 'referent': 'robot1_instance', 'type':
'robot'}}, 'e_features': None}, 'speechAct': 'direct',
'predicate_type': 'command', 'return_type': 'error_descriptor'}
```

4.4.3 System Requirements

- Python 2.7 or higher
- Pyre [24]
- Jython [25]
- Six [26]
- Java (No version specified)

4.4.4 Implementation process

Adaptation

In order to adapt the model to a new domain, it is necessary to add new vocabulary to the grammar file. One of the best ways to do this is to use the ECG

[24]https://github.com/zeromq/pyre/archive/master.zip
[25]http://www.jython.org/
[26]https://pypi.org/project/six/

workbench [27] tool provided by the authors. The tool has an option to add tokens specifying the lemma, parent type, ontology file, token file, and the role or action of the token. Knowing the proper parent type for each of the words is not trivial as it requires advance knowledge of linguistics to properly differentiate between the different grammatical categories, e.g. property nouns, container nouns, stage verb types, etc. We believe this is one of the most important aspects of the model as it plays a major role when inferring the meaning of a sentence.

Execution

First, the analyzer needs to be launched. In order to do so, the shell file *analyzer.sh* is executed. This script specifies the federation required for the communication protocol and uses Jython to call the jar file where the analyzer is encoded, specifying the grammar to be used.

In a different console, the shell file *setup.sh* is executed to specify the federation and launch the UI-Agent, indicating the grammar to be used and the name of the agent and the text agent. This console is where the user is prompted for the sentences. In order for the model to correctly identify the received sentence as a command, the character "!" needs to be added at the end of the sentence, otherwise, it cannot be processed. It was necessary to modify the *user_agent.py* file to have access to the action specification produced by the specializer. Otherwise, we could not visualize the interpretations generated by the model.

The problem solver could also be executed through the *setup.sh* file, but requires the definition of the world model and the algorithms that are going to be required for the supported actions.

[27] `https://github.com/icsi-berkeley/ecg_workbench_release`

4.5 Features

Table 4.4 presents an overview of the different features each of the models have. Clarifying some of the terminology here used, monologues refer to sentences that contain more than one action. The ready to use column expresses the fact that the authors have provided with pre-trained models capable of dealing with a set of natural language commands. The linguistic knowledge required to train or adapt the models to new domain is also expressed as a feature. A high level means that in order to obtain the full potential of the model, it is necessary to be almost an expert in linguistic, while a moderate level indicates that it is possible to implement the model in a new domain without much linguistic knowledge. LU4R is not applicable to this feature, as it is not possible to adapt it to new domains.

The features here presented make Rasa and Mbot quite attractive options. Both, present customizable action and argument labels, meaning that you can set the models to identify almost any actions and entities desired. It is also possible to change the interpretation format and used it to build a planner to execute actions. They are capable of ignoring unnecessary information in the commands, such as intros and capturing entities composed of more than one word, similar to LU4R. For the case of Mbot, it is a great aspect that the model can be used out of the box to understand a great variety of actions. While for Rasa, it is quite appealing that it can be used in almost any language.

Regarding LU4R, it would be better if it was possible to retrain the model and adapt it to new domains. Nonetheless, it already understands different kinds of commands and complementary information. It is also a nice feature to have, the fact that you can choose from a predefined set of output formats to the one most suitable for your needs. Additionally, similar to Mbot, it can deal with sentences containing multiple actions in them.

The major drawback we can appreciate from ECG is the required of high level linguistic knowledge to properly use the model. Also, it is quite unreasonable that the model can not understand entities composed of more than one word, like *living room* or *chocolate bar.*

Models	Customizable action labels	Customizable action arguments labels	Customizable output format	Support sentences with intros	Support multiple word entities	Support monologues	Ready to use	Language	Programming language	Used for robots	Linguistic knowledge required to adapt
Mbot	☑	☑	☑	☑	☑	☑	☑	English	Python	☑	Moderate
Rasa	☑	☑	☑	☑	☑	☒	☒	Almost any	Python	☒	Moderate
LU4R	☒	☒	☒	☑	☑	☑	☑	English, Italian	Java	☑	NA
ECG	☑	☑	☒	☑	☒	☒	☒	English	Java, Python	☑	High

Table 4.4: Features of the chosen NLU models. ☑ represents yes, ☒ represents no.

5

Methodology

5.1 Setup

The four models, presented in 4, are going to be implemented in a laptop Asus ROG GL552V with an intel core i7 processor, 12 GB of RAM, and 130 GB of SSD running Ubuntu 16.04.

Based on the results obtained from the experiments, we are going to select the best model and deploy it in Lucy 5.1 to have the develop a complete process of providing a voice natural language command, interpreting the command, create a plan of the required actions and execute them.

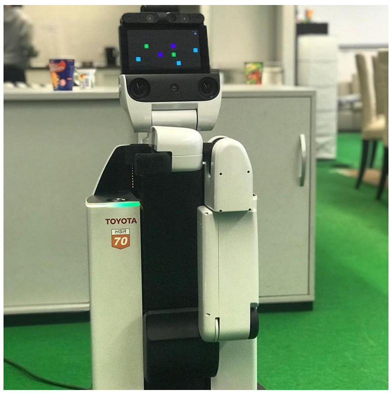


Figure 5.1: Lucy is our local B-it-bots Human Service Robot developed by Toyota. Image obtained from the team's Facebook page.

5.2 Use cases

5.2.1 General Purpose Service Robot

The *RoboCup@Home*[Matamoros et al. [51]] competition is one of the major competition that encourages the development of service and assistive robot technology in the context of personal and domestic applications. The competition is divided into three leagues:

(1) Domestic Standard Platform League (DSPL):

Focuses on the assistance of humans in domestic environments, mainly in the context of elderly and people with illness or disabilities.

(2) Social Standard Platform League (SSPL):

Focuses in the HRI aspect from a more active perspective, not just the passive servant role of the robot.

(3) Open Platform League (OPL):

Allows the implementation any kind of robot to solve domestic tasks.

