# Choosing the model and the dataset used for POS tagging:

**Arxiv paper:** [[2206.13289] Analyzing Encoded Concepts in Transformer Language Models (arxiv.org)](https://arxiv.org/abs/2206.13289)

Model: Finetuned & pretrained bert model

Dataset: Penn treebank (Marcus et al., 1993)

The model & Dataset is chosen for two reasons. First, the data is chosen, such to get a comprehensive and easy to understand/parse grammatical categorization. This makes it easier to build a parser which can figure out how to translate the output to robot language.

The other reason for the chosen model & Dataset is also to make sure it is a bert model which is chosen (pretrained transformer), as this then correlates with, he learned ideas, which should be explained in the bachelor thesis. It is also possible to use a statistical model (don’t know what that is), but even if the output is solid, then the theory behind would be lacking.

**CON:**

It seems that the dataset POS tag penn treebank is a payed for dataset, which means I don’t have access to it.

A picture containing table

Description automatically generated

Figure 1 - Example output post processed

# Choosing to use URsim on ubuntu to “feel” out how the URscripts can manipulate the UR robot.

**Errors which should be noted**

Before it worked, then there were some errors in the program which must be dealt with, the errors are not written within the online guide.

When running ./install.sh, then there is an error because there is 2 packages that are no longer available on newer versions of linux, namely libcurl3 and lib32gcc1.

To deal with this problem you must go into the install.sh file, and change the names to:

libcurl3 -> libcurl4

Text

Description automatically generated

lib32gcc1 -> lib32gcc-s1

Text

Description automatically generated

these files are available substitutes, which will now be downloaded when the install.sh file is run. Alternatively, you might need to download the newer files manually, but this is easily done, as they are available as terminal instructions.

**Reasons to using URsim**

URsim is used to have a robot available which can run URscripts. Therefore it enables the ability to study robot interaction without the need to be in the I4.0 lab, and with guarantee to not damage the UR arm while experimenting.

# Choosing to use UR\_rtde

UR\_rtde is a library for remote controlling the Ur robot. This is a really obvious good choice to make, as it enables easy interfacing of the ur robot using python.

Pros:

Easy to use python to ur robot control

Cons:

I cannot make my own implementation, oh noo, how baad.

# Using a class structure/Object oriented design to create the parser

It could be cool to create a class structure design, as it would mean cleaner and intuitive programming. With object-oriented design, a more iterative approach can then be used (adding a new class method or each new feature), which would work well with the fact, that I know jack-shit about URScripting as of writing.

Pros:

Better designed program for handling both the problem, and for handling my lack of experience with the given subject.

Cons:

Is a constriction in how I am capable of navigating while programming.

Using classes is more constricted than writing everything in one block But it will look a hell of a lot cleaner.

# Using text files to handle grammatical rules and remember different words

It is definitely possible to use more sophisticated methods of handling data which should be remembered (mySQL databases, excel files), but the reason for choosing .txt files is for the simplicity of the project. This project will not be determined by the cuteness of the database implementation, so the quickest and easiest solution will be the best to work with, in this project

Pros:

Easy and intuitive and debuggable in python

Cons:

Not an actual database, primitive.

# Directly avoiding using letters as placeholders for positions

This decision is made, because of the duality of using the letter a in a sentence. Since it needs to be able to support spoken language, it is not possible to know the difference between big letter A, and small letter A, and the neural network model usually mistakes the small letter for the determiner class instead of the noun class which it actually is.

Example:

Move to point a

The NLP processes this as half a sentence, where a is a determiner, rather than seeing it as a noun.

*Example of the other half of the sentence*

Move to point a stick towards a door.

Where a is the determiner for the noun “stick”

Will be solved by not using letter a as position

# Choosing to use a verb and noun stack for programming design

The idea of using stacks to represent the verbs and the nouns, is to enable the verbs and nouns to be in any order without needing a new representation detection form.

# Choosing to use dependency parsing

For a long time, it has only been assumed that we would use POS tagging, but since we introduced to types of ambiguity (grammar categorization and pronoun hell), then it would also be nice to have an AI pipeline which could deal with the second ambiguity problem. This is where the dependency parsing comes in handy. The thesis can then be written with the structure of “there is these problems with natural language in regards of using it as a command language, these methods systemize and solve these problem, am I now capable of using natural language as a command language?”

# Choosing Stanza over the other models or a majority vote

Stanza was chosen as it had an intuitive dependency parsing routing which was easy to comprehend. This made it possible to decipher the meaning of the sentences with an algorithm like strategy, which is exactly the kind of intuition needed to base the parser on.

The reason why the majority vote is bad, is because all the different models (spacy, standford, stanza) use different dependency parsing models. Therefore, it makes no sense to compare the different models if we know that the stanza is generally easier to work with.

# Working with robotics system toolbox

Robotics system toolbox allows me to use the UR5 robot as its one of their packages. Therefore, the robot can be directly manipulated within the software to create a demonstration of the given movement before deploying the movement on a real life robot. Though for this to work, it is not possible to use UR invers kinematics anymore, as it would deviate from the robotics toolbox movement demonstration. Invers kinematics is then calculated on the PC using the robotics system toolbox whereafter values for each of the joints are given which is then inserted the UR-robot for movement.

# Version 1 of system diagram

Diagram, schematic

Description automatically generated

Essentially this system design focuses on the modules (RBT and stanza) being upgrade from the parser. So basically, we could have the parser directly analysing the text interface data, and then directly manipulating the UR-arm. But it makes more sense to extract more information out of the text data using the stanza AI, and using the robotics system toolbox to better describe and create the movements needed. So all in all, the system could be done only having text interface -> parser -> UR-arm, but the complexity of both movement and text input would be minimal. These complexities has become heightened as the two upgrade pipelines enable greater control.

# From parser to Ur arm through robotics system toolbox design

Diagram

Description automatically generated

The first picture is a simplistic version, as the picture of the **whole system diagram**. The lower is a “zoomed in” version, where the actual functionality and design implementation is illustrated. This specific **design** is chosen to optimize where and how many places needs the **“Complexity expansion”** task. This means, that all other places not marked with **“complexity expansion”** would in theory stay the same, no matter how many different **Atomic/Complex actions** is added to the system. It also is created to highlight the **properties** of each block. The RST block is by far the greater **calculator** and is therefore much greater for the assignment of merging and creating the actual deployable movements. The parser on the other hand, is far greater at knowing the **rhetoric meaning** of the sentence. It is therefore in this region, that all the core **movement complexities** are created in a hypothetical level, such that it can assimilate the actual command best.

What is not shown is the **databases**, which the **parser** will be able to extract key information from, which is used to **deploy** already created **complex actions** from **memory**.

# Using frame terminology for default locations

Frames with numbers is an easy NLP pipeline word for location. This is in contrast for point, which could both be a noun and a verb. The other core reason is to strengthen the idea, that it is not only a position, but a whole transformations. Which means each position also has transformation properties, this ability would likely be hidden, if points would be used as the terminology instead