Deep reinforced learning for the board game Dominion

An assignment for the course AI tools

written by

Peter Khiem Duc Tinh Nguyen
Pengu20@student.sdu.dk

Course lector: Xiaofeng Xiong
ECTS: 5

The code for this project is available at
when the code is public put the github link in the curly brackets
`https://gitlab.sdu.dk/sdurobotics/medical/student-projects/2023/Peter_Duc_Bachelor_NLP`

the Faculty of Engineering (TEK)
University of Southern Denmark

Date of Hand In 31. of May

Abstract

The aim of this thesis is to create a system capable of controlling the motion of a 6-DOF manipulator, equipped with a parallel gripper, using natural language. The manipulator is a UR5e robot arm, and the parallel gripper is a Hand-e robotiq gripper. The intent is to enable an arbitrary user to solve tasks using the robot, without the user needing prior knowledge about robot control.

The system is created using three pipelines. The first pipeline processes natural language and outputs grammatical information. The second pipeline parses the information into executable actions for the robot. The third pipeline executes the actions on the robot. Furthermore, the system can also calculate and store positions and frames in space. These points and frames are used to set up tasks in the robot manipulator's work environment.

The system is evaluated based on a test using volunteers that solves robot manipulation tasks using natural language. The results show that the system has a 70% success rate in interpretation, in which 50% of all errors stems from the users formulating instructions for moving the robot relatively. The reason is that they typically use context-based language that the system cannot interpret. An example is *'move up to the height of the starting position of the task'*, where an interpretable alternative is *'move 20 centimeters along the z-axis'*.

The final conclusion is that the system performs with 70% success rate, but does not support the use of context-based language along with high level instructions that contain several actions, like *'remove all blocks from the table'*. Based on user experience, these abilities would likely increase the success rate of the system.

Contents

Chapter 1

Introduction

The wave of technological advancements named the fourth industrial revolution, has introduced a significant amount of robots into the industrial environment [1]. A field expanded upon due to this wave, is human-robot collaboration (HRC). But even with all the collaborative robots made to this date, the level of collaboration used in the industry is still at the coexistence and synchronization levels [2]. It is therefore impervious, that more technological advancements are made with the intent of benefiting HRC. Advancements in HRC could pave the way for human-robot interaction both in the industry and outside. The advancements made in the industrial setting could possibly enable the implementation and expansion of robots in other industries, with a new found success. This could be in the medical field, where robots could be used as patient assistants for food delivery or general-purpose task solvers. They could also be used by surgeons as medical assistants during operations. It could also be within the service industry, where robots could be implemented as robotic waiters that could accomplish tasks given by the customers. This thesis will focus on reducing complexity in robot control, with the intent of benefiting HRC. It is therefore hypothesized that a robot controller capable of understanding natural language, is a good proposition for a simplified robot controller. by expanding upon HRC, then this thesis will therefore benefit the technological advancements of the fourth industrial revolution.

1.1 Problem formulation

The objective of this bachelors thesis is to design and develop a system that uses natural language text to control the motion of a 6-DOF manipulator equipped with a parallel gripper. The system must translate natural language input into executable robot commands. Formulated as a problem formulation, the problem is:

Problem formulation

Can a system be developed to use natural language text commands to control the motion of a 6-DOF robot manipulator equipped with a parallel gripper?

The problem formulation is then divided into sub-tasks, as shown below.

- Implementing a language model for Natural Language Processing (NLP).
- Designing a parser for translating NLP information into robot executable code.
- Designing a constricted set of robot movements that enables the robot to solve tasks.

The project is evaluated based on:

- In terms of success rate, How well the system interprets natural language instructions.

1.2 Problem constraint

The task given in this thesis is solved based on a list of constraints used to set expectations for the project and limit the scope. The first constraint is what kind of natural language input is expected for robot control. This constraint is set to limit the user from using natural language input that would require too much processing complexity to be

solved. Like the input *'cook an omelete'*. Another constraint is based on the hardware limit for this project. The constraints given due to the reasons mentioned are shown as follows:

- The Natural Language Processing is limited to support non-contextual natural language formulations.
- Only a finite list of actions is supported by the robot. and not all natural language input can be processed, disregarding contextuality.
- The hardware is limited to a UR5e robot arm[1] and a Robotiq Hand-E gripper[2].

With these constraints, the scope of the thesis has been placed alongside the overall problem. The next section will outline the contents of each chapter in the thesis.

1.3 Thesis outline

The thesis will focus on solving the task given in the problem formulation. Figure 1.1 shows a system overview.

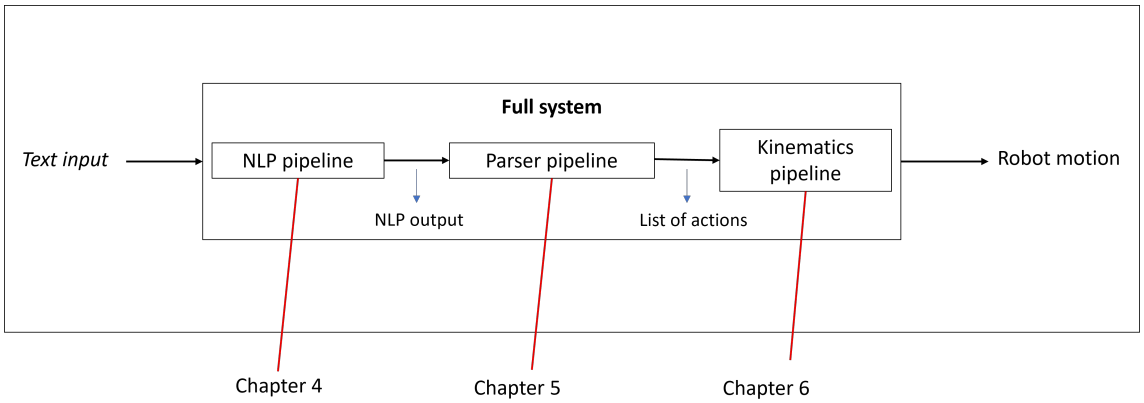


Fig. 1.1: Figure shows an illustration of the whole system, along with chapter referenes.

The content of the thesis is outlined in the list below.

- Chapter 2 is the background for robot control using NLP. This information serves as inspiration for how the problem formulation is solved.
- Chapter 3 gives an overview of the system design structure. It is used to describe the input-output structure, the motion planning structure and the user interface of the system.
- Chapter 4 as shown in figure 1.1 is the first chapter describing a sub-pipeline. This chapter analyzes natural language command text, and describes which NLP tools is used in this thesis. Afterwards the chapter describes the output from this pipeline.
- Chapter 5 explains the parser pipeline, and how it processes the output from the NLP pipeline into actions executable by the robot manipulator.
- Chapter 6 introduces the kinematics of the UR5e robot arm and the Robotiq Hand-E gripper. This chapter explains how the robot is modelled and how it is controlled.
- Chapter 7 is an evaluation of the effectiveness of the system based on the evaluation criteria given in the problem formulation.

[1] https://www.universal-robots.com/da/produkter/ur5-robot
[2] https://robotiq.com/products/hand-e-adaptive-robot-gripper

- The last three chapters are the discussion, conclusion and future work of the thesis.

The next chapter introduces the background for the thesis.

Chapter 2

Background

Given that robots are becoming more prevalent in the industry, as stated in the previous chapter. It is with increasing benefit that robots can be controlled using non-experts. Several papers use Natural Language Processing (NLP) as a tool for decreasing robot control complexity. An approach to NLP for robot control could be to use neural networks to directly map input Natural Language (NL) to robot actions[3]. The advantage of using this technique is that it removes the need to analyze the input deterministically, as this is a difficult task for NL. But this method also relies on unique training data that, in the given paper, had to be created manually. Another approach is to use neural networks to map NL into a deterministic language called Robot Control Language[4]. The neural network is trained on data pairs consisting of NL commands and their respective RCL expressions. Whereafter a processing technique relying on the syntactical analysis of Robot Control Language is used to determine the actions. Mapping to RCL is a simpler goal, than mapping directly to actions, due to the neural network performing a subtask instead of processing NL to robot motion. The advantage is the combinatory use of a neural network to process NL, and afterward, a syntactical parser to determine the actions within the RCL expression. By parting the task into two subtasks the complexity of the neural network is reduced. The disadvantage shown in the paper is that training data again was gathered manually. The last approach uses both preprocessing and postprocessing, which is done before and after the neural network categorization process respectively[5]. The preprocessing tool identifies possible commands before the sentence is split up into subsentences and fed through a neural network one subsentence at a time. The postprocessing stitches the actions back together before movement is executed on the robot. This paper intends to ease the neural network process load, by preprocessing the NL command before input. This is done, as it is known, that neural networks trained on self-created data, is a difficult task, due to lack of training data.

The general theme is to mainly use a neural network categorization tool and compensate for the workload so that the neural network task is simplified. The simplification is due to the lack of complexity of the neural network models, that in turn is caused by the limited availability of task-specific datasets. Neural networks will also be used in this thesis. But for this thesis, neural networks are used as a linguistic language model that outputs grammatical analysis information regardless of robot context. Using grammatical analysis language models enables the use of models trained on online available datasets, that are not task-specific. Furthermore, it also allows the use of pre-trained language models made in previous studies, which significantly lowers neural network implementation time. Also, by removing the time constraint of creating neural networks, several neural networks can be implemented to further gain more grammatical information from NL inputs.

To analyze the grammatical information, a parser is created, that uses the information to extract the robot movement actions. The actions are then executed by a robot.

Since the thesis uses neural networks, a short explanation of the most used neural network models is given. The following sections go into detail about how transformers work and shortly on how LSTMs work.

2.1 Transformers

Transformers are a neural network tool for Natural Language Processing (NLP). This section focuses on the details of how transformers work.

The reason why transformers are mentioned is because of their widespread use as an NLP tool. Their popularity can be seen based on the GLUE benchmark[6], where the leaderboard is dominated by bert models (a transformer neural network). The transformers have multiple features, including enhanced training efficiency and constant sentence dependency path length.
The idea of sentence dependency path length is illustrated on figure 2.1, where an example of a path length between 'move' and 'point 0' is shown. Normally, the word 'point 0' would be two words having their index. But it is treated as one word in this example. The length between the two words is 9 words. Different methods have different path length complexities for finding dependencies this is illustrated in table 2.1. Given this table, it is observed that transformers have the smallest path length of the three types, as it is constant.

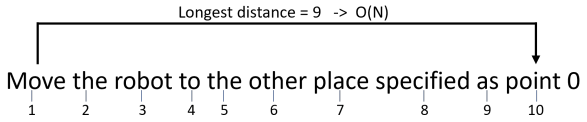


Fig. 2.1: Figure shows the maximum path length between the word 'move' and the word 'point 0'.

Neural network types	Recurrent	Convolutional	attention
Maximum path length complexity	O(n)	O(log(n))	O(1)
length given n = 9	9	3	1

Table 2.1: Table showing the maximum path length of the sentence in the previous figure.

Both of these features (training efficiency, and a constant dependency path length) are effects of the attention-based model design. Transformers use a neural network layer called multi-headed attention to compute every word in parallel. This both enhances parallel training potential and trivializes sentences with long dependencies, as attention heads can be used to compute the dependency path length of any sentence in constant time. The following sections go into detail about how transformers work.

2.1.1 Transformer design

The transformer model is as shown in figure 2.2. The rest of the subsections goes through the process of how the transformer works based on this figure.

2.1.2 Word embeddings and positional embeddings

The transformer input is passed into a word embedding. Word embeddings transform words into high-dimensional vector representations. It is trained separately, usually by predicting the next word in a training sentence, to capture the semantic word-to-word relation. The second transformation in the pipeline is positional embedding. Positional embedding is a transformation that injects values into the vector representations depending on the indexation of the vector. This is an essential part of the transformer pipeline, as it gives positional information to the vector transformation, that transformers would not gain otherwise. The reason is a side effect of the Transformers' constant complexity for dependency path length.