One of our use cases is going to be the General Purpose Service Robot (GPSR) task that belongs to the fist league of the *RoboCup@Home* competition. During the GPSR task, a natural language command, randomly generated, is given to the robot indicating an action (or set of actions) that it needs to execute to test the abilities of the robot to perform in a service environment. The generated command could belong to one of the following categories:

- Category 1:

  Low degree of difficulty concerning actions like go, take, find, answer, and tell.

- Category 2:

  Moderate degree of difficulty including actions like go, take, find, tell, follow, guide, and meet. The commands in this category are more in context of HRI.

- Category 3:

  High degree of difficulty, including commands with incomplete/erroneous information.

5.2.2 Ropod

As another use case, we have the Ropod project [1]. Ropod seeks to implement multiple robotic platforms in a hospital environment. The idea is to aid with the transportation of beds around the hospital in an efficient manner. We decided to explore the possibility of using natural language commands, in the context of the hospital environment, to move the platforms around. We find this use case to be useful to provide a better perspective of the generalization capabilities of the model as it is not a domestic service environment.

5.3 Experimental design

5.3.1 Datasets

Three datasets are going to be used to evaluate all the NLU models, mentioned in 4. One for GPSR category 1, one for GPSR category 2, and one for Ropod.

The datasets for category 1 and 2 were created with the help of the command generator tool [2]. We started by generating a random set of 10,000 sentences for each category and pre-processed the sentences by removing those that were not commands and converting them to lowercase. For each category, we chose a total number of 110 random sentences making sure of covering all the different actions involve. We organized the sentences of each dataset in two inputs files, one containing single action sentences and one containing multiple action sentences. In those files, the sentences were organized in groups based on the action behind the command for the single action sentences, and based on the number of actions for the multiple actions sentences.

[1]`http://www.ropod.org/`

[2]`https://github.com/kyordhel/GPSRCmdGen`

GPSR category 1 files were split as follows

- Single action commands: (86) [3]
  - Go: (11)
  - Take: (18)
  - Find: (20)
  - Answer: (3)
  - Tell: (34)
- Multiple actions commands: (24)
  - 2 actions (14)
  - 3 actions (10)

GPSR category 2 files were split as follows [4]:

- Single action file: (80)
  - Follow: (20)
  - Guide: (10)
  - Tell: (20)
  - Take: (10)
  - Find: (10)
  - Meet: (10)
- Multiple actions: (30)
  - 2 actions (10)
  - 3 actions (20)

The dataset for Ropod was built by manually creating a total number of 97 sentences. We included commands that we believe were suitable in the context of a hospital environment. We split the sentences in a similar manner to the previous two datasets, as follows:

- Single action: (71)
  - Go (20)
  - Attach (8)
  - Detach (8)
  - Push (8)
  - Find (10)
  - Guide (7)
  - Follow (10)
- Multiple actions: (26)
  - 2 actions (20)
  - 3 actions (6)

For each single and multiple actions file, we manually developed an outputs file containing all the expected interpretations of the sentences following the interpretation

[3]The number in parentheses indicates the number of sentences.

[4]We decided not to include single *Go* actions sentences as they were already covered in the dataset for GPSR category 1 and they are included in the multiple action sentences.

format presented in 4.1. We found this format to be quites useful as it displays, in a clear and simple way, the intention behind the sentences and the complementary arguments.

5.3.2 Evaluation metrics

Similar to the benchmark presented in [Vanzo et al. [75]], we are going to evaluate different aspects of the NLU models:

- Action classification (AC):

  Measures the ability of the models to perform correct detection of the actions in the sentences. AC will be measured through **precision** 2 ("*percentage of correctly identified actions among all actions found*"), **recall** 3 ("*percentage of correctly identified actions with respect to all the actions in the labeled dataset*"), and **F1 score** or **F-measure** 4 ("*harmonic mean between precision and recall*") [5].

- Full command recognition (FCR):

  Measures the ability of the models to understands the commands completely, i.e. recognizing the correct actions and complementary information. FCR will measured through **accuracy**, calculated using the equation 1, which was obtained from [Martins et al. [49]].

- Run-time:

  Measures the time required for the models to process the complete dataset. The values are going to be expressed in seconds.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}^{6} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

[5]The equations to calculate these metrics were obtained from [Chesworth et al. [18]] and the definitions from [Vanzo et al. [75]]

[6]TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4}$$

The results obtained from our experiments are going to be reported using a table similar to the one shown in 5.1, one for single action sentences and one for multiple action sentences.

Single action sentences															
Metrics	Action classification									Full command recognition			Run-time		
	Precision			Recall			F1			Accuracy					
Datasets / Models	Cat 1	Cat 2	Ropod	Cat 1	Cat 2	Ropod	Cat 1	Cat 2	Ropod	Cat 1	Cat 2	Ropod	Cat 1	Cat 2	Ropod
Mbot															
LU4R															
Rasa															
ECG															

Table 5.1: Experiment results table format

5.3.3 Experiments description

The experiments are going to evaluate the capabilities of the NLU models, presented in Chapter 4, to interpret natural commands in the context of domestic service robots and hospital service robots. There will be a total of 3 experiments, one for each use case: GPSR category 1, GPSR category 2, and Ropod. The models are going to receive one command at a time, extracted from one of the dataset files, to identify the action, or actions, desired to be executed, as well as the necessary complementary information to fulfill the command and generate an interpretation of the received command. The interpretations generated are going to be converted into the expected interpretation format, similar to the one used by Mbot, and we are going to compare them against our manually labeled datasets to compute the evaluation metrics, previously explained in 5.3.2.

5.3.4 Assumptions

- All the models are going to be evaluated on the same set of natural language robot commands.

- The models receive the commands exactly as they are written in the dataset file, without any alteration during the sending process.
- There are no repeated sentences in the datasets.
- The models have already been trained and prepared for the evaluation.

