Human Language to Robot Actions: Comparing Traditional NLP Methods and LLM Approaches for Autonomous Task Execution

Anonymous ACL submission

Abstract

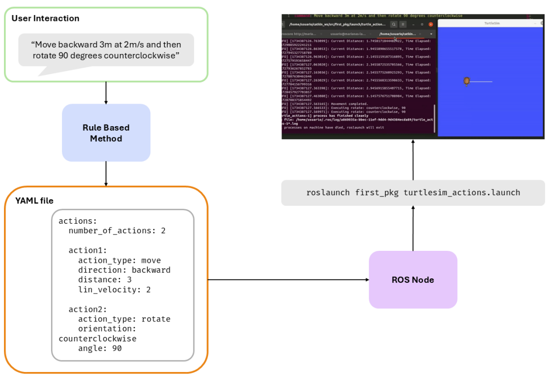
Robots operating in real-world environments must effectively interpret and execute commands provided in natural language, such as "bring me the cup of coffee" or "clean the sink." The challenge lies in bridging the gap between the ambiguity of human language and the precision required for robotic actions. While modern approaches leverage Large Language Models (LLMs) to generate executable code from commands, traditional Natural Language Processing (NLP) methods like Part-of-Speech (POS) tagging and Context-Free Grammars (CFGs) offer rule-based alternatives. This project compares the effectiveness of traditional NLP methods with LLM-based approaches, such as ROS-GPT using GPT 3.5 turbo, in linking commands to robot actions. We highlight the strengths and limitations of both methods, offering insights into their trade-offs.

1 Introduction

The ability of robots to understand and execute natural language commands is a critical step in bridging the gap between humans and robots in real-world environments. Instructions like “move the cup 20cm to the left” demand not only linguistic comprehension but also precise execution of physical tasks. However, translating natural language into robot actions is challenging due to the inherent ambiguity, variability, and context-dependence of human language, which contrasts sharply with the structured precision required by robotics.

Recent advancements in LLMs have enabled the direct generation of executable code from natural language inputs, offering a promising approach to this problem. However, these models are resource-intensive and may lack transparency, raising the question of whether traditional, rule-based NLP techniques can serve as competitive alternatives.

In this project, we test how effective are traditional NLP methods, such as CFGs, in translating natural language prompts into robot actions compared to modern LLM-based approaches. We developed a CFG-based system with word sense disambiguation to transform natural language commands into code that can be interpreted by the robot controller and compare its performance against an LLM approach. By evaluating both approaches on identical tasks, we aim to assess their respective strengths, limitations, and potential for practical deployment in robotics.


User Interaction
“Move backward 3m at 2m/s and then rotate 90 degrees counterclockwise”
Rule Based Method
YAML file
ROS Node
TurtleSim

```
actions:
  number_of_actions: 2

  action1:
    action_type: move
    direction: backward
    distance: 3
    lin_velocity: 2

  action2:
    action_type: rotate
    orientation:
counterclockwise
    angle: 90
```

```
roslaunch first_pkg turtlesim_actions.launch
```

```
1  command: Move backward 3m at 2m/s and then rotate 90 degrees counterclockwise
/home/usuario/catkin_ws/src/first_pkg/launch/turtle_action...
roscore http://maria...   usuario@marianas-la...   /home/usuario/catkin...
FO] [1734387126.763099]: Current Distance: 1.7458171844482422, Time Elapsed:
729085922241211
FO] [1734387126.863053]: Current Distance: 1.9455890655517578, Time Elapsed:
727945327758789
FO] [1734387126.963014]: Current Distance: 2.1455159187316895, Time Elapsed:
727579593658447
FO] [1734387127.063020]: Current Distance: 2.3455872535705566, Time Elapsed:
727936267852783
FO] [1734387127.163036]: Current Distance: 2.5455775260925293, Time Elapsed:
727887630462646
FO] [1734387127.263029]: Current Distance: 2.7455568313598633, Time Elapsed:
727784156799316
FO] [1734387127.363398]: Current Distance: 2.9456915855407715, Time Elapsed:
728457927703857
FO] [1734387127.463080]: Current Distance: 3.1457576751708984, Time Elapsed:
728788375854492
FO] [1734387127.563165]: Movement completed.
FO] [1734387127.566533]: Executing rotate: counterclockwise, 90
FO] [1734387127.569971]: Executing rotate: counterclockwise, 90
rtle_actions-1] process has finished cleanly
 file: /home/usuario/.ros/log/a860035a-bbec-11ef-9dd4-9d4384ecda89/turtle_act
s-1*.log
 processes on machine have died, roslaunch will exit
```