Afterward, the input is passed through an arbitrary amount of encoder layers, which consist of multi-headed attention mechanisms and feed-forward neural networks. Furthermore, the encoder layers are designed with residual connections, followed by normalizations. The encoder layer is larger, as it uses masked multi-headed attention intended to make the transformer model autoregressive. Meaning that during training, only previous inputs relatively, are used to process the current output. The output from the encoder layer/layers is used as input

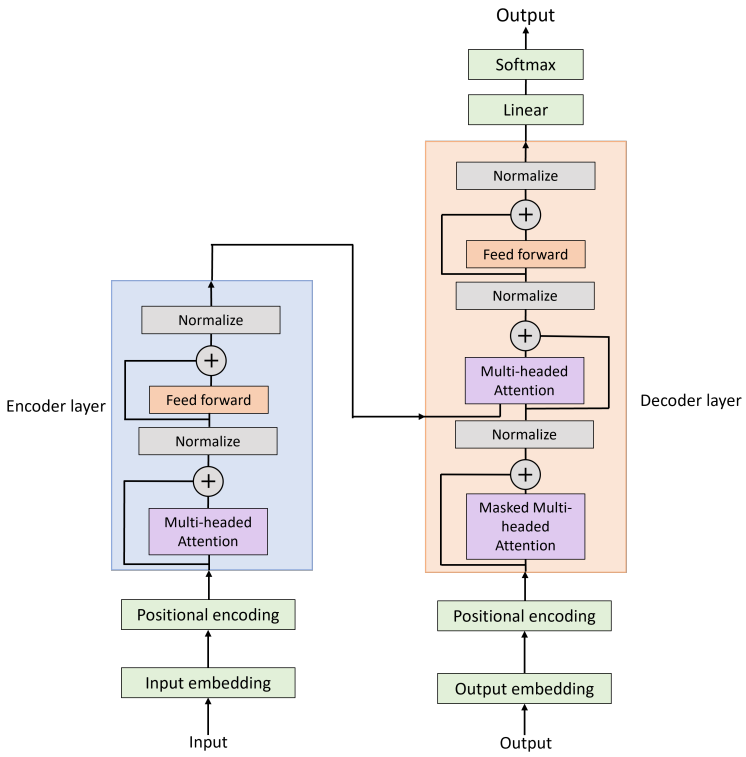


Fig. 2.2: Figure presenting the transformer model with inspiration from [7]

for the multi-headed attention within the decoder layer. Specifically, the encoder information is passed into the multi-headed attention of the decoder layer as keys and values. Keys, values, and queries are terminologies that are explained in the next section. As the decoder layers, then the decoder also has feedforward neural networks, along with residual connections followed by normalizations. The last part of the transformer is the linear projection before the softmax function. These two last layers transform the decoder output into next-token probabilities. The next sections will explain the multi-headed attention mechanism.

2.1.3 Scaled Dot-product attention

Attention is about giving numerical representations of importance. Given the calculation of an arbitrary word, then attention would output the importance of all other words in comparison to the given word. Scaled Dot-Product Attention is the formula used to achieve this goal. This calculation is done by using an input of three matrixes. The output is described as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{Q\ K^T}{\sqrt{d_K}})V$$

The three matrixes are the query matrix (Q), the key matrix (K), and the value matrix (V). The key and query matrix is used in combination to create a matrix representing the attention information of the sentence. This information is then multiplied onto the value matrix that holds the vector representations, such that the attention information is used to weight the vector representations. The scaling factor d_k is the dimension size of the keys matrix. The dot product between Q and K^T becomes inefficiently large if the dimension of the keys (d_k) becomes too large. It is therefore solved by dividing the dot product by the square root of the key dimension. This formula serves as the key to the multi-headed attention explained in the next subsection.

2.1.4 Multi-headed attention

Multi-headed attention is at the core of transformer models. It is an arbitrary number of scaled dot-product attention heads in parallel, as shown in figure 2.3. The output from the scaled Dot-product attention is concatenated with each other. But that would also mean, that the output matrix has n times more columns than the input. Therefore, after the concatenation of all the results, a linear projection is used to reduce the matrix size back to the original word embedding size.

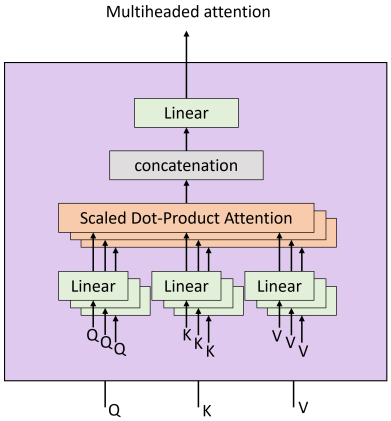


Fig. 2.3: Figure presenting a multiheaded attention mechanism made based on [7]

The linear projections are an essential part of the system, as they are the trainable part of the multi-headed attention mechanism. In the original paper[7], it is stated that the separate attention heads, using different linear projections, seemingly exhibit behavior related to the syntactic and semantic structure of language. This behavior is achieved by the trained linear projections.

2.2 Recurrent neural networks

transformer model are typically superior over Recurrent neural networks (RNN) models and convolutional models, given their numerous appearances on the GLUE NLP benchmark leaderboard[6]. But RNN's still have a feature where they are superior to transformer models. The idea of RNNs is that each input word is processed through the model and becomes the new input for the next word, as shown in figure 2.4. In other words, the RNN has a linear path length complexity with respect to the input length (as shown previously in table 2.1). Furthermore, it also has a linear calculation complexity. Calculation complexities with respect to the input length of different neural network types are given in table 2.2. Where n is the input word count, d is the dimension of the vector representation, r is the range of words used in a restricted transformer network, and k is the kernel size of a convolutional neural network.

Even though the transformer has a constant path length dependency, as stated in Section 2.1 then every word must still be processed by the multi-headed attention mechanism. Every word is therefore processed with respect to every other word. Even if the longest path length is constant, then the calculation complexity becomes quadratic with respect to the input length. Therefore, if n is greater than d in table 2.2, then a recurrent neural network has fewer calculations than a transformer. Recurrent neural networks are therefore better suited for tasks requiring long input sequences, like thesis summarization.

The RNN encoder model explained in this thesis, is the LSTM model. The next section will go through the encoder models used in this thesis.

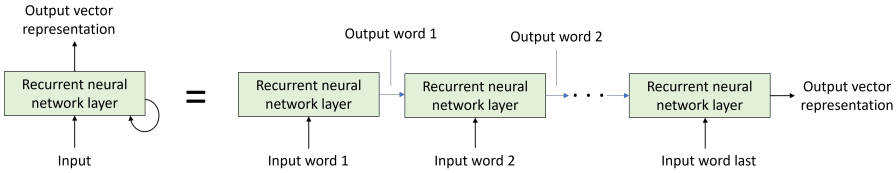


Fig. 2.4: Figure showing how a recurrent neural network processes data.

Neural network types	Calculations per layer	Maximum path length
Transformer	$O(n^2 d)$	O(1)
Recurrent	$O(n d^2)$	O(n)
Convolutional	$P(k n d^2)$	O(log(n))

Table 2.2: Table showing the processing complexity of different neural networks.

2.3 Encoder models

There are three main model types for NLP, as shown below.

- Encoder-decoder models (sequence-to-sequence)
- Encoder models
- Decoder models

Encoder-decoder models are, in the simplest terms, the full transformer design described in the previous section. The input is processed in the left part of the transformer, and the output is generated in the right part of the transformer. The encoder-decoder model, also called the sequence-to-sequence model, is mostly used for text summarization. The left part of the transformer is the encoder model. It focuses on processing the input, and is usually used for text information extraction, such as assigning grammatical categories or analyzing sentence structure. The right side is called the decoder model. It is used for generating output sentences. This is used for general text-generation tasks. As this thesis needs a tool that can analyze input sentences and gain information from the input, the encoder model is the most prominent. With encoder models, it can be expected that the model outputs information about the input, rather than more words.
The most common NN encoder model used in this thesis is the BERT model. This model is the encoder model used with the transformers

2.3.1 BERT

A recent development in the field of neural networks in relation to NLP is the release of the BERT model, which stands for (Bi-directional Encoder Representations Transformer) [8]. BERT, unlike other transformers, is trained using two distinct methods. word-by-word relation and sentence-by-sentence relation. Word-by-word relations are trained by letting the model predict masked words in a sentence, as shown in figure 2.5. Sentence-by-sentence relations are trained by giving the model sentence pairs marked as sentence A and sentence B. The model afterwards determines whether sentence A is the continuation of sentence B in a text. An example is shown in figure 2.6.
Bi-directionality stems from the BERT model's word-by-word training design, where the full sentences are shown except for the masked word. This is opposed to previous transformer models that were trained left-to-right or right-to-left. The training methods make the model achieve state-of-the-art results on a wide variety of NLP tasks without task-specific training. In other words, the training allows the BERT model to gain a generalized understanding of natural language. This enables the BERT to be released as a pre-trained language model. All further training will then be done with comprehension of natural language as the starting point. This method is called finetuning and allows the model to train faster on any NLP task.
The following section describes another important encoder model that is used in this thesis.

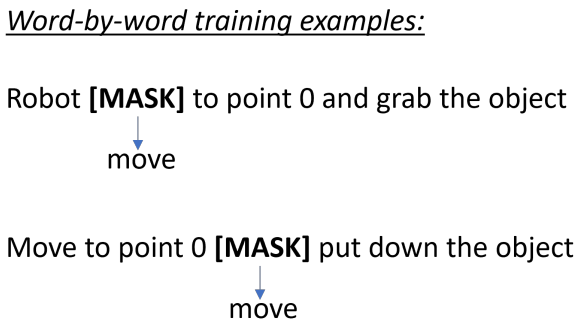


Fig. 2.5: Figure shows the training method that learns the model's word-by-word relations.

	A	B
Correct sentence continuation	Move the robot towards point 0.	Then move the robot to point 1.
Erroneous sentence continuation	Grab the object at point 0.	The anatomy shows that the cats have thin bones.

Fig. 2.6: shows sentence-by-sentence relation training examples, with a correct sentence continuation and a wrong sentence continuation.

2.3.2 LSTM

The Long Short Term Memory (LSTM) cell, is a type of RNN node that is widely used for NLP. This model was the prevalent design for NLP tasks before transformers and is still used today. This section will shortly explain how the LSTM models work.

RNNs have a linear path dependency. It is therefore also expected, that the representation calculation is done using information from all previous words. But some words are more important to remember than others. This problem makes general RNNs bad at long-term dependencies, as they try to carry information from all previous words into the current word calculation. figure 2.7 shows a representation of the LSTM cell.

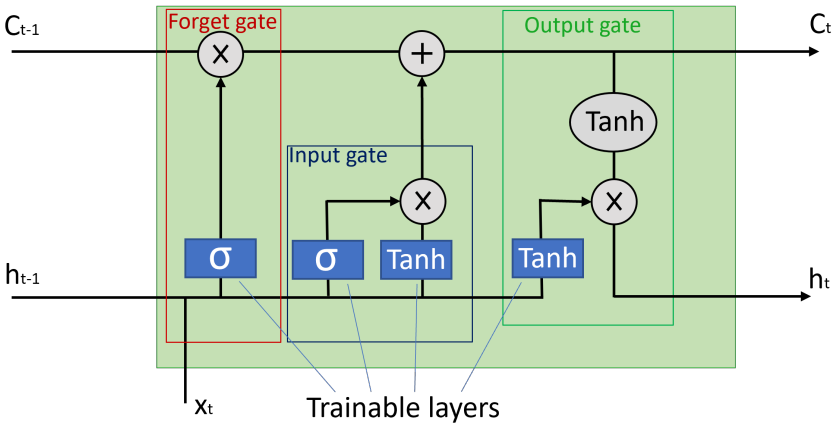


Fig. 2.7: Shows a LSTM cell that consists of learnable layers marked with a blue rectangle, and pointwise operations marked as grey circles.

The purpose of the LSTM cell is to train the cell to hold long-term dependencies with higher priorities than meaningless information. The sigmoid and the tanh layers marked with a blue rectangle are the trainable layers of the LSTM cell. The sigmoid layer marked in the "Forget gate" is a layer with values ranging from 0 to 1, determining

how much input information should be forgotten. Afterward, the sigmoid and the tanh layers in the area named "Input gate" are used to inject information into C_t. The last tanh layer within the area named "Output gate" is used to calculate the next hidden state for the next LSTM layer. The use of the LSTM cell is the fundamental block for LSTM models. There are multiple alternative model styles for LSTM models, But they will not be discussed in this thesis.

Chapter 3

Overview

The purpose of this chapter is to give an overview of the collected system so that the reader has a better idea of the purpose of each chapter in this thesis. Moreso, it is also to define exactly what the system is capable of doing and how it perceives and manipulates its environment. In the coming sections, the terminology of "the system" is explained, along with its capabilities for solving tasks. This information is carried into the subsequent sections, where the whole system is reviewed.

3.1 Introduction to the system

The full pipeline is named 'system' in this thesis. Figure 3.2 presents a diagram describing the input and output of the system. The system is capable of receiving text as input, which is expected to be natural language. The input will have three distinct terms, as shown in figure 3.1. The input sentence is the input the user passes to the system. The second term is 'instruction'. The instructions are all natural sentences in the input sentence. Instructions are used instead of the input sentence later in the thesis, as they have the feature that the root of the instruction is the verb used to determine the instructions actions. The terminology of word roots is explained in chapter 4. The last terminology of the input is 'actions'. The actions are objects that are executable on the robot manipulator. Usually, the instructions contain only one action each. But because the action of grabbing does not require specification from the user, it is possible to combine it with the action to move, as shown in figure 3.1.