5.3.5 Experimental procedure

1. Specify the use case desired to be evaluated in the *tester.py* file.
2. Launch LU4R and ECG, following the procedure explained in the implementation process in Chapter 4.
3. Execute *tester.py*. The script does the following:

   3.1. It uses the *read_sentences* and *read_expected_values* methods from *reader.py* script to extract the sentences from the inputs datasets file, as well as the expected outputs from the outputs dataset file.

   3.2. For each model, the single action sentences are first analyzed, one at a time. Once the single sentences analysis is done, the script continues with the multiple action sentences. The analysis procedure for each single sentence goes at follows:

   - Call the evaluation method corresponding to the model currently being analyzed, i.e. *test_mbot_nlu, test_lu4r_nlu, test_ecg_nlu, and test_rasa_nlu* , passing the input sentence and the expected output as arguments.
     1. i. The phrases that compose the input sentence are identified.
     2. ii. Each phrase is sent to the NLU model.
     3. iii. The interpretation for each phrase is received. And, for the case of LU4R, ECG, and Rasa, the interpretation is translated into the expected output format.
     4. iv. All the translated phrases are combined into a single sentence and compared against the expected output.

3.3. Once all the single action, or multiple action, sentences in the dataset files have been compared, the evaluation metrics are computed and displayed in the console so that we can manually add the results to the corresponding report table.

5.3.6 Technical details

All the evaluation scripts have been coded using *python 3.5*.

6 Solution

6.1 Training and adaptation

Three out of the four models presented in chapter 4 required training and adaption for our use cases. The only model we could not train was LU4R, due to the fact that we could only obtain the pre-trained version of it. This pre-trained version was itself good enough to understand most of the actions that were expected in our datasets.

6.1.1 Mbot

Following the training procedure explained in Section 4.1, we generated new set of *inputs* and *outputs* files containing the training sentences and labels for our datasets, one for GPSR category 1 and category 2 with *200,000* commands, and one for Ropod with *199,997* commands. It was necessary to train new classifiers because the pre-trained classifiers provided by the authors could only understand the commands that involved GPSR category 1, but not category 2, nor the new set of commands necessary for Ropod.

The hyper-parameters used to train the intention classifiers are shown in 6.1. We kept most of the parameters that were already tuned by the authors and changed only the number of validation batches and the total number of batches. The training process for both of the intention classifiers required only 2 out of 500 epochs to reach a loss of 0 and a validation accuracy of 100%.

The hyper-parameters used to train the slots classifiers are shown in 6.2. Similar to the intention classifiers, we only changed the number of validation batches and the total number of batches. The slot classifier for GPSR trained for 55 epochs, reaching a loss of 0.712 and a validation accuracy of 99.999%. The slot classifier for Ropod trained for 38 epochs, getting a loss of 3.738 and a validation accuracy of 99.995%.

All the classifiers were trained in a GPU GeForce GTX 1070 with 8GB of memory.

Listing 6.1: Hyper-parameters of the intention classifiers for GPSR and Ropod

```
number_of_batches: 50
number_of_validation_batches: 25
forget_bias: 1.0
learning_rate: 0.001
n_lstm_layers: 2
n_epochs: 500
n_steps: 15
rnn_size: 500
batch_size: 1
embedding_size: 300
```

Listing 6.2: Hyper-parameters of the slots classifiers for GPSR and Ropod

```
batch_size: 1
number_of_batches: 50
number_of_validation_batches: 25
forget_bias: 1.0
learning_rate: 0.001
n_lstm_layers: 2
n_epochs: &n_epochs 500
n_steps: 15
rnn_size: 500
embedding_size: 300
n_parallel_iterations_bi_rnn: 32
```

6.1.2 Rasa NLU

The markdown file required to train Rasa NLU was developed using a similar script to the one used to generate the inputs file for Mbot. We generated a total number of *200,000* training sentences for GPSR, both categories combined, and *199,997* for Ropod. We used the default hyper-parameters for the model because the documentation did not present any recommendation on how to change them or optimize them. Additionally, we chose the pre-configured tensorflow_embedding pipeline for Rasa NLU because our datasets contained more than 1000 sentences and we thought that generating word embeddings for our particular domains would obtain better results than using pre-trained word embeddings. The training process for the GPSR dataset ran for a total of 300 epochs, reaching a loss of 0.007 and a training accuracy of 100%. The model trained for Ropod ran as well for 300 epochs and obtained a loss of 0.004 and a training accuracy of 100%. Both models were trained on a CPU intel core i7.

6.1.3 ECG

Due to the nature of the ECG framework, it was not necessary to train it, but to add new vocabulary to the grammar file. In order to do so, we took advantage of the ECG_workbench [1] tool provided. This resulted to be quite a challenge, because the documentation of the tool only covered a simple example on how to add tokens that represent verbs to the grammar, but not objects or locations. Furthermore, it was not clear how to choose the proper grammatical category for some of the words, which might affect the performance of the model when interpreting natural language commands. Also, it was not possible to add tokens that corresponded to two words objects or locations, like *living room* or *towel rail.*

As explained in more detail in Section 4.4, the interpretation produced by the model that contains the information about actions in the commands is the action specification, named n-tuple in the code of the model. In the original implementation ECG, it is not possible to directly obtain the interpretation produced by the model

[1]`https://github.com/icsi-berkeley/ecg_workbench_release`

when receiving a natural language command. Therefore, It was necessary to modify the source code to obtain the action specification every time the model processes a command.

7 Results

The results obtained from the experiments using the dataset files with single action sentences are reported in Table 7.1 and plotted in Figure 7.1. The results showed in Table 7.2, and plotted in Figure 7.2, corresponds to the dataset files containing multiple action sentences. The values for precision, recall, F1-score, and accuracy corresponds to percentages in a scale from 0 to 1.0, 0 meaning 0% and 1.0 meaning 100%.

7.1 Action classification

As explained before, we measured the ability of the models to correctly classify the actions or intentions behind the given commands using precision, recall and F1 score. Precision is a metric that represents the percentage of actions that were correctly classified from all the actions that the models were able to classify, while recall represents the percentage of actions that were correctly classified with respect to all the actions contained in the datasets, i.e. all the actions the models should have classified. A value of 1.0 in precision indicates that no false positives were presented and a value of 1.0 in recall indicates that no false negatives were obtained. F1 gives equal importance to both precision and recall and represents the balance between them.