Figure 1: User input to action commands

2 Related Work

The task of interpreting natural language commands for robotic systems has been widely studied, with approaches ranging from rule-based methods to modern machine-learning techniques. One of the earliest examples on this field was the Winograd’s SHRDLU system (Winograd, 1971), which used rule-based procedures to interpret and execute commands in a simulated blocks world. This work demonstrated the potential of combining syntactic parsing with predefined representations to drive robotic actions.

Subsequent advances focused on interpreting natural language directions in real-world settings (Kollar et al., 2010), introducing a framework for spatial language that enabled robots to follow navigational instructions by integrating language with geometric reasoning.

The advent of LLMs has revolutionized the way robots interpret and act on human language. ROSGPT (Koubaa, 2023) and CoPAL (Joublin et al., 2024) exemplify this shift by integrating GPT-based models to generate executable robot plans. ROSGPT leverages GPT-3.5 to enable human-robot interaction highlighting its flexibility in understanding unstructured inputs and parsing to a ROS-compatible format to be used by ROS commands. On the other hand, CoPAL focuses on corrective planning of robot actions, allowing plans to be refined dynamically using natural language corrections.

Recent innovations like Code as Policies (Liang et al., 2023) represent a further step by directly generating executable robot programs from high-level instructions. This approach allows for compositional generalization, where robots can perform new tasks by combining known primitives based on the generated code.

3 Method

To compare the performance of LLMs and Rule Based Methods (RBMs) for the task of command generation in robotic implementations, we have proceeded as follows.

First, we selected YAML as the format for code generation. YAML is a human-readable data serialization format frequently used for configuring applications and systems (YAML Organization, 2021). It features a simple, clean syntax that emphasizes readability and structure through indentation, making it accessible to both humans and machines.

In robotics, YAML is commonly used to define configurations, behaviors, and sequences of commands due to its readability and compatibility with the Robot Operating System (ROS) and other frameworks. YAML effectively serves as a bridge between humans and robots by structuring commands that robots can execute, which is why it was chosen as the appropriate format for this task.

Our ultimate goal is to develop a pipeline that begins with natural language commands, which are then processed by a model to generate the corresponding YAML code. This YAML code is subsequently fed into an existing ROS node, which contains all the possible actions of a robot in a virtual environment. The actions of the robot are then observed visually.

3.1 Dataset and Testing

The dataset used in this project was specifically generated by an expert in robotics due to the specialized nature of the task and the unique requirements of the robot in question. Since no pre-existing datasets were available for this particular robot, the syntax was developed based on sequences created by a Master's student focused in robotics at McGill University with experience in YAML, ROS and the turtlesim simulation. This expertise ensured that the dataset accurately reflected all the possible actions the robot could perform within the virtual environment, providing foundation for the system's development and evaluation.

ROS (Robot Operating System) is widely used in the robotics community due to its flexibility and comprehensive set of tools for building and controlling robotic systems. In this project, the final YAML file, containing all possible commands that the robot in the simulation can perform, is read by a ROS-Python node. For simplicity and practical reasons, and since the tests in the ROSGPT paper were conducted using the turtlesim simulation, we used the same setup provided by ROS for testing. The turtlesim package in ROS is a simple 2D graphical simulator used primarily for testing and learning about the Robot Operating System. It simulates a virtual robot, represented by a turtle, which can perform basic tasks such as moving forward, rotating, and changing its color. The purpose of turtlesim is to provide a beginner-friendly environment for learning how to control robots within ROS. We ran the experiments using ROS1 with the Noetic distribution on Ubuntu 20.04. The Python node reads the *actions.yaml* file containing the actions for the corresponding command, analyzes which actions need to be performed, and executes them accordingly. To initialize the YAML file with the ROS node, we used a launch file called *turtlesim_actions.launch*.