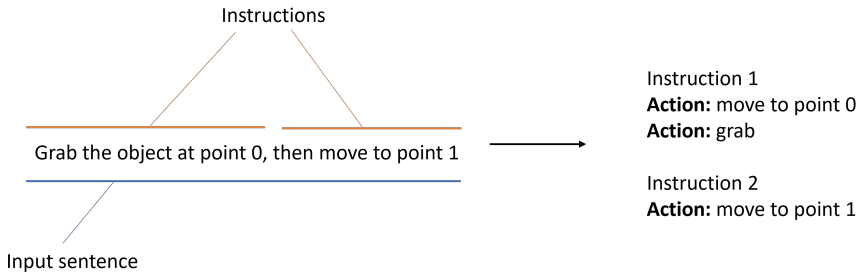


Fig. 3.1: Figure shows the relation between the terms input sentence, instructions, and actions.

The system works by using three sub-pipelines. The interpretation of natural language is done by using natural language processing (NLP) tools. By using the NLP tools, the input can be split into its instructions. Moreso, NLP information regarding sentence structure and grammatical categories are also given. This part is named the NLP pipeline and is the first sub-pipeline of the system. Afterward, the parser analyzes the NLP output to figure out what actions the sentence consists of. This second sub-pipeline is called the parser pipeline. The output of the parser pipeline is a list of actions in sequence given to the third and final sub-pipeline. The third sub-pipeline is the kinematics pipeline, which handles the system's interaction with the environment. Mainly, it is used to move the UR5e robot according to the actions in the list. But it is not limited to only moving the robot. The kinematics pipeline is also responsible for placing points and frames in space and manipulating the robot gripper. Frames and points are information regarding pose and location stored in the database of the system. The next section delves deeper into the idea of points and frames.

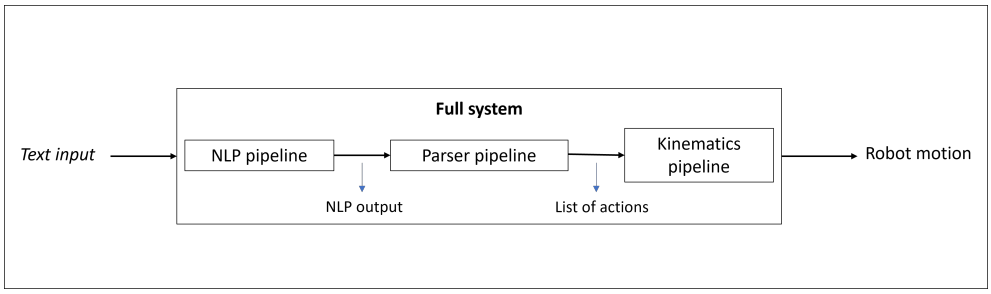
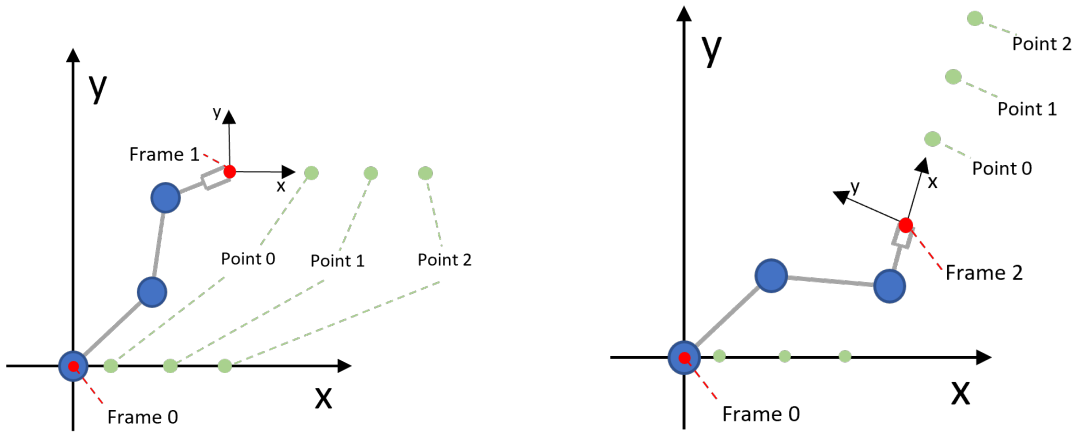


Fig. 3.2: Illustration of the information process given the full pipeline.



(a) Visualization of points in frame 1.

(b) Visualization of points in frame 2.

Fig. 3.3: 2D visualization of the point frame system created.

3.2 Frames and points

The positions stored in the database serve the purpose of supporting the UR5e robot so that it becomes easier for it to solve tasks. It is important to explain in detail what positions and frames are, and how the system operates using them.

Frames are coordinate systems set in space in reference to the base coordinate system. The base coordinate system is placed at the base of the robot. The purpose of the ability to set poses and frames online is to bring a coordinate system closer to the task operation field. This ability facilitates the robot's setup of the correct points in space. Figure 3.3 shows a 2D visualization of how the points and frames are intertwined. Points set in the system are relative to coordinate systems. The figure shows how points 0, 1, and 2 are set in extension of the x-axis and are therefore apparent along the given axis in all of frames 0, 1, and 2. Subfigure 3.3.b shows that the feature allows the user to set points in a complex position, compared to trying to set them up in reference to the base frame (frame 0). Using frame 2, setting up the three points would be a task solved in 1 dimension only (the x-axis). The system only sees one frame at a time. Meaning that all points in space are relative to the frame that the system is currently working on. So to change the working frame, the user must specify this action for the system. A language command example could be *"Set frame 1 as the current frame"*. And to create new frames, the language command could be *"Set position as a frame"*. This design makes the language control simpler, as the language processor does not need to process commands such as *"Move to point 1 in frame 1"*.

3.3 System interfacer

The system interfacer is the interactional part, serving the purpose of communicating with the user. Given that the thesis revolves around natural language communication control, it naturally follows that the system interface is a terminal with no input constraint. This is shown in figure 3.4.

Fig. 3.4: Input from the system after instructing it to move to point 2.

In the given figure, it is shown that the interaction happens twice. The first time is a safety prompt, as it is unusual to move to any point while the robot is working in the base frame. It usually causes the robot to collide with itself as the base frame is placed at the base of the robot. This warning is ignored in this instruction to further show the second interaction. The second prompt is for user confirmation that the action stack is as desired. Further in the thesis, it is also shown that a simulation of the robot's movement is made to make sure that the movement is not erroneous.

3.4 Natural language input constraints

The natural language instructions used to control the system are only a subset of full natural language, as there are constraints made to simplify the task for this thesis. These constraints are mentioned in chapter 1 as the constraints of the problem. This section delves deeper into how the system should act if the input is incomprehensible in any way. As an example, sentences such as *"Do a funny dance"* cannot be successfully executed as they require contextual knowledge and advanced movements. The subset of instructions that the system is capable of are instructions that focus on simple sequential tasks, such as *"Move to point 1, then move to point 2, then grip the object."*. Therefore, though the interface can input arbitrary text, only a specific language style is converted into actions. If the user inputs instructions in any way that is incomprehensible to the system, it would either not output any actions, give a better explanation of why the instruction was not understood, or give a guess as to what the user might have meant. These options are explored in chapter 5.

3.5 Summary

The system is a programmed pipeline, with a UR5e robot arm equipped with a gripper, capable of processing natural language such that it becomes capable of manipulating its environment using the robot. What has been covered as part of the overview is a short explanation of the purpose of the pipeline and each of its sub-pipelines, how the physical space is mapped with points and frames, and how the system is interacted with. The overview gives the reader information that will be used later to further explain how each sub-pipeline has been created. The next chapters of the thesis cover the theory and the creation of the sub-pipelines.

Chapter 4

Natural Language Processing pipeline

The Natural Language Processing (NLP) pipeline is the first subpipeline of the system. It is expected that natural language is given as input. Natural Language Processing (NLP) is therefore used to process and analyze the input. This is done with the purpose of later transforming the input into executable actions. To give the reader a clearer understanding of the NLP tools used, an example of a tool is the POS tagger. Figure 4.1 shows how the POS-tagger is used to categorize each word into its grammatical category.

Words:	Set	the	TCP	as	a	new	point
POS tags:	**Verb**	**Determinant**	**Noun**	**preposition**	**determinant**	**Adjective**	**noun**

Table 4.1: Table illustrating the output of a POS tagger.

4.1 Analysis of natural language commands

The input for the system pipeline is regarded as an instruction-based sentence. It is assumed that the basic parts of an instruction-based sentence are a verb and a noun. The verb is used to determine the action, and the noun is used to determine the subject of the action. The sentence "move the tool point" is an example of an instruction-based sentence. The verb is "go", and the noun is "robot". More nouns or numbers combined with units can be added to expand the instruction and further describe the action. The sentence "move the tool point to object A" is an example of an instruction sentence with one verb and two nouns. The verb is "go", the first noun is "tool point" and the second noun is "object A". This is only a simplified example of how instructions could be structured, since the complexity of language can be unlimitably complex by adding to instructions any number of verbs, nouns, prepositions, adjectives, or others. Natural language input can also consist of any number of these complex instructions. For summarizations then there is two key informations. The first being, that instruction is likely to have a verb used to determine the action within the instruction. It is also likely to be accompanied by nouns specifying it. The second piece of information is that sentence structure analysis is vital for the NLP pipeline, as the instructions can be arbitrarily complex. The approach to analyzing the instruction-based natural language input is to use a sentence structure analyzer to find the important verbs, nouns, and others in the sentence. By stating all grammatical categories and having a structure analysis of the sentence, it is possible to determine the actions contained within the instructions. Using this information it is possible to figure out, what natural language processing tools is needed to solve the task. The next section introduces stanza, and the tools used in the NLP pipeline.

4.2 Stanza Pipeline

The stanza pipeline is a Standford NLP package containing several different language models [9]. The models make the stanza pipeline capable of several different NLP tasks. This makes the Stanza pipeline useful for using several NLP analysis tasks in one tool.
The following section is a short description of the tools selected as the NLP tools for the NLP pipeline used. A complete list of the given tool names is given below.

- tokenize

- mwt
- pos
- depparse

The tools used are based on the neural network model named "en", which stands for English. This model was downloaded from the hugging face community website[1]. For this thesis, it was important to find a model which made use of transformers, but the English, danish, and multilingual stanza language models did not use transformers. They were all based on LSTM models. Therefore the most popular model was chosen instead.

4.2.1 Prerequisite NLP tasks (tokenize and mwt)

Most of the tools in the stanza pipeline require other basic NLP tools. An essential tool for all other NLP tools is a tool for segmenting the input. The stanza pipeline uses two segmentation tools. Firstly, the tokenization module is used to segment the input into words. Secondly, the sentence segmentation module is used to segment the input into sentences. Another prerequisite task is "mwt". This is the multi-word token expansion module used to expand multi-word tokens into several single tokens. This is done by using a dictionary of multi-word tokens. An example could be the word "aren't" which can be broken into the two tokens "are not".

4.2.2 Part of speech tagging (pos)

Stanzas Part of speech tagging module is a tool capable of categorizing words into their respective grammatical categories. This is used to determine the meaning of each word by assigning it a grammatical category, like a noun or verb. This tool is essential for the parser in the next pipeline to extract the correct actions from the instruction. An example of POS tag processing is shown in table 4.1.

4.2.3 Dependency Parsing (depparse)

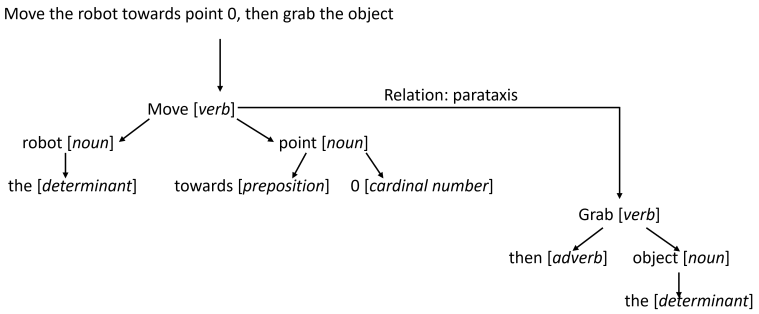


Fig. 4.1: Figure illustrating an example of the dependency parser structure, where 'Move' and 'grab' are the roots of their respective sentences.

The dependency parsing tool creates connections between the words within the sentence and their respective syntactic heads. An understanding of a syntactic head is that traveling from word to syntactic head always leads to the root of the sentence. The root of the sentence is the most descriptive word in the sentence and does not have a syntactic head. If the input sentence is an instruction, then the root is almost always the instruction verb. This structure is shown in figure 4.1. Figure 4.1 also shows that a relation is given between the two verbs. The dependency parser gives relations between all words and their syntactic heads, but only relations between verbs are used by the parser. The relations between verbs are used to determine if the sentence has more than one instruction.