7.1.1 Precision

From the results shown in both tables, we can appreciate that both Mbot and Rasa obtained better precision than LU4R and ECG, in most of the use cases. LU4R obtained the worst precision for all the datasets because it misclassified commands involving actions like *tell, guide and detach.* ECG had a decent performance in terms of precision because most of the time it did not understand a command, it did not give any classification at all. On the contrary, Mbot basically guessed the actions behind some of the commands for Ropod, because it did not have vector representations in its dictionary for some of the words used to indicate *detach* actions.

7.1.2 Recall

In terms of recall, Mbot and Rasa obtained a full score for all the datasets. This means that they were able to provide an interpretation for all the commands. The really low recall obtained by ECG shows that from all the commands, very few were actually classified. The results obtained by LU4R shows that it struggled to provide an interpretation to commands that contained actions not covered by the semantics frames on which it was already trained.

7.1.3 F1 score

The F1 scores give us more insight about the actual ability of the models to classify the actions behind the given commands. We can see that for GPSR category 1 and 2, both Mbot and Rasa had an almost perfect score, meaning that they were able to always interpret the actions when receiving a command, and that the interpreted action was correctly classified. For the case of Ropod, Rasa outperformed Mbot getting a perfect score, while Mbot misclassified some of the actions. This showed that the use of pre-trained word embeddings on Mbot limits it to only understand words that are already present in its dictionary, and because Rasa created its own word embeddings during training, it was able to understand not common words like *uncharge or undock.* ECG obtained the worst results on all the datasets, showing that it is not a good model to classify actions. The results obtained by LU4R were better than expected taking into account the fact that it was not possible to adapt nor retrain the model to our domains.

7.2 Full command recognition

Using full command recognition as a metric allows us to perceive the abilities of the models to capture the complementary information behind the commands. We use accuracy to measure how capable the models are able to understand the complete natural language robot commands. In this case, Mbot outperformed all the models for GPSR category 1 and 2, and Rasa obtained the best results for Ropod. Rasa had problems differentiating between source and destination entities which cause the model to not perform well on the multiple sentences dataset for GPSR category 1. Rasa did better than Mbot in the Ropod datasets due to the reasons mentioned before for using pre-trained word embeddings.

The results obtained for ECG showed that it failed to understand complete commands. The major reason for this is the inability of the model to understand two words tokens, which appear quite frequently in the datasets. LU4R failed mostly because it does not support a some of the commands that were presented in the datasets and it struggled with complementary information involving people and certain locations.

7.3 Run-time

We also measured the time required for the models to process the complete dataset files, expressing the run-time in seconds. From the results, we can appreciate that Rasa was the fastest model to process all the datasets. This could be due to the shallow structure presented for the CRF used to perform NER and the embedding classifier used to identify the intentions. Both Mbot and LU4R were considerably slower than Rasa, with LU4R presenting a shorter Run-time than Mbot for GPSR category 2 and Ropod, but only because LU4R could not understand some of the commands given in them, so no interpretation time was spent for those. In the case of GPSR category 1, where LU4R was able to understand most of the commands, it showed a slower performance than Mbot. This could be caused due to the HTTP communication used to send a sentence to LU4R.

We could not measure the run-time for ECG because of the communication process between the ECG analyzer and the Agent UI. The model presented a delay when a sentence was sent to the analyzer and when it received the response containing the interpretation. Also, the model failed to interpret the commands when they were sent automatically, because in some cases the analyzer failed to receive the commands and in others it gave the interpretation from previous commands instead of the current one. Therefore, it was necessary to manually send the commands in the datasets one by one and verifying that the interpretation generated coincided with the command sent.

Single action sentences															
Metrics	Action classification									Full command recognition			Run-time		
	Precision			Recall			F1			Accuracy					
Models \ Datasets	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod
Mbot	**1.0**	**0.97**	0.92	**1.0**	**1.0**	**1.0**	**1.0**	**0.98**	0.96	**1.0**	**0.75**	0.77	3.85	3.36	4.23
LU4R	0.71	0.57	0.74	0.61	0.57	0.53	0.66	0.58	0.62	0.41	0.29	0.35	4.95	2.89	1.82
Rasa	**1.0**	0.94	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**0.98**	**1.0**	0.90	0.66	**0.89**	**0.14**	**0.17**	**0.11**
ECG	**1.0**	0.83	0.95	0.14	0.06	0.29	0.24	0.12	0.44	0.08	0.05	0.21	NaN	NaN	NaN

Table 7.1: Results table for the experiments done with single action sentences dataset files. The values in blue represents the best values obtained for each metric on each dataset file.

Multiple action sentences															
Metrics	Action classification									Full command recognition			Run-time		
	Precision			Recall			F1			Accuracy					
Models \ Datasets	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod	Cat1	Cat2	Ropod
Mbot	**1.0**	**1.0**	0.97	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	0.98	**1.0**	**0.83**	0.73	2.10	4.02	3.56
LU4R	**1.0**	0.83	0.89	0.67	0.48	0.57	0.80	0.60	0.68	0.17	0.07	0.15	1.62	1.34	0.91
RASA	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	0.58	0.80	**0.81**	**0.06**	**0.14**	**0.09**
ECG	**1.0**	**1.0**	0.96	0.25	0.23	0.40	0.40	0.36	0.56	0.04	0	0.15	NaN	NaN	NaN

Table 7.2: Results table for the experiments done with multiple actions sentences dataset files. The values in blue represents the best values obtained for each metric on each dataset file.