The quality of the parsing from natural language to robot commands is evaluated on two levels: first, by examining the generated YAML code, which is easily interpretable by humans, and second, by verifying that the robot's actions align with the commands it was given. Similarly to what is implemented in other papers (González-Santamarta et al., 2023), the metric we are using has to be nuanced. Indeed, creating a syntax able to parse the entire universe of possible syntactic constructions in English into code would be a challenging task. Furthermore, we are limited by the range of possible actions the robot can take in the simulated environment. Therefore, a trial in our experiment is considered successful if the prompt is in accordance with the yaml code (i.e. the information conveyed is the same in both), and the robot is acting in accordance with the command given in the virtual environment. A statistical success rate would be the number of successfull trials over the number of all trials.

3.2 The Rule Based Method

The first method we implemented is the "Rule-Based Method" (RBM) approach for parsing natural language into YAML code. This process is carried out through the following steps:

1. The natural language input is first parsed by removing punctuation, unless the punctuation is numerically relevant (e.g., "0.5" is retained as "0.5," not "05").

2. Each verb in the command is then resolved to one of five predefined robot actions: *"move"*, *"rotate"*, *"change color"*, *"pen down"*, *"draw"*.

3. This mapping is achieved by calculating the cosine similarity between the embedding of each verb in the command and the embeddings of the five predefined verbs, returning the verb with the highest cosine similarity.

4. Once the verb is identified, a CFG is used to parse the entire command into a series of parameters corresponding to the selected action. For example, the information provided in the context of the verb "move" will be used to populate the *"distance"*, *"velocity"*, and *"direction"* parameters associated with the "move" action.

5. Once a successful parsing is achieved, the final list of actions is outputted in YAML format. Examples of this implementation are provided in the Results section of this paper.

Figure 2 illustrates the structure of the RBM used in this process.

3.3 The LLM based method

We aimed to replicate the implementation described in the ROSGPT paper (Koubaa, 2023). ROSGPT is a fine-tuned version of GPT-3.5 that integrates with ROS.

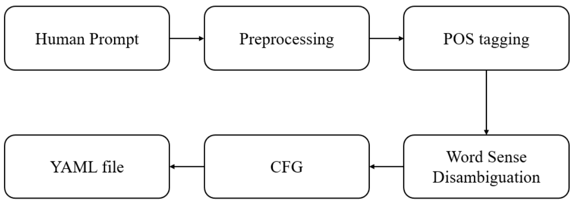


Figure 2: Rule Based Method Structure

However, the original server on which ROSGPT was hosted is no longer available for external use. To address this limitation, we implemented our own iteration of ROSGPT using GPT-3.5 Turbo. Instead of relying solely on the original architecture, we fine-tuned our model using a two-shot learning approach before feeding it the entire list of commands. Notably, we observed that the results produced by zero-shot learning were significantly poorer in comparison.

Below is an extract of the task inputted to our LLM :

Send the input command to GPT-3.5 and generate the actions in the desired format. If some values are not specified, just assume a baseline value of 1 for any parameter for which there is missing information. For every verb that is inputed, note that you can only resolve that verb to one of 5 options : rotate, move, change_color, draw, or pen_down. Any other verb would have to be first switched to the one it corresponds the most to in this list, then parsed in yaml.

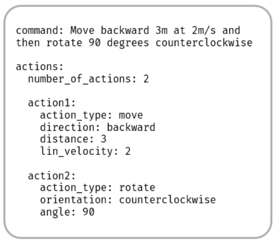
4 Results

We evaluated 15 prompts (human commands) using both the RBM and LLM approaches. The resulting YAML outputs were tested by inputting them into the robot code to determine whether the robot could correctly interpret and execute the specified commands.