[1] https://huggingface.co/stanfordnlp/stanza-en

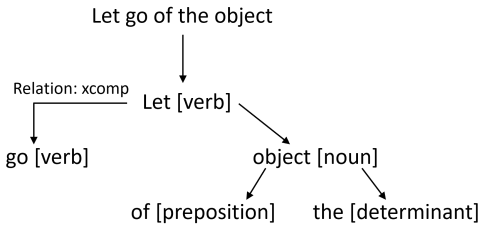


Fig. 4.2: Figure illustrating two verbs but one instruction

The relation between the two verbs shown in figure 4.1 is parataxis. Parataxis means that the two verbs describe two separate actions. This means that the two verbs are not dependent on each other and therefore should be treated as two separate instructions. Figure 4.2 shows an example of a verb relation that indicates that 'go' is not an instruction by itself. This information is important, as 'go' should not be used by itself to indicate any actions.

4.2.4 Stanza NLP output

Using all of the given tools, the NLP tool is now capable of segmenting all of the words into their respective grammatical categories, along with passing structural information about the sentence. The output from the pos tagger, along with the output from the dependency parser, is the output of the NLP pipeline. The output is visualized as shown in figures 4.1 and 4.2.

Chapter 5

Parser pipeline

The NLP pipeline output, as described in the last chapter, is the input for this chapter. The NLP pipeline output is the natural language input instruction with NLP tags on each word (grammar category and syntactic head). The parser pipeline is tasked with deconstructing the NLP pipeline output into actions that the kinematics pipeline can execute. To do this, the NLP output is first deconstructed into verb sentences. Verb sentences are explained in section 5.3. Before the processing is explained, it is important to first introduce the system database, as the parser makes use of the database to process the NLP pipeline output.

5.1 Memory handling

Two essential information types are stored in this thesis. The first information type is words. The words are used to identify actions, their noun subjects, the noun describing the movement destination, and more. Figure 5.1 shows an example of how the words in the database are used to match the verb 'steer' with the action 'move', as the verb is stored under the 'move' action verbs. Another noteworthy word is 'towards'. It is used to deduce that the 'move' action must be done towards the noun 'point 1'. This is because the word 'towards' is a preposition stored in the database as a viable proposition for the destination noun of the action 'move'. It should also be known that if a number appears after the word 'point' or 'frame', then the combination of the word and the number is treated as one noun. This method is used to determine the action in the instruction and the role of all other words in the instruction with respect to the action. The other type of information is the location of points and frames. This

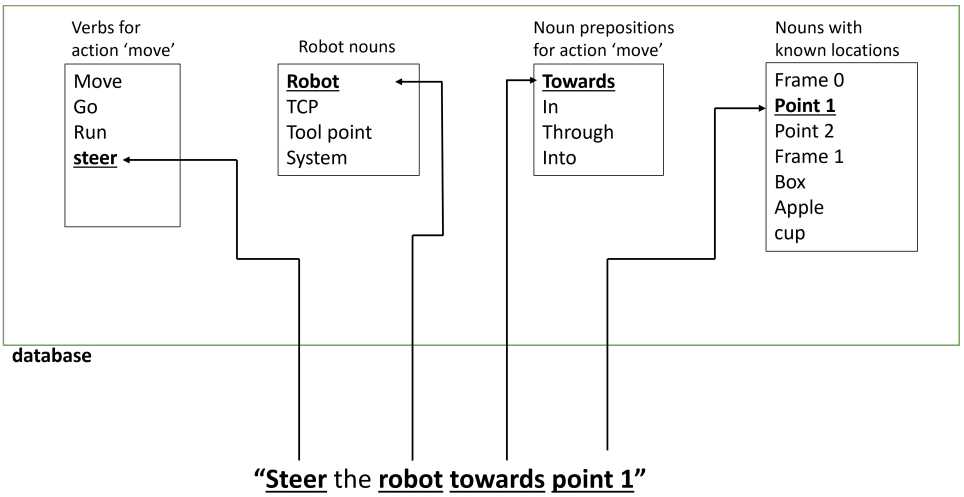


Fig. 5.1: Figure illustrating an example of how the database works.

information is used in the kinematics pipeline to move the robot. The location information of points is stored as vectors denoting position, and frames are stored as homogenous transformations. It is determined that the database is a tree structure with .txt files, as this structure is practical for fast reading and writing.

5.2 Word comparison using cosine similarity

An NLP tool is used for the comparison of input words with database words. This tool is used to catch errors when the input words do not match any words in the database. Like using 'proceed' as a synonym for 'move'. Also, if the system is not constrained by the dictionary of words in the database, it would gain a larger corpus, which would increase the natural language interpretation ability. Ideally, the system would not compare the words grammatically but rather with their intentions. The comparison method used is therefore the cosine similarity of word embeddings. Which uses the dot product of two word embedding vectors to calculate their similarity. The similarity values go from -1 (no similarity) to 1 (full similarity). Figure 5.2 shows how the similarity score is used in this thesis. If the word used in the input does not match any words in the corresponding database word list, then the database word with the highest similarity score is suggested to the user. If the user disagrees, then the next highest similarity score option is given. This repeats until the user is satisfied or a threshold is reached, at which point the system gives up. In chapter 7 it is evaluated that the lowest similarity score needed to match all words in the data is 0.40.

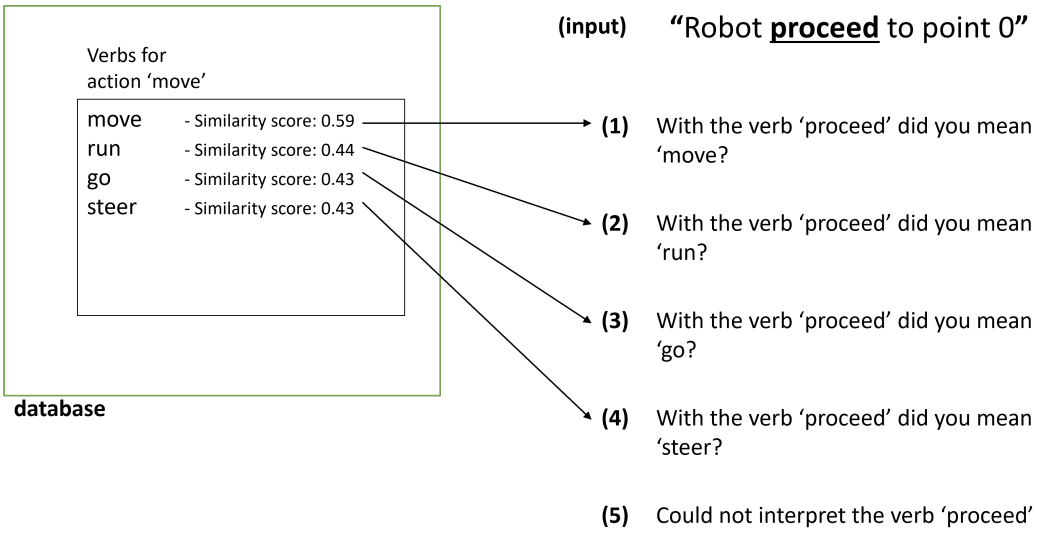


Fig. 5.2: Figure illustrating how the cosine similarity is used to communicate with the user.

5.3 Parsing NLP pipeline output to actions

The earlier sections describe methods used to process individual words. The database holds the words matched with the words in the input sentence to find actions. The cosine similarity communicates with the user if any input words do not match the words in the database. It would be impractical to match all words in the database with all words in the sentence. It is therefore necessary to analyze the NLP pipeline output to gain more information about the words in the sentence before matching the words with the database words. Figure 5.3 shows a visualization of the information gained by the NLP pipeline output. Firstly, it is known that the verb-by-verb relation 'parataxis' is used to split the sentence into its several instructions. Afterward, the root of each instruction is matched with the action verbs in the database, as the root is always a verb. The nouns connected to their respective root verbs are analyzed to find the information needed to execute the actions. The analysis methods for the nouns are action-specific. It is therefore explained in the later sections.

Using the comparisons method explained previously and using the information gained from the NLP pipeline,

the sentence in figure 5.3 can be transformed into two actions as shown in figure 5.4.

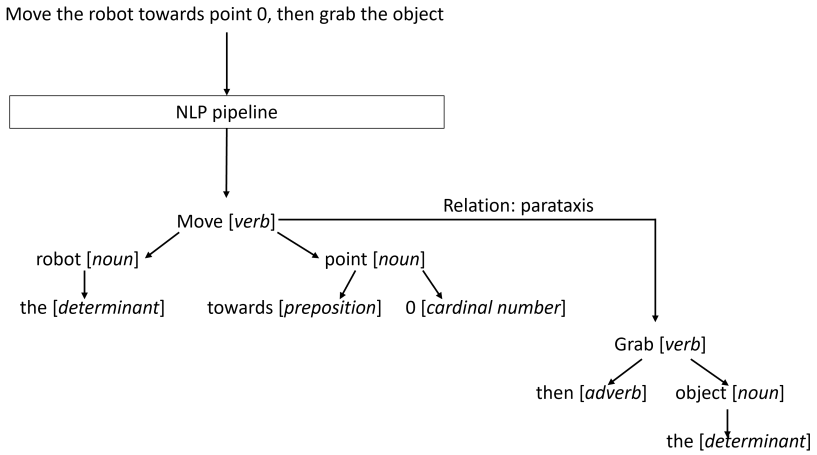


Fig. 5.3: Figure illustrating the NLP pipeline output from the previous chapter.

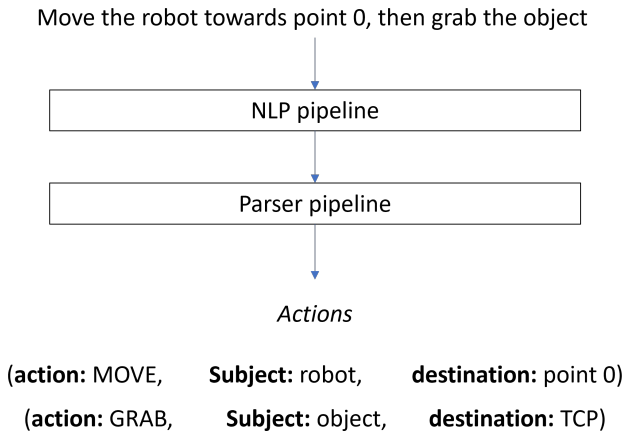


Fig. 5.4: Figure showing the parser pipeline output as actions.

5.4 Actions

The last part of the parser pipeline is the development of the actions. The actions are designed to move the robot manipulator so that it is capable of solving tasks. The key is making the actions simple so that all tasks are a concatenation of them, as described by the user. The action objects are divided into three categories, movement actions, meta actions, and system actions. The movement actions describe a robot's movement. The meta actions manipulate the sequence of actions. The system's actions manipulate points and frames in the physical environment. Meta actions are resolved in the pipeline before sending the list of actions to the kinematics pipeline. A list of the actions available is given below.

- move
- rotate
- pick up

- put down
- repeat - Meta action
- set point or frame - System action

An in-depth description of each action, along with a visualization of the action objects, is given in the following sections.

5.4.1 Action: Move

The move action is split into the following two movement types:

- Movement with reference to a point in space
- Movement relative in an axis

The first type describes the movement of the robot, based on points or frames in space. The second type of movement describes how the robot should move if it is given a direction and a length. An example of this type of movement is *'move 20 cm in the x-axis'*. It is only capable of moving along the axes of the robot TCP. The action object passed on is given as a list written as in figure 5.5.

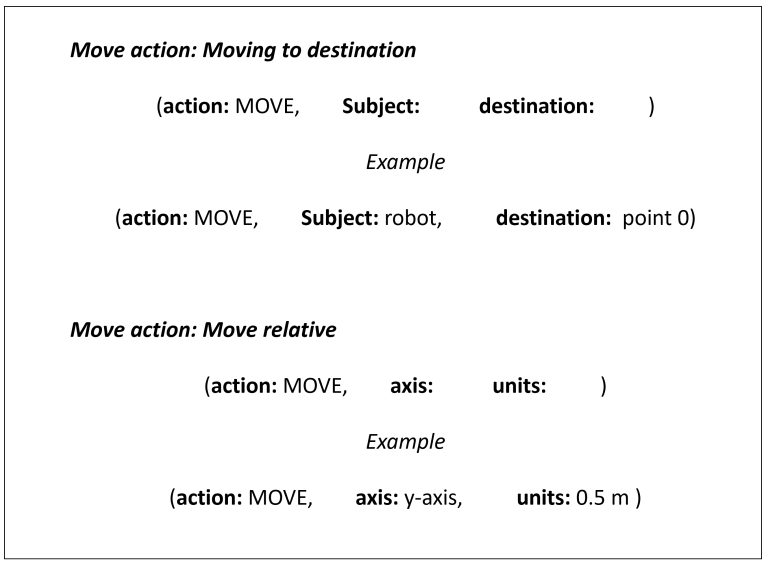


Fig. 5.5: Figure illustrating how the 'move' action would look in the system.