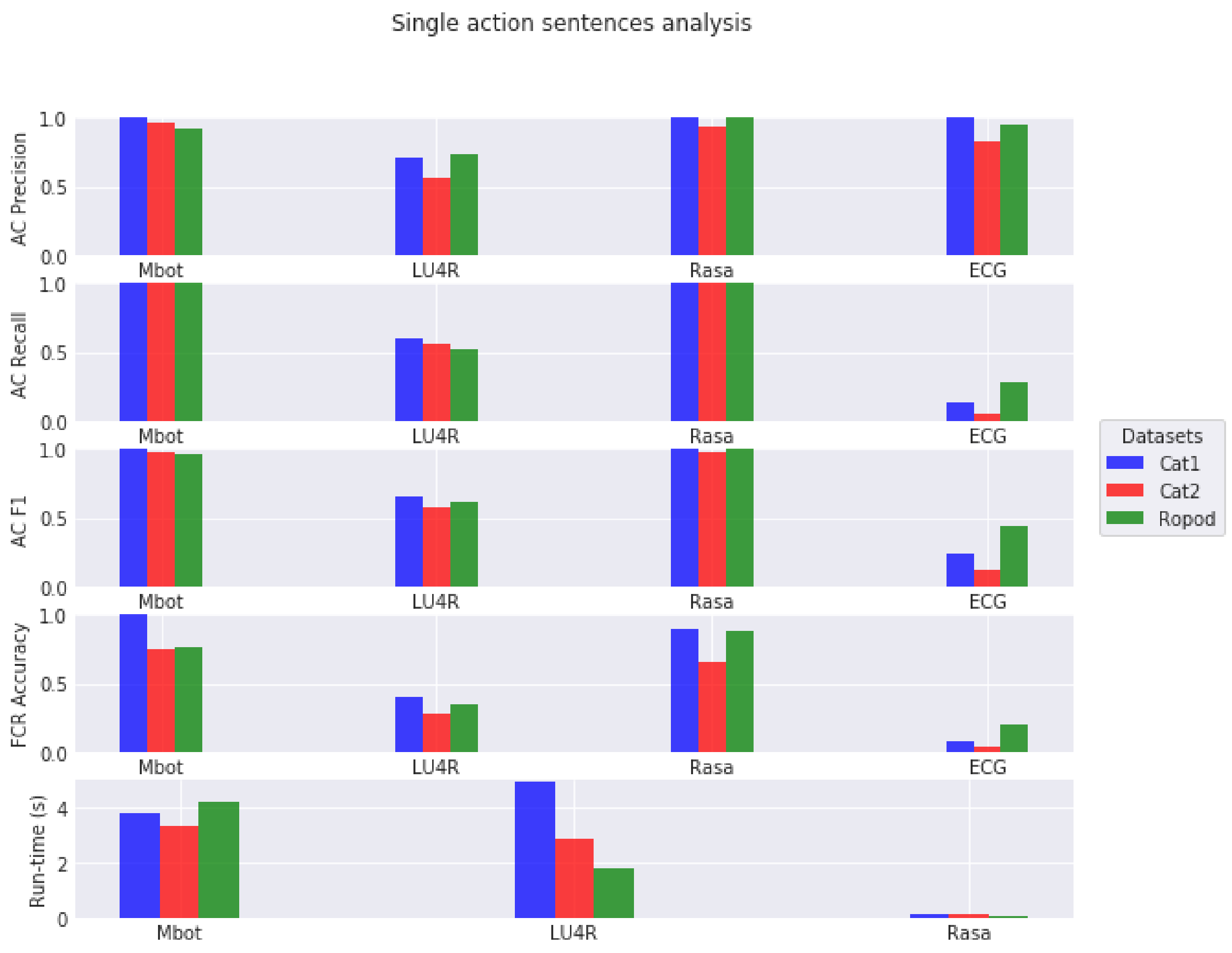


Figure 7.1: Results of the 4 NLU models for our use cases concerning single action natural language robot commands.

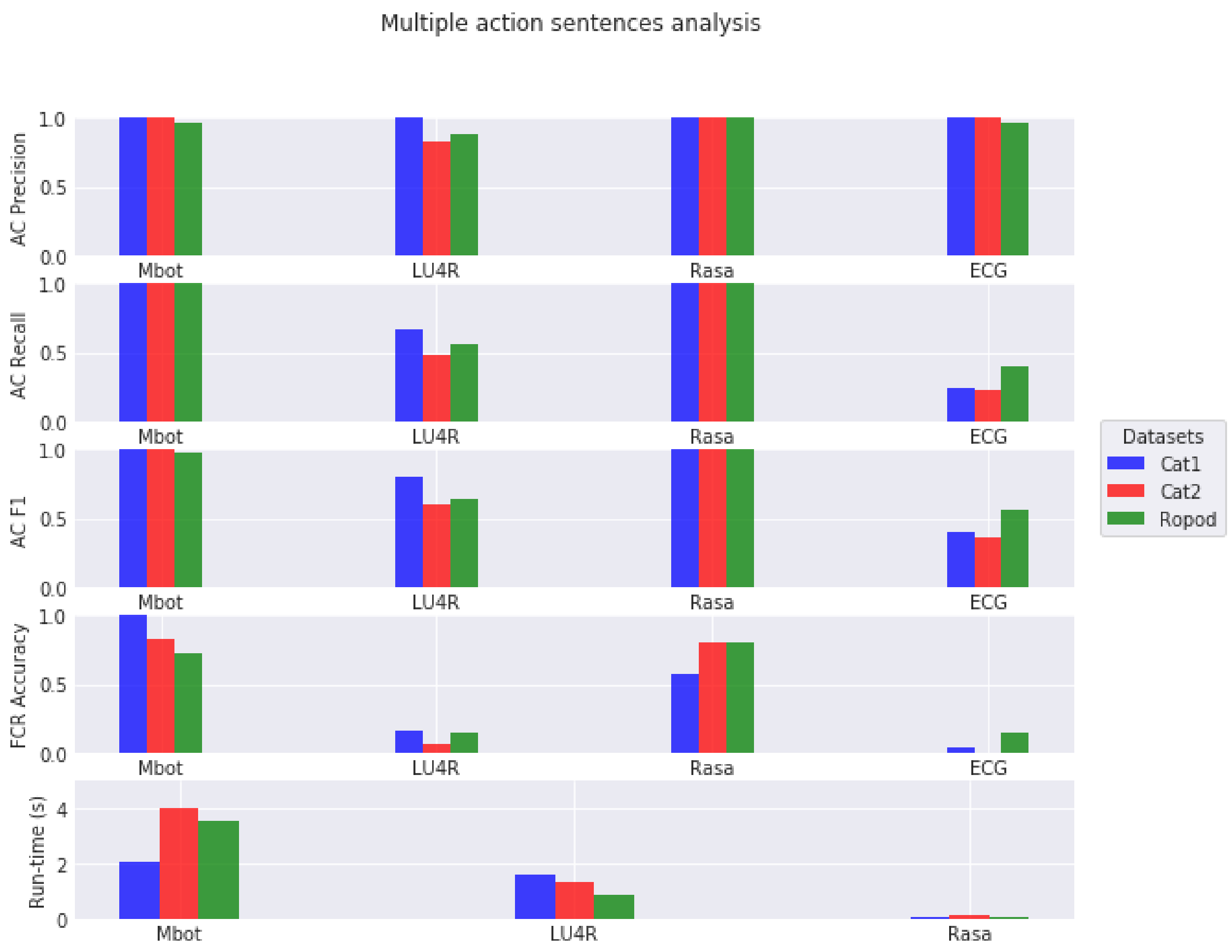


Figure 7.2: Results of the 4 NLU models for our use cases concerning multiple actions natural language robot commands.