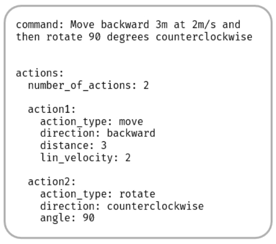
Below, we present a sample of the most significant results observed during the evaluation process:

Figure 3 displays the YAML outputs generated by both the RBM and LLM approaches. In this example, both models perform equally well, producing the expected results when the YAML output is executed by the robot code to carry out the task.

For the case shown in Figure 5, the RBM correctly executes the task of drawing a square since "square" is part of the CFG. The corresponding YAML actions generated by the RBM are shown



```
command: Move backward 3m at 2m/s and
then rotate 90 degrees counterclockwise

actions:
  number_of_actions: 2

  action1:
    action_type: move
    direction: backward
    distance: 3
    lin_velocity: 2

  action2:
    action_type: rotate
    orientation: counterclockwise
    angle: 90
```



```
command: Move backward 3m at 2m/s and
then rotate 90 degrees counterclockwise

actions:
  number_of_actions: 2

  action1:
    action_type: move
    direction: backward
    distance: 3
    lin_velocity: 2

  action2:
    action_type: rotate
    direction: counterclockwise
    angle: 90
```

Figure 3: YAML output of RBM in the right, LLM on the left for the command "Move backward 3m at 2m/s and then rotate 90 degrees counterclockwise".

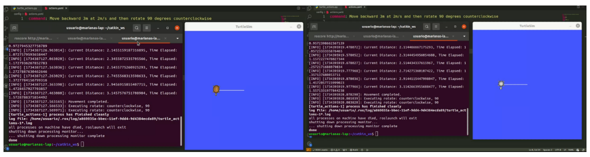


Figure 4: Comparison of results for the command "Move backward 3m at 2m/s and then rotate 90 degrees counterclockwise". The left side shows the output of the RBM, while the right side presents the output of the LLM.

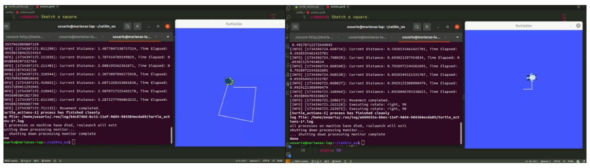
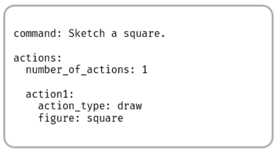
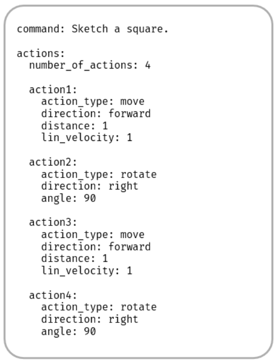


Figure 5: Comparison of results for the command "Sketch a square". The left side shows the output of the RBM, while the right side presents the output of the LLM.



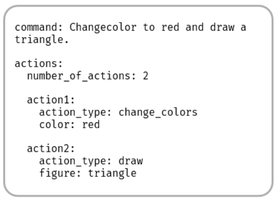
```
command: Sketch a square.

actions:
  number_of_actions: 1

  action1:
    action_type: draw
    figure: square
```



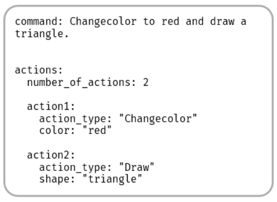
```
command: Sketch a square.

actions:
  number_of_actions: 4

  action1:
    action_type: move
    direction: forward
    distance: 1
    lin_velocity: 1

  action2:
    action_type: rotate
    direction: right
    angle: 90

  action3:
    action_type: move
    direction: forward
    distance: 1
    lin_velocity: 1

  action4:
    action_type: rotate
    direction: right
    angle: 90
```

Figure 6: YAML output of RBM in the right, LLM on the left for the command "Sketch a square".

in Figure 6. In contrast, the LLM