For the action 'move' specified by destination, then the subject of the movement is found by searching the NLP input for a noun without a preposition as in the word 'robot' in figure 5.3. Afterward, the noun is also checked in the database to make sure it is valid and that it is not a random noun in the sentence. The destination is found by searching the sentences for a noun with a preposition. In figure 5.3 it can be seen that the noun 'point' has a preposition 'towards' that furthermore also matches the prepositions in the database for noun destinations of the action 'move'. This makes 'point' a destination for the action move in the example. The cardinal number '0' is observed to not always be connected with its noun. This is the reason for the solution of just treating cardinal numbers in front of the nouns 'point' and 'frame' as part of the noun.

The action 'move' specified as a relative movement, is chosen if two conditions are met. If a noun describing one

of the three frame axes is detected, And if no destination noun is found. Furthermore, the noun describing the axis must also have a preposition that shows that the action is done towards the axis noun. Examples of prepositions could be *"along or using"*. If these conditions are met, then instead of searching for a noun destination, the system instead searches for a cardinal number along with units. An example could be *"meters, m, centimeters, cm"*. After finding the axis and the cardinal number describing distance, it is passed on as an action.
It should also be noted, that if no noun subject is found, then it is by default 'robot'. Such as in the sentence *'move to point 0'*.

5.4.2 Action: Pick up/Put down

Though picking up and putting down are two separate actions, it makes more sense to treat them as a pair. Kinematically, they have no influence, as these actions control the gripper. The system is capable of remembering the objects it is holding, so it does not try to pick up objects when it is already holding one. As of now, no real objects are coded into the system, meaning that only the default object called "object" is available. The system is capable of supporting a design where each object has its own coordinate in space. But this work is left for future work. The destination space in figure 5.6 is if the grab is done at a specific location, as with the sentence *'pick up the object at point 0'*.

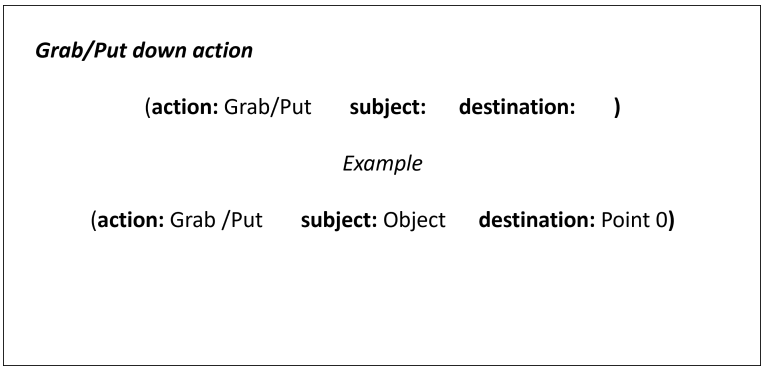


Fig. 5.6: Figure illustrating how the put/pick action would look in the system.

The system finds the subject by searching the instruction for a noun without a preposition. The words found are checked in the database to make sure they are valid. The destination is found by searching for a noun with a preposition, exactly the same as with the move action. The destination noun is, per default, TCP. and the subject is, by default, the object "object". Therefore, a valid sentence for picking up or putting down could simply be *'pick up'* or *'put down'*. where the noun subject would be 'object' and the noun destination would be 'TCP'.

5.4.3 Action: Rotate

As the UR5e robot only has revolute joints, this action is made with the intention of manipulating any part of the robot. An example of an instruction with the rotate action could be *'rotate 90 degrees around wrist 3'*. The action object created is given in figure 5.7.
The subject of the action is found by searching for a noun (with or without prepositions) that corresponds to a part of the UR5e robot. The units are found the same way as the units are found in the action move relative. The subject is by default set to 'robot'. Meaning that the sentence *'rotate 90 degrees'* would be interpreted as rotating the base of the robot 90 degrees.

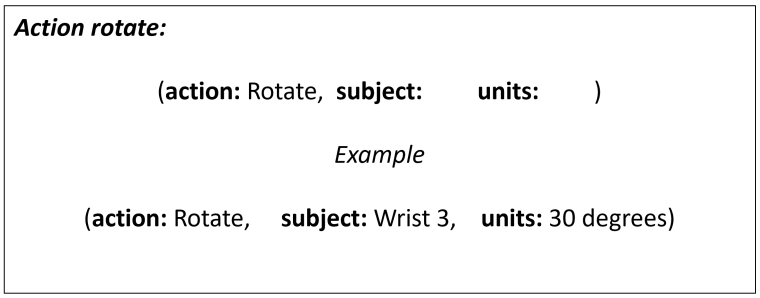


Fig. 5.7: Figure illustrating how the rotate action would look in the system.

5.4.4 Action: Repeat

Repeat is a meta action, meaning that it serves to manipulate other actions while the system translates verb sentences into actions. There are three types of repeat actions. Either it repeats the actions in the previous part of the sentence, the next part of the sentence, or the whole sentence, regardless of the statement's position. The correct type of repetition action is determined based on the preposition bound to the subject of repetition. An example could be the following sentence *'repeat the next actions 3 times'*. where the repeat statement would repeat the next actions. But substituting the preposition 'next' with 'previous', would make the repeat statement repeat the previous actions. The action refers to all actions in the sentence if it cannot detect the use of either preposition hinting towards repeating either previous or next actions. The action object is shown in figure 5.8.

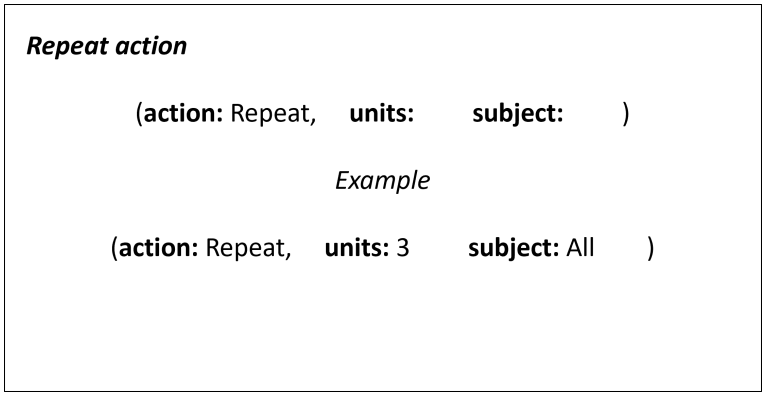


Fig. 5.8: Figure illustrating how the repeat action would be perceived in the system.

The units are found by searching for a noun describing the repeat amount. Like the noun *'times'* or *'cycles'*. Then afterward use the number that comes before the unit. The subject is found as described in how the action is split into three categories.

5.4.5 Action: set

This action is a system action, meant for setting up the tasks in the environment. Even if it is the only system action, the versatility of the verb 'set' allows it to do three different things. Specifically, the things stated in the following list.

- Store the position of the tool center point as a new point.
- Store the pose of the tool center point as a new frame.
- Switch the frame that the robot works in with another frame.

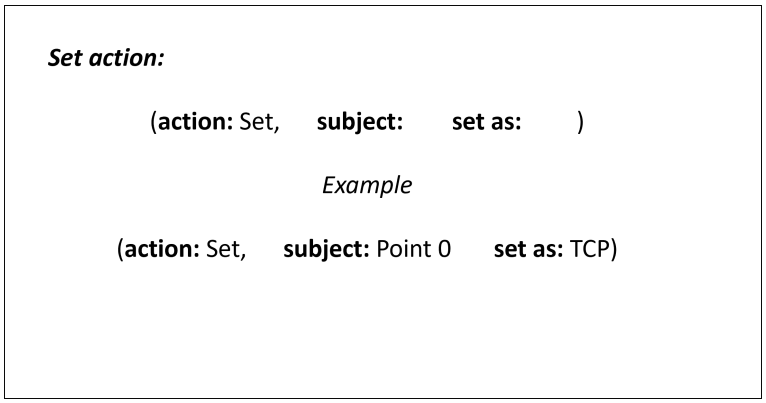


Fig. 5.9: Figure illustrating how the set action would be perceived in the system.

The set action is shown in figure 5.9.
The subject is found by searching for a noun without a preposition. 'set as' is not processed and is always its default value, which is TCP. But the idea is that it could be used to set points at TCP + 7 cm on the z-axis. But this information is too complex and is therefore left for future work. If the noun subject is a frame with an index already present in the system, then the system switches the work frame to the new frame. If no index is given (both for the nouns 'point' and 'frame'), then a new 'point' or 'frame' is created.

Chapter 6

Kinematics pipeline

This chapter focuses on the physical manipulation of space using the actions and objects given in the previous chapter. The kinematics pipeline handles the manipulation of points, frames, the UR5e robot arm, and the gripper. Figure 6.1 shows the information process for the kinematics pipeline. The list of actions is received by the Robotics System Toolbox (RST)[1], where the necessary kinematical operations are done to convert the actions into a list of joint configurations. A joint configuration is a list of joint angles describing the robot's pose. Afterward, a simulation of the movement is used to confirm with the user that the given action is correct. Only then will the RTDE Python package instruct the UR arm to move to the given joint configurations.[2]

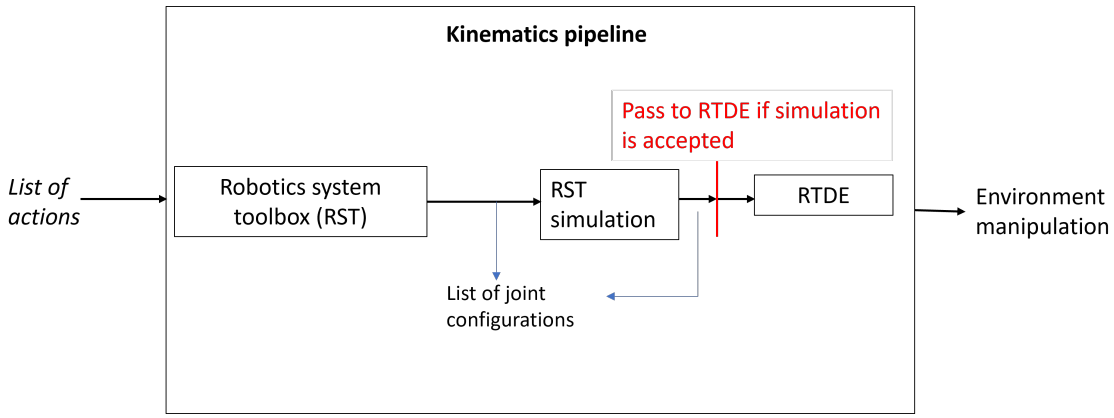


Fig. 6.1: Figure showing the pipeline for the kinematics part of the system.

For this chapter, the main focus is on the kinematics of the UR5e robot, the software tools used to do the physical operations, and how the pipeline is used to support the system. Furthermore, an explanation is given for how points and frames are created and calculated. The next section introduces the UR5e robot arm and gives a short review of the kinematics of the robot.

6.1 UR5e kinematics Introduction

The UR5e robot arm is the robot manipulator of this system. It is essential to model the kinematics of the robot for the system to be able to control the robot arm. The kinematics of the arm is based on the DH parameters, as the DH parameters transform the coordinate system from the base frame through every joint towards the TCP frame. The DH parameters are shown in table 6.1. The DH parameters were tabulated from the Universal Robots official website.[3]

Table 6.1 shows the numbers for the DH parameters of the UR5e arm. Figure 6.2 shows a visualization of the given DH parameters. The figure visualizing the DH parameters shows the extension of the UR5e arm links and their natural rotation along the x-axis of each frame. To get a better comparison between the DH parameter visualization and the physical UR5e arm, figure 6.3 is created to show the movable joints of the UR5e arm, placed in the DH

[1] https://petercorke.github.io/robotics-toolbox-python/intro.html
[2] https://sdurobotics.gitlab.io/ur_rtde/
[3] https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/

Kinematics	θ [radians]	a [m]	d [m]	α [radians]
Base (Joint 1)	θ_1	0	0.1625 (d1)	π/2 (α1)
Shoulder (Joint 2)	θ_2	-0.425 (a2)	0	0
Elbow (Joint 3)	θ_3	-0.3922 (a3)	0	0
Wrist1 (Joint 4)	θ_4	0	1.333 (d4)	π/2 (α4)
Wrist2 (Joint 5)	θ_5	0	0.0997 (d5)	-π/2 (α5)
Wrist3 (Joint 6)	θ_6	0	0.0996 (d6)	0

Table 6.1: Table showing the values of the DH parameters for the UR5e robot arm, gotten from Univeral Robots official website.

parameter visualization. The kinematics of the UR5e robot is now possible to operate with, as the DH parameters are given.