8 Conclusion

8.1 Contributions

With this work, we have presented a survey of the state of the art for NLU models and approaches that can be used to process natural language robot commands. We explored the different approaches in the state of the art and found open source options that could be used for to command domestic service robots. We performed a comparative analysis of a selected set of open source models, explaining the different features each of them has, the requirements needed to use them, how to train them or adapt them to new domains and how to implement them.

We evaluate the selected models on three use cases: GPSR category 1, GPSR category 2, and Ropod. The datasets for these use cases contained sentences with single and multiple actions. They allowed us to test the abilities of the models to interpret natural language commands that are normally used in a domestic service scenario. The results of the experiments were quite satisfactory in the sense that two of the models, Mbot and Rasa, were capable of understanding most of the commands and proved to be viable options.

Based on the results obtained in the experiments and the analysis done in our comparison we can conclude that Mbot is the best model to understand natural language robot commands and it is going to be use by our local *B-it-bots* team for the GPSR competition. We chose Mbot over Rasa because of the problems Rasa faced when differentiating location entities from *destination* and *source* categories.

Nonetheless, we have thought some approaches to overcome this problem and take advantage of the tremendous speed Rasa offers compared to the interpretation time required for Mbot.

8.2 Lessons learned

Working in this project has taught us how proper research is actually done, from collecting and reading a great number of papers, to the process of designing experiments and explaining the results obtained. It was quite interesting to learn about the field of NLU and domestic service robots. Additionally, it was also surprising the amount of work that has been done in this field and all the achievements that have been obtained throughout the years.

8.3 Future work

Even though both Mbot and Rasa obtained great results for almost all the experiments and proved the usefulness of NLU systems to process robot commands, there are things that could be done to improve their interpretation abilities, but were not possible to do due to time constraints. We suggest the following things as future work:

- Train custom word embeddings to adapt Mbot better to new domains, similar to Rasa.
- Use different word embeddings representations like *FastText* to evaluate if it improves the understanding capabilities of Mbot.
- Experiment with different hyper-parameters for the intention and slot classifiers for Mbot to seek improvements.
- Include certain prepositions when labeling entities in the training dataset of Rasa to improve the classification and help to solve the uncertainty to differentiate between destination and source.
- Implement an approach to take into account the grounded information obtained from a semantic map, similar to LU4R and helps to solve ambiguity in the interpretation.

- Explore the use of dialogue manager to allow corrections in the interpretations.
- Implement google Syntaxnet [1] to properly split the sentences into phrases, each containing an action desired to be executed.
- Implement a pronoun resolution approach to properly identify the object and people that is being identified withe use of pronouns in the commands.
- The next step to actually execute the actions after generating a proper interpretation of the commands is to develop a planner capable of generating the sequence of actions to perform the actions behind the commands. We have already started to implement a basic planner capable of performing primitive actions like *go* and *take* using hard coded set of conditions. This needs to be improved and explore the use of a knowledge base, similar to the one presented in PRAC [Nyga et al. [60]]. The ultimate goal would be to properly generate action plans when receiving primitive and non-primitive actions.

[1] `https://opensource.google.com/projects/syntaxnet`

A

Acronyms

AC $\hookrightarrow$ Action Classification

ADCG $\hookrightarrow$ Adaptive Distributed Correspondence Graph

AI $\hookrightarrow$ Artificial Intelligence

CNN $\hookrightarrow$ Convolutional Neural Network

CFR $\hookrightarrow$ Command Frame Representation

CRF $\hookrightarrow$ Conditional Random Field

DCG $\hookrightarrow$ Distributed Correspondence Graph

DRL $\hookrightarrow$ Deep Reinforcement Learning

DSPL $\hookrightarrow$ Domestic Standard Platform League

ECG $\hookrightarrow$ Embodied Construction Grammar

ERL $\hookrightarrow$ European Robot League

FCR $\hookrightarrow$ Full Command Recognition

FFNN $\hookrightarrow$ Feed-Forward Neural Network

G^3 $\hookrightarrow$ Generalized Grounding Graph

GPSR $\hookrightarrow$ General Purpose Service Robots

GSMN $\hookrightarrow$ Grounded Semantic Mapping Network

GUI $\hookrightarrow$ Graphical User Interface

HDCG $\hookrightarrow$ Hierarchical Distributed Correspondence Graph

HMM $\hookrightarrow$ Hidden Markov Model

HRI $\hookrightarrow$ Human Robot Interaction

IFR $\hookrightarrow$ International Federation of Robotics

ILLG ↪ Interactive Learning through Language Games
LSTM ↪ Long Short Term Memory
MDP ↪ Markov Decision Process
MDCG ↪ Monologic Distributed Correspondence Graph
MEMM ↪ Maximum Entropy Markov Models
NER ↪ Named Entity Recognition
NLI ↪ Natural language Interpretation
NLP ↪ Natural Language Processing
NLU ↪ Natural Language Understanding
OOMDP ↪ ObjectOriented Markov Decision Process
OPL ↪ Open Platform League
POS ↪ Part Of Speech
RNN ↪ Recurrent Neural Network
ROS ↪ Robotic Operating System
SDC ↪ Spatial Directional Clauses
SLU ↪ Spoken Language Understanding
SSPL ↪ Social Standard Platform League
SVM ↪ Single Vector Machine

References

[1] Hyemin Ahn, Sungjoon Choi, Nuri Kim, Geonho Cha, and Songhwai Oh. Interactive Text2Pickup network for natural language based human-robot collaboration. *arXiv preprint arXiv:1805.10799*, 2018.

[2] Jacob Arkin, Matthew R Walter, Adrian Boteanu, Michael E Napoli, Harel Biggie, Hadas Kress-Gazit, and Thomas M Howard. Contextual awareness: Understanding monologic natural language instructions for autonomous robots. In *Robot and Human Interactive Communication (RO-MAN), 2017 26th IEEE International Symposium on*, pages 502–509. IEEE, 2017.

[3] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Edward C Williams, Mina Rhee, Lawson LS Wong, and Stefanie Tellex. Grounding natural language instructions to semantic goal representations for abstraction and generalization. 2017.

[4] Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

[5] Emanuele Bastianelli, Luca Iocchi, Daniele Nardi, Giuseppe Castellucci, Danilo Croce, and Roberto Basili. Robocupathome spoken corpus: Using robotic competitions for gathering datasets. In *Robot Soccer World Cup*, pages 19–30. Springer, 2014.

[6] Emanuele Bastianelli, Danilo Croce, Andrea Vanzo, Roberto Basili, and Daniele Nardi. A discriminative approach to grounded spoken language understanding in interactive robotics. In *IJCAI*, pages 2747–2753, 2016.

[7] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3 (Feb):1137–1155, 2003.

[8] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

[9] Daniel M Bikel, Richard Schwartz, and Ralph M Weischedel. An algorithm that learns what's in a name. *Machine learning*, 34(1-3):211–231, 1999.

[10] Yonatan Bisk, Deniz Yuret, and Daniel Marcu. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 751–761, 2016.

[11] Yonatan Bisk, Kevin J Shih, Yejin Choi, and Daniel Marcu. Learning interpretable spatial operations in a rich 3d blocks world. *arXiv preprint arXiv:1712.03463*, 2017.

[12] Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A Knepper, and Yoav Artzi. Following high-level navigation instructions on a simulated quadcopter with imitation learning. *arXiv preprint arXiv:1806.00047*, 2018.

[13] Tom Bocklisch, Joey Faulker, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management. *arXiv preprint arXiv:1712.05181*, 2017.

[14] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[15] Brendon Boldt, Ivan Gavran, Eva Darulova, and Rupak Majumdar. Precise but natural specification for robot tasks. *arXiv preprint arXiv:1803.02238*, 2018.

[16] Alexander Broad, Jacob Arkin, Nathan Ratliff, Thomas Howard, and Brenna Argall. Real-time natural language corrections for assistive robotic manipulators. *The International Journal of Robotics Research*, 36(5-7):684–698, 2017.

[17] Xavier Carreras, Lluis Marquez, and Lluís Padró. Named entity extraction using adaboost. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–4. Association for Computational Linguistics, 2002.

[18] Donald Chesworth, Nathan Harmon, Leslie Tanner, Stephanie Guerlain, and Michael Balazs. Named-entity recognition and data visualization techniques to communicate mission command to autonomous systems. In *Systems and Information Engineering Design Symposium (SIEDS), 2016 IEEE*, pages 233–238. IEEE, 2016.

[19] Istvan Chung, Oron Propp, Matthew R Walter, and Thomas M Howard. On the performance of hierarchical distributed correspondence graphs for efficient symbol grounding of robot instructions. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5247–5252. IEEE, 2015.

[20] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[21] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[22] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[23] Carlos M Costa, Germano Veiga, Armando Sousa, and Sérgio Nunes. Evaluation of stanford ner for extraction of assembly information from instruction manuals. In *Autonomous Robot Systems and Competitions (ICARSC), 2017 IEEE International Conference on*, pages 302–309. IEEE, 2017.

[24] Danilo Croce. Huric (human robot interaction corpus). University Blog. URL: `http://sag.art.uniroma2.it/demo-software/huric/`. Last visited on 2018/12/18.

[25] Anibal T de Almeida and Joao Fong. Domestic service robots [tc spotlight]. *IEEE Robotics & Automation Magazine*, 18(3):18–20, 2011.

[26] DeepLearning.TV. Recurrent neural networks - ep. 9 (deep learning simplified). `https://www.youtube.com/watch?v=_aCuOwF1ZjU`, 2015. Last visited on 2018/11/06.

[27] Kais Dukes. Semeval-2014 task 6: Supervised semantic parsing of robotic spatial commands. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 45–53, 2014.

[28] Manfred Eppe, Sean Trott, and Jerome Feldman. Exploiting deep semantics and compositionality of natural language for human-robot-interaction. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 731–738. IEEE, 2016.

[29] Manfred Eppe, Sean Trott, Vivek Raghuram, Jerome A Feldman, and Adam Janin. Application-independent and integration-friendly natural language understanding. In *GCAI*, pages 340–352, 2016.

[30] Daniele Evangelista, Wilson Umberto Villa, Marco Imperoli, Andrea Vanzo, Luca Iocchi, Daniele Nardi, and Alberto Pretto. Grounding natural language instructions in industrial robotics.

[31] Jerome Feldman, Ellen Dodge, and John Bryant. Embodied construction grammar. In *The Oxford handbook of linguistic analysis*, pages 38–11. 2009.

[32] Charles J Fillmore. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences*, (1):20–32, 1976.

[33] Charles J Fillmore. Frames and the semantics of understanding. *Quaderni di semantica*, 6(2):222–254, 1985.