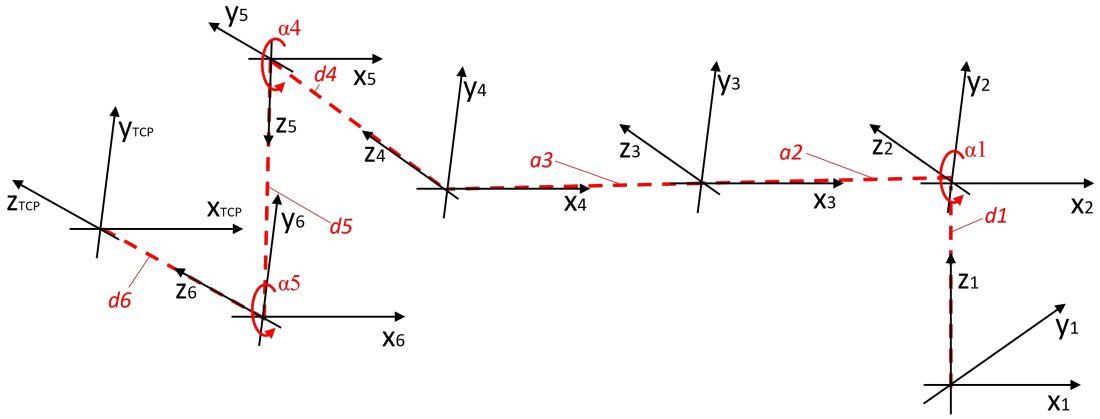


Fig. 6.2: Figure illustrating the DH parameters of the robot arm.

6.2 Python robotics system toolbox

For this thesis, the kinematics are handled using the robotics system toolbox (RST) package for Python. RST is a package used to do kinematics operations on an arbitrary robot, given its DH parameters. The reason why the kinematics operations are done using this package instead of the UR software itself is that it offers an online robot simulation that can be used to confirm that the robot makes the correct movements. This feature is important for this thesis, as NLP interpretation of natural language instructions might lead to unexpected movements of the robot arm. Therefore, all inverse kinematic operations are handled by the RST package. Figure 6.4 shows an in-depth explanation of the RST block shown in figure 6.1 at the start of this chapter.
For figure 6.4 it should be noted that the actions are passed into the RST kinematics pipeline, one action at a time. The figure shows how each action is processed. Path **(1)** shows that an action can be reduced to nothing. This happens if the action in the list of actions does not cause the robot to move but instead is a system manipulation action. For example, *"set position as point 1"* or *"set frame 3 as the current frame"*. Path **(2)** is for handling movement to points or frames. Path **(3)** is for handling the rotation of joints. actions moving the robot in reference to itself, such as stated in the instruction *"move 10 cm along the y-axis"* is also in path **(2)**, as the change in position is viewed by the system as a new point in space that the robot should move to. Though it is viewed as a new point in space, it is not stored in the database. The kinematics pipeline has access to the same database described in chapter 5. It, therefore, extracts locations from the database matching the destinations given in the action objects. This is the reason why the list of actions in figure 6.4 immediately transforms into numerical values in path **(2)** and **(3)**. Where

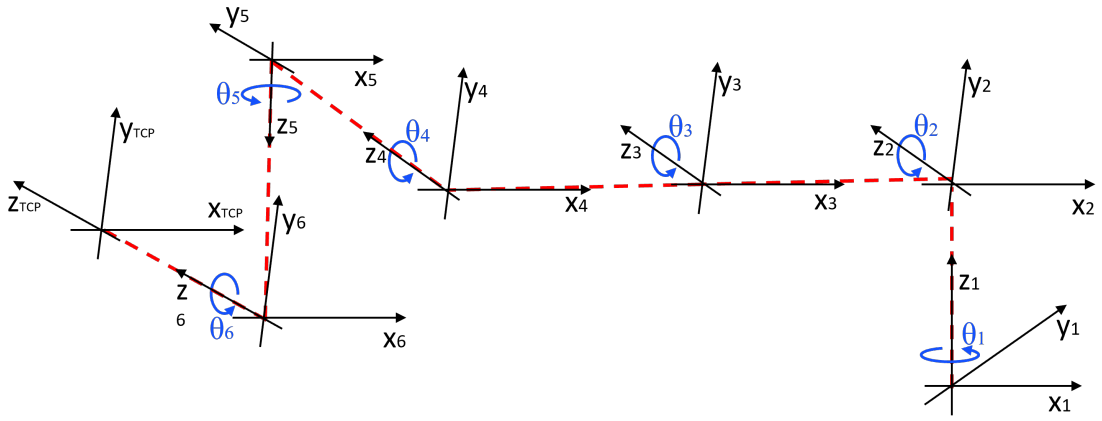


Fig. 6.3: Figure illustrating the movable joints as θ in the DH parameter visualization.

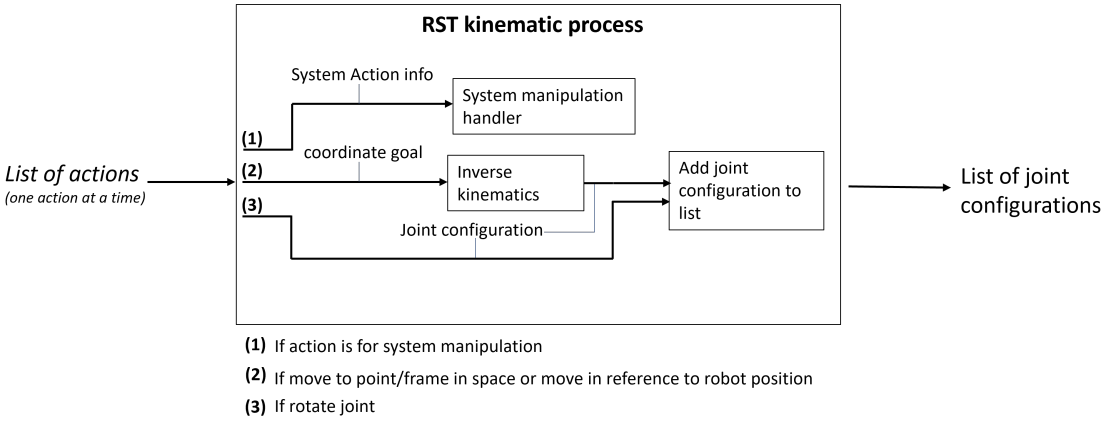


Fig. 6.4: Figure illustrating the inner workings of the RST pipeline the numbers indicate requirements for the action object to take the given path.

the actions become either a coordinate the robot should move to or a joint configuration that indicates a rotation of a single joint. The inverse kinematics are calculated using a numerical method called LM (Levenberg-Marquadt) with optimization made based on Chan's method. Though an analytical expression for the inverse kinematics can be created for the UR5e, it is not present in RST. Therefore, RST's inverse kinematics methods are used instead. The output of the RST process is a list of joint configurations. This is because if both the RST simulator and the UR API use joint configurations to move the robot, then the movements created by the two separate software should be similar. The exact movements might vary based on the velocity profiles of each joint, but this variation has been observed to be small enough, through testing, to be negligible. The exact tests that are referred to are the ones done in chapter 7. A video presentation of the online demonstration of the movement in comparison to the real robot is given at the link below.[4] The output from RST is passed into the RTDE package, which is used to communicate with the UR5e robot arm. This is discussed in the next section.

6.3 RTDE - python package

A C++ interface for RTDE (Real-Time Data Exchange) wrapped into Python is used to control the UR5e robot arm. It allows a programmer to use UR function commands from Python, given the IP address of the arm. For this specific thesis, two functions are used.

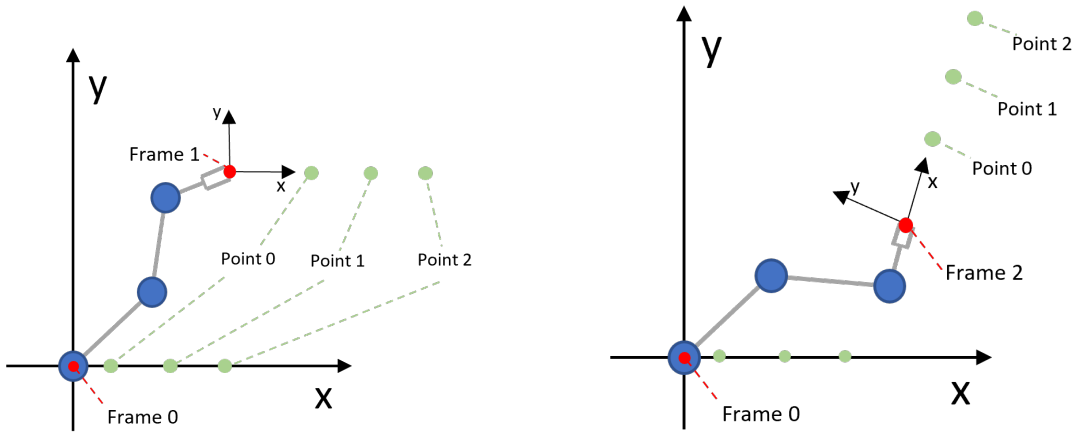
[4]https://www.youtube.com/watch?v=pzW4MFeVL18

- rtde_receive.getActualQ()
- rtde_control.moveJ(joint_config)

getActualQ() is a function that returns the joint positions of the robot. This function is used to synchronize the system's robot pose with the actual robot pose. This is important for the robot to move relative to its own positions and for the online simulation to start at the correct location. The function MoveJ(joint_config) is used to move the joints of the robot to the target joint configuration input, given as joint_config. As stated before, the reason why RTDE is not used for anything other than the necessary communication is to handle all information in the RST pipeline. Therefore, no more than two functions are used.

6.4 Points and frames creation

The last part of this chapter is about the mathematical operations done to create the points and frames in space and how RST is used to support the operations. The figure from chapter 3 is given again as figure 6.5 to show how the points and frames are linked together. What the figure shows is that points are created in reference to frame coordinate systems in general. Therefore, if a point is created 5 cm along the x-axis of frame 1, then it could also be used as a position 5 cm along the x-axis of any frame.



(a) Visualization of points in frame 1.

(b) Visualization of points in frame 2.

Fig. 6.5: 2D visualization of the point frame system created.

The mathematical operations needed to store the tool center point location in reference to the current frame that the system is working in are therefore given as follows:

$$\begin{bmatrix} {}^{A}p_{i+1} \\ 1 \end{bmatrix} = ({}^{O}_{A}T_{current})^{-1} \begin{bmatrix} {}^{O}p_{tcp} \\ 1 \end{bmatrix}$$

where ${}^{A}p_{i+1}$ is a new point in the system, in reference to the current working frame given as a vector. ${}^{O}_{A}T_{current}$ is the transformation from the base frame (marked as 'O') to the current frame (marked as 'A'), given as a 4×4 homogeneous transformation. ${}^{O}p_{tcp}$ is the current location of the tool center point in base frame coordinates. The point ${}^{O}p_{tcp}$ is calculated by using the RTDE function getActualQ() and then using a forward kinematics function in RST to calculate the tool center point location (disregarding orientation). The point ${}^{A}p_{i+1}$ is then written to the database and can be used as a position that the robot can move towards. To move to point ${}^{A}p$, it should be transformed back into base frame coordinates. This is done as

$$\begin{bmatrix} {}^{O}p_{tcp} \\ 1 \end{bmatrix} = {}^{O}_{A}T_{current} \begin{bmatrix} {}^{A}p_{i+1} \\ 1 \end{bmatrix}$$

Afterward, the point location ${}^{O}p_{tcp}$ is given as input for RST's numeric invers kinematics (along with the robot pose as the initial query). The output from the inverse kinematics function is a joint configuration passed into the RST simulator and then into the RTDE Python function moveJ and executed on the physical robot.
Frames are calculated in a similar way to how TCP location is calculated (using the joint configuration of the robot, then passing it through forward kinematics). The difference is that the orientation is not disregarded and is given along with the translation as a homogeneous transformation.

Chapter 7

System evaluation

The evaluation of the system is used to determine the effectiveness of the natural language instruction-based control. The focus of this test is verbal flexibility. The system is built on sets of keywords used to connect words to actions. In itself, it would mean that words that do not appear in the database would not be usable. But since the system makes use of a cosine similarity comparison method, words outside the database should be interpretable. Another focus is the NLP pipeline in general, as this pipeline is most likely to introduce errors as it solves a complex task (NLP) compared to the information transformation task the parser and the kinematics pipeline do.
To get natural language instructions, it is thought that the best approach is to use other humans as test users. These people cannot know what the system is capable of and are therefore not biased toward using words that appear in the database.
As the system was meant to be intuitive for the end user, it was evaluated on test users with no previous experience with the system. To avoid bias, the test users were not made aware of which words appeared in the database. This chapter explains how the test is set up and the results obtained.

7.1 Test setup

The test is a series of three tasks that the test users must solve using the robot manipulator. The test users are given a video of the robot solving the task, which is used to show what the user must make the robot do. The video presentation is used because it introduces no wording bias, as the user is not given any words during the presentation. The test users write an instruction intended to be executed on the robot to copy the movements seen in the video. The three tasks are shown in the video linked below.[1] Figure 7.1, 7.2 and 7.3 shows the step-by-step actions executed on the robot arm.

Task 1: Put the object on top of the other object:
- *Start in origin of frame 1.*
- Grab object at point 0.
- Lift the arm to point 2.
- Move the arm to point 1.
- Let go of object.

Fig. 7.1: List of actions done in task1.

The videos present a robot manipulating 2 to 3 objects in its environment. The tasks are designed to heavily use the 'move' action, as it is the most important action the system has. It was deemed a lower priority to get tasks that made the user do the meta action 'repeat' or the system actions 'set point' or 'set frame'. Therefore, due to time constraints, no tasks were made that used the given actions.
The instructions of the test users were collected and replayed on the system to test if the robot acted as intended.

[1] https://www.youtube.com/watch?v=zVLaS9Vfv1k

Task 2: Put pushed object on top of another object
- *Start in point 7.*
- Move robot to point 3.
- Push the objects by moving to point 4.
- Move out of the way, by going to point 5.
- Grab object at position 6.
- Lift object 10 centimetres up the z-axis.

Fig. 7.2: List of actions done in task2.

Task 3: Stack three objects on top of each other
- *Start in point 8.*
- Pickup object at point 9.
- Go back to point 8.
- Move to point 10.
- Rotate wrist 3 90 degrees.
- Let go of object.
- Move 20 centimetres up the z-axis.
- Move to point 8
- Move to point 13
- Then grab at point 11.
- Move back to point 8.
- Move to point 12.
- Rotate wrist 3 45 degrees.
- Put down object.
- move 5 centimetres up the z-axis.
- go back to point 8

Fig. 7.3: List of actions done in task3.

7.2 Test results

Figures 7.4, 7.5, 7.6 and 7.7 present graphs that show the test results. Each task was solved by 10 people. All the actions combined from all three tasks sum up to 29 actions. This gives a total of 290 actions tested on the system. The first graph shows the success ratio of all the action types throughout the test. The next three graphs show the success ratio of all individual actions in the given task. The cosine similarity threshold was set to 0. But through further investigation, it was shown that the lowest needed threshold for this sample data was 0.40. The lower similarity score came from the instruction "Slide to point 4", where the instruction was first accepted when the system guessed the word "slide" as "move" with a score of 0.40. Figure 7.8 shows the type of errors made, which will prove useful when discussing the nature of the errors and how to improve the system. Furthermore, figure 7.9 shows the number of errors produces, relative to successful commands.

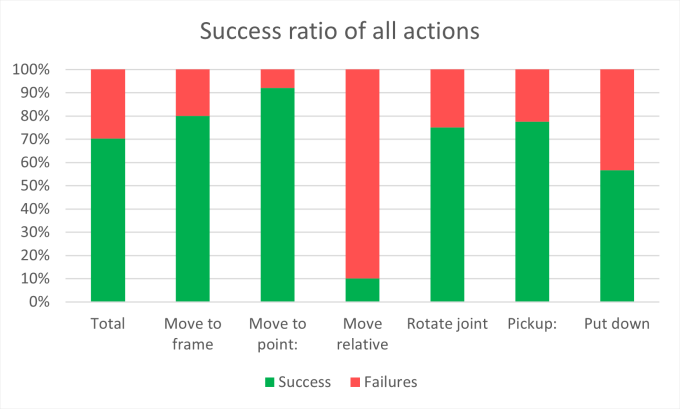


Fig. 7.4: Graph illustrating the success ratio of all actions in the whole test.

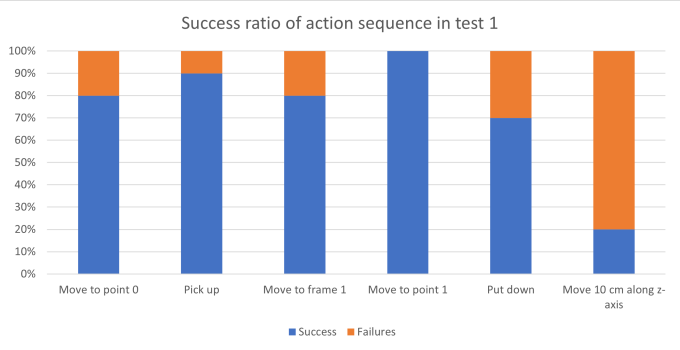


Fig. 7.5: success ratio of each action in task 1.

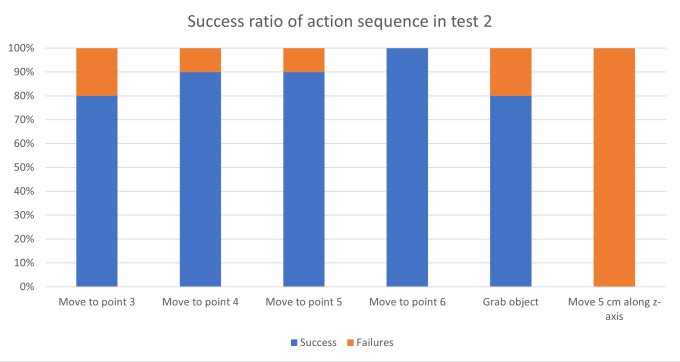


Fig. 7.6: Success ratio of each action in task 2.

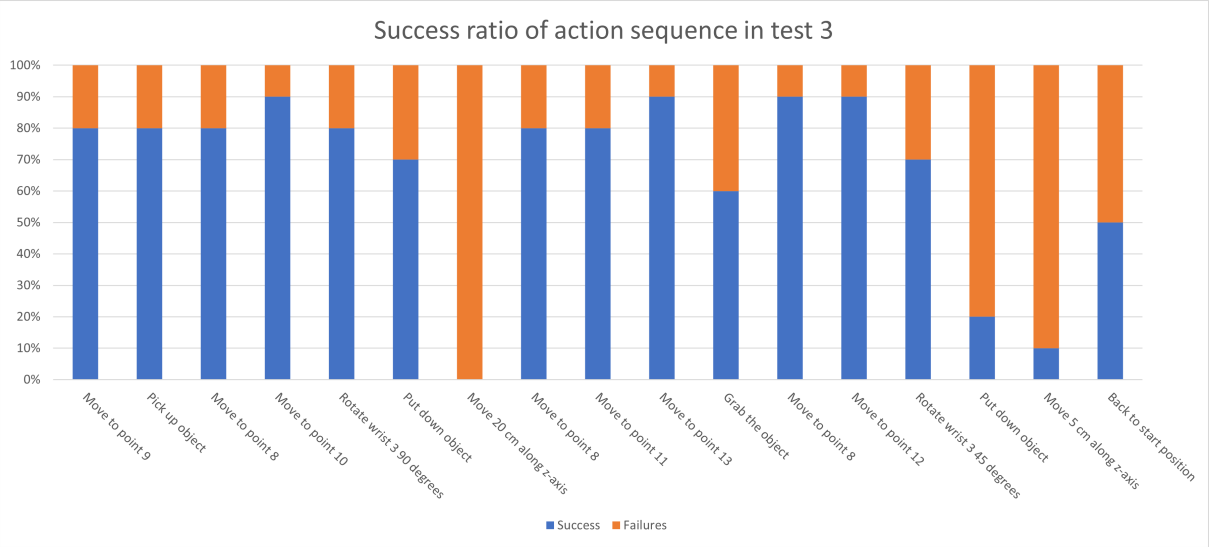


Fig. 7.7: Success ratio of each action in task 3.

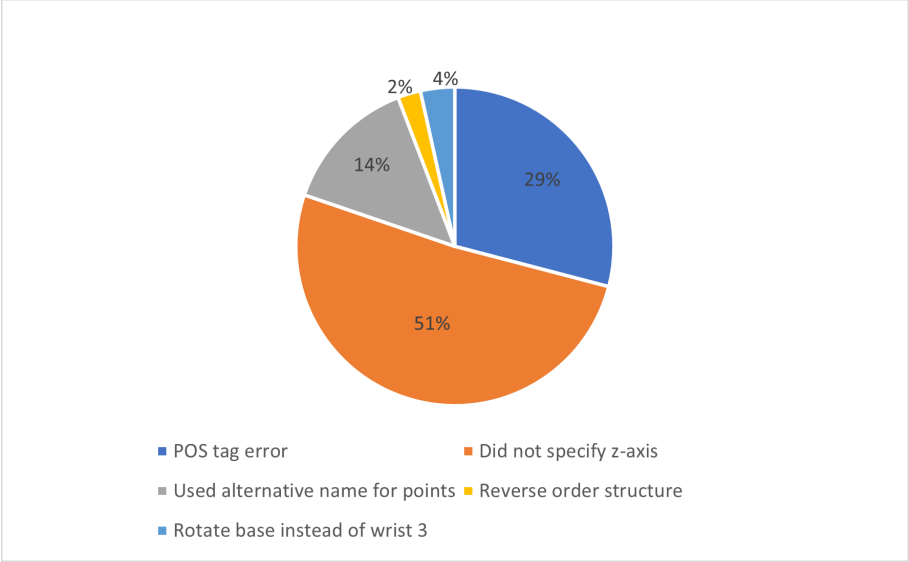


Fig. 7.8: Graph showing the type of errors made.

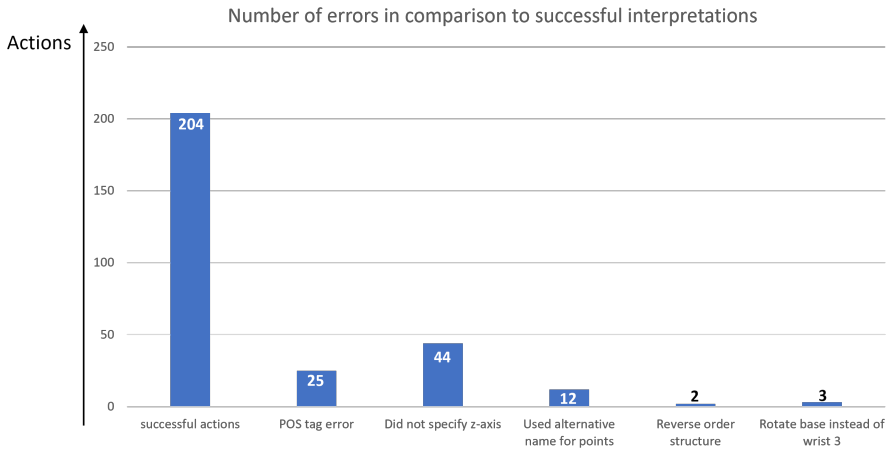


Fig. 7.9: Graph showing the number of errors made in comparison to successfull interpretations.

Chapter 8

Discussion

A discussion of the test is given in this chapter. The main focus is to discuss the system's capability to be supportive of natural language control. The three common errors of the test actions given in the previous chapter are used to discuss the system's natural language interpretation skills. Therefore, the first three sections are about the errors, while the last sections are a discussion of the system as a whole regarding the subject of natural language control.

8.1 Error when not specifying axis direction

Given the graph in figure 7.8 in the previous chapter, it is seen that the most common error is using the term 'up' to describe a movement in the z-axis. An example from the dataset is *"move 10 centimeters up"*. The system would be able to interpret a movement of 0.1 meters, but not in what direction. The adverb meaning of 'up' does not refer to any direction in the system. The words 'up' and 'down' require the robot to know how the user perceives the environment. It is therefore difficult to know what direction is meant and, therefore, how the words should be processed. It is possible to assign the word 'up' as a synonym for the z-axis of the work frame, but that would not guarantee a correct interpretation. Figure 8.1 shows that the z-axis points directly downward. In this particular system, 'up' should be interpreted as a negative z-axis movement.

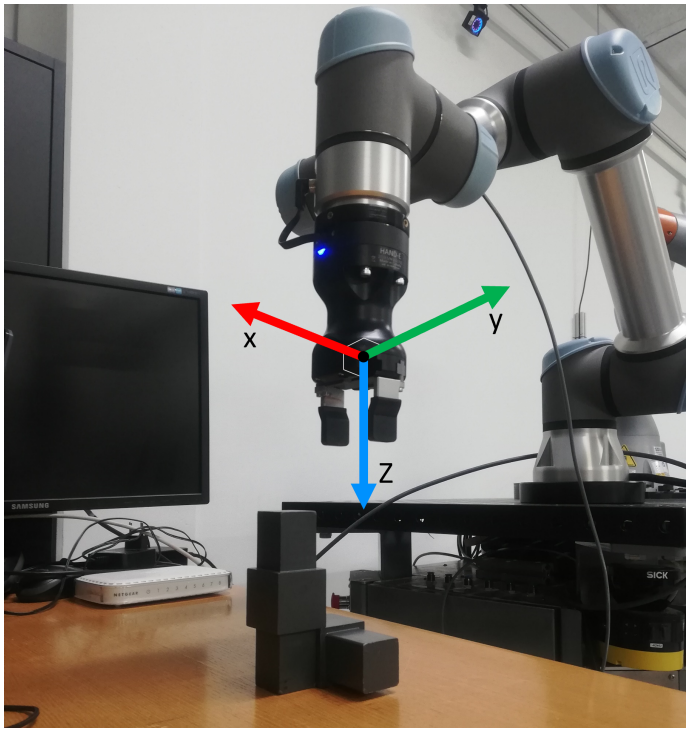


Fig. 8.1: Figure showing the axis of the TCP frame in the given work environment.

And if 'up' was assigned as a negative z-axis movement, then another TCP frame could be imagined where that would also be incorrect. Even if it was assigned to the z-axis of the base frame of the robot (which points upwards in reference to gravity, given the robot is mounted horizontally), then an environment could be imagined where the z-axis of the base frame was not pointing upwards in the workspace of the robot TCP. Figure 8.2 shows a task

where a robot is working on a skewed wooden plank. In this case, upwards is arguably the direction of the z-axis on the TCP frame instead.

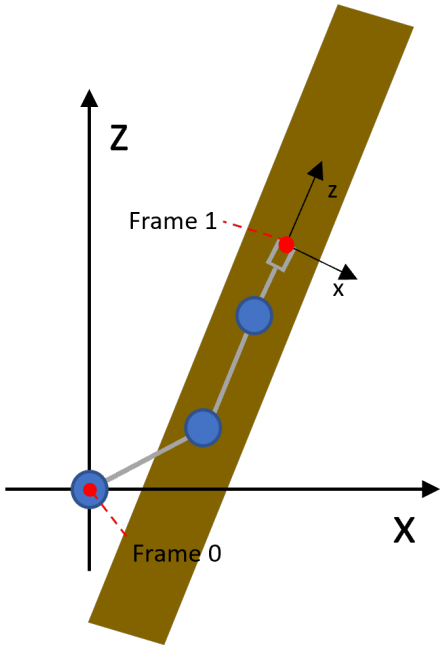


Fig. 8.2: Figure of a robot doing arbitrary work on a skewed wooden plank.

A possible fix is to make the system ask users what direction was meant and temporarily save it while working on the current frame. But this would make the system harder to control, as it could potentially force users to work using 3-dimensional vectors. It would be tedious if one had to specify an instruction as *'move 10 centimeters up, and up is the direction of the vector (x=0.48, y = 0.64, z = 0.6)'*. This type of ambiguous language is therefore left uninterpretable.

8.2 Errror made by the POS tagger

Another error is the POS-tag error. This is an NLP pipeline error where the words are assigned the incorrect grammatical categories. Examples of this error are shown in figure 8.3. The instruction in subfigure 8.3.a is intended to make the system do the gripping maneuver with its arm. Technically, the NLP pipeline is not wrong, as it could also be read as the object being a 'computer grip'. Subfigure 8.3.b is less intuitive but still correct. The system perceives it as the user telling the robot that there is a gripper close by. Instead of the user instructing the robot to close its gripper. Since the key verbs are perceived as nouns, they go undetected by the parser pipeline and are therefore uninterpretable by the system. This cannot be fixed quickly, as it would require training the language model on a new dataset, which would make the language model prone to looking at sentences as instructions. This would be a time-consuming task, which is why it is left as it is.
The POS tagger is still greater than a simple word detector, as it gives the correct grammatical tags to words with many meanings. The word 'point' has been used throughout the thesis as a location in space, but it could also be perceived as a verb. The POS tagger can then be used to analyze the sentence given in figure 8.4. The verb is then used to instruct the robot, and the noun is used as the destination. It should be noted that the sentence serves as an example and that the system is not capable of orienting itself towards points, as indicated.

Computer	grip
noun	noun

(a)

Robot	close	gripper
noun	noun	noun

(b)

Fig. 8.3: Figure of two tables, where the upper words are instruction words, and the lower words are the grammatical categories assigned by the NLP pipeline.

Point	the	gripper	towards	point	1
Verb	Determinant	Noun	Preposition	Noun	Cardinal number

Fig. 8.4: Figure of a sentence where the word 'point' is used for two different meanings.

8.3 Error made when using other words instead of points

The last error is the use of alternative names instead of points to move to positions. An example from the data is the instruction *'go back to the starting position'*. In this instruction, it is unknown for the system where the 'starting position' is located. A fix to this problem might be to assign names to the points at creation so the user has the ability to call the points by the names set. This would, however, require the system to support movements to destinations specified by arbitrary names. This is not supported by the system, as the parser pipeline only creates move actions to destinations given as nouns (points and frames). Setting up a system that could support point naming would then require a new iteration of the system parser pipeline, where the instruction could contain an arbitrary name that the parser pipeline had to detect. An example of why this task is so difficult could be if one were to name a point 'move'. The parser pipeline would then have to be able to interpret the word 'move' as a point and not as a verb while knowing that the verb 'move' sometimes also correlates to the 'move' action. This is the reason why the given event is left uninterpretable.

8.4 limitations

This section explains the overall limitations which are observed within the system given the test evaluation. It is shown that the system has a clear lack ability of to interpret context-based information. This limitation comes from the design of the parser pipeline, as it can only parse a limited set of formulations. Furthermore, the system fails to interpret some inputs, as it relies on the language model to deliver the correct information. Therefore, if the language model fails, for any reason, then it would likely mean, that the rest of the pipeline would fail as well. A limitation of the action design created is the limited amount of information passed from the parser into the kinematics pipeline. It is possible to expand the complexity of each instruction. But the parser pipeline would also become increasingly complex, as it would need to extract more information from the language model.

The limitations explained, mostly reside within the parser pipeline, where it is observed that the extraction method is insufficient, as it cannot interpret all commands given by the test users. Solutions to the issue, unbound by the structure and design of natural language processing in this thesis, include using self-trained neural networks to map commands into the actions themselves. This method has the feature, that the language used by the test users is directly trained on to gain the actions needed. A method for foreseeing the input structure and processing it would

then become unnecessary. In theory, all context-based commands would then be mapped to the actual actions, given that the system is trained enough to interpret the input. The main drawback of this method is the lack of datasets. Given the niche field of using Natural Language to control a 6-DOF manipulator with a parallel gripper, there is little training data available. Training data would therefore be needed to be developed by hand. Another drawback is also the lack of flexibility, as the system would be bound to the actions present at the training. Developing new actions would then require new training in the system. Assuming available training data, it would be possible to train the system to become greater at interpreting the natural language commands given in the test evaluation.

8.5 Evaluation of natural language support

The test users were observed to prefer natural names for points and adverbs to describe relative movements. Some also did not specify movement distance and instead used instructions like 'textitmove up' or '*move away from the block*'. The analysis of the errors from the test results also shows that most of the errors stem from the user specifying the instructions using words regarding the work environment like '*move up to the initial height*' or '*go back to the starting position*'. As the specification could mean different positions given interpretation, the system is not capable of interpreting the instructions. Excluding moving relatively, then the system has good test results interpreting actions that did not need detailed specification, like '*open the gripper*', or '*move to point 0*'. Where good test results mean around 70% success rate.

Chapter 9

Conclusion

For this thesis, the problem has been given:

if a system can be developed to use natural language input to control a UR5e robot arm equipped with a Hand-E gripper.

A system has been developed that consists of three pipelines. The system starts with a natural language processing (NLP) pipeline that uses the stanza package to implement neural network language models used to analyze the input sentences and output instructions accompanied by information regarding sentence structure and POS-tags. Afterward, a parser pipeline was designed and developed to interpret the NLP pipeline output and extract the actions within it using grammatical rules. A set of actions was developed to move the robot manipulator, consisting of a UR5e robot arm equipped with a Hand-E gripper. The actions were then passed into a kinematics pipeline, tasked with executing the given actions. Furthermore, the kinematics pipeline also calculated the position and pose of points and frames in space, respectively.

The evaluation was made based on the results of the system tests, which showed that it was capable of extracting actions from natural language instructions with a success rate of 70%. But lacked the ability to interpret instructions with specifications relative to the given environment, like *"move up to the height of the starting position of the task*. It was stated in the problem constraints, that context based commands was out of scope for this thesis. But that ultimately meant, that the system became harder to control for the test users. The system solved the problem formulation given the contraint set on the thesis. However, based on the evaluation results, then the 70% success rate could be increased, if context-based commands was supported.

Chapter 10

Future work

This thesis builds a foundation for future work in many directions in the field of NLP for robot control. Specifically, NLP control that uses grammar analysis methods. The possible expansions of the system is based on the observations made in the discussion, chapter 8.

For language analysis using neural networks, it might prove useful to fine-tune an encoder language model with instruction-type sentences. This is due to the observation in the discussion, chapter 8, that the natural language model used, is prone to choosing noun interpretation of words, where errors would have been avoided if the model chose verb interpretation. Like in the sentence *robot grip*, where the system sees both words as nouns.

Contextual instructions might be interpretable by the parser if more information was stored in the system database. The parser would then match context-based instructions like *'Move up to the height of the start point of the task.'* with context knowledge in the database for instruction interpretation and finally instruction execution.

Another field that could be expanded upon is kinematic actions. More actions could include object avoidance, movement velocity specifications, and workspace constraints. These actions could be used to solve more complex tasks than those given in the thesis and to make the robot arm safer in stricter environments.

Another implementation that could be added for future work is a decoder language model used as a chatbot. The reason for using a decoder model is for exploiting the unsupervised learning technique of decoder models. Which is training the model on arbitrary unlabeled text. The model could then train on significantly more data, than when training encoders, as the data for the decoder does not need to be tagged in comparison to the encoder model data. The result would be a decoder model that potentially is capable of processing complex natural language.
It could be implemented for communication with the user to make communication more natural than the user communication protocol in this thesis.
A likely area of the thesis where a decoder model could be of use, is within the parser. The parser uses grammatical information to find several high-level representations of instructions. This includes verbs describing environment manipulation, nouns serving as subjects of the instructions, nouns describing destinations, cardinal numbers describing units or point indexes, and more. The decoder model could aid the parser in finding these representations, by highlighting areas within the sentence where specific words might have a high importance. It could also be used to pick a candidate, given that several words match the criteria that the parser is looking for. This method could enable the parser to process more complex information, and maybe handle context-based instructions, as the decoder might be able to reformulate the context-based instructions into simpler instructions that the parser can interpret.

The thesis holds many areas for future work, summarized as the implementation of more analytical processing tools, expansion of the system's kinematic action repertoire, and implementation of AI decoder models for NLP processing.

Bibliography

[1] Olivares-Alarcos, A., "On inferring intentions in shared tasks for industrial collaborative robots," *Electronics*, vol. 8, Nov. 2019.

[2] Wilhelm, B., Manfred, B., Braun, M., Rally, P., and Scholtz, O., *Lightweight robots in manual assembly – best to start simply! examining companies' initial experiences with lightweight robots*, Oct. 2016.

[3] Wu, S.-H. and Hong, X.-S., "Integrating computer vision and natural language instruction for collaborative robot human-robot interaction," in *2020 International Automatic Control Conference (CACS)*, 2020, pp. 1–5. DOI: 10.1109/CACS50047.2020.9289768.

[4] Matuszek, C., Herbst, E., Zettlemoyer, L., and Fox, D., "Learning to parse natural language commands to a robot control system," in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, Desai, J. P., Dudek, G., Khatib, O., and Kumar, V., Eds. Heidelberg: Springer International Publishing, 2013, pp. 403–415, ISBN: 978-3-319-00065-7. DOI: 10.1007/978-3-319-00065-7_28. [Online]. Available: https://doi.org/10.1007/978-3-319-00065-7_28.

[5] Kahuttanaseth, W., Dressler, A., and Netramai, C., "Commanding mobile robot movement based on natural language processing with rnn encoder-decoder," in *2018 5th International Conference on Business and Industrial Research (ICBIR)*, 2018, pp. 161–166. DOI: 10.1109/ICBIR.2018.8391185.

[6] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R., "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[7] Vaswani, A., Shazeer, N., Parmar, N., *et al.*, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf.

[8] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. N., "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805.

[9] Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D., "Stanza: A Python natural language processing toolkit for many human languages," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020. [Online]. Available: https://nlp.stanford.edu/pubs/qi2020stanza.pdf.