[34] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[35] Ralph Grishman and Beth Sundheim. Design of the muc-6 evaluation. In *Proceedings of the 6th conference on Message understanding*, pages 1–11. Association for Computational Linguistics, 1995.

[36] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

[37] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[38] Thomas M Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6652–6659. IEEE, 2014.

[39] Albert S Huang, Stefanie Tellex, Abraham Bachrach, Thomas Kollar, Deb Roy, and Nicholas Roy. Natural language command of an autonomous micro-air vehicle. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2663–2669. IEEE, 2010.

[40] Huda Khayrallah, Sean Trott, and Jerome Feldman. Natural language for human robot interaction. In *International Conference on Human-Robot Interaction (HRI)*, 2015.

[41] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[42] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction*, pages 259–266. IEEE Press, 2010.

[43] Thomas Kollar, Stefanie Tellex, Matthew Walter, Albert Huang, Abraham Bachrach, Sachi Hemachandra, Emma Brunskill, Ashis Banerjee, Deb Roy, Seth Teller, et al. Generalized grounding graphs: A probabilistic framework for understanding grounded commands. *arXiv preprint arXiv:1712.01097*, 2017.

[44] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[45] Rui Liu and Xiaoli Zhang. A review of methodologies for natural-language-facilitated human-robot cooperation. *arXiv preprint arXiv:1701.08756*, 2017.

[46] Bill MacCartney. Understanding natural language understanding. University Lecture. URL: `https://nlp.stanford.edu/~wcmac/papers/20140716-UNLU.pdf`. Last visited on 2018/12/18.

[47] Computing Machinery. Computing machinery and intelligence-am turing. *Mind*, 59(236):433, 1950.

[48] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demostrations*, pages 55–60, 2014.

[49] Pedro Henrique Martins, Luís Custódio, and Rodrigo Ventura. A deep learning approach for understanding natural language commands for mobile service robots. *arXiv preprint arXiv:1807.03053*, 2018.

[50] Pedro Henrique Alves Martins. Human-robot speech interaction for service robots, 2017.

[51] Mauricio Matamoros, Caleb Rascon, Justin Hart, Dirk Holz, and Loy van Beek. Robocup@home 2018: Rules and regulations. `http://www.robocupathome.org/rules/2018_rulebook.pdf`, 2018.

[52] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013.

[53] Paul McNamee and James Mayfield. Entity extraction without language-specific resources. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–4. Association for Computational Linguistics, 2002.

[54] Hongyuan Mei, Mohit Bansal, and Matthew R Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI*, volume 1, page 2, 2016.

[55] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM, 2008.

[56] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[57] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[58] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Contextsensitive grounding of natural language to mobile manipulation instructions. In *in RSS*. Citeseer, 2014.

[59] Agnieszka Mykowiecka. Natural-language generation—an overview. *International Journal of Man-Machine Studies*, 34(4):497–511, 1991.

[60] Daniel Nyga, Subhro Roy, Rohan Paul, Daehyung Park, Mihai Pomarlan, Michael Beetz, and Nicholas Roy. Grounding robot plans from natural language instructions with incomplete world knowledge. In *Conference on Robot Learning*, pages 714–723, 2018.

[61] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning in natural language processing. *arXiv preprint arXiv:1807.10854*, 2018.

[62] Rohan Paul, Jacob Arkin, Nicholas Roy, and Thomas M Howard. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. 2016.

[63] Jeffrey Pennington. Glove: Global vectors for word representation. University website. URL: https://nlp.stanford.edu/projects/glove/. Last visited on 2018/12/18.

[64] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[65] Nisha Pillai and Cynthia Matuszek. Unsupervised selection of negative examples for grounded language learning. 2018.

[66] Vivek Raghuram. Ecg home page. https://github.com/icsi-berkeley/ecg_homepage/wiki, 2018. Last visited on 2018/11/06.

[67] Juliano Efson Sales, Siegfried Handschuh, et al. An open vocabulary semantic parser for end-user programming using natural language. In *Semantic Computing (ICSC), 2018 IEEE 12th International Conference on*, pages 77–84. IEEE, 2018.

[68] Pararth Shah, Marek Fiser, Aleksandra Faust, J Chase Kew, and Dilek Hakkani-Tur. Follownet: Robot navigation by following natural language directions with deep reinforcement learning. *arXiv preprint arXiv:1805.06150*, 2018.

[69] Rahul Sharnagat. Named entity recognition: A literature survey. *Center For Indian Language Technology*, 2014.

[70] Richard Socher. Natural language processing and deep learning: Introduction. University Lecture. URL: http://web.stanford.edu/class/cs224n/lectures/lecture1.pdf. Last visited on 2018/11/06.

[71] Komei Sugiura and Hisashi Kawai. Grounded language understanding for manipulation instructions using gan-based classification. In *Automatic Speech Recognition and Understanding Workshop (ASRU), 2017 IEEE*, pages 519–524. IEEE, 2017.

[72] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural

language commands for robotic navigation and mobile manipulation. In *AAAI*, volume 1, page 2, 2011.

[73] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[74] Sean Trott, Aurélien Appriou, Jerome Feldman, and Adam Janin. Natural language understanding and communication for multi-agent sytems. In *AAAI Fall Symposium*, pages 137–141, 2015.

[75] Andrea Vanzo, Luca Iocchi, Daniele Nardi, Raphael Memmesheimer, Dietrich Paulus, Iryna Ivanovska, and Gerhard Kraetzschmar. Benchmarking speech understanding in service robotics. 2018.

[76] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC, 1971.

[77] Peratham Wiriyathammabhum, Douglas Summers-Stay, Cornelia Fermüller, and Yiannis Aloimonos. Computer vision and natural language processing: recent approaches in multimedia and robotics. *ACM Computing Surveys (CSUR)*, 49 (4):71, 2017.

[78] Daniel H Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.

[79] Junpei Zhong, Tetsuya Ogata, Angelo Cangelosi, and Chenguang Yang. Understanding natural language sentences with word embedding and multi-modal interaction. *Development and Learning and Epigenetic Robotics (ICDL-Epirob)*, 2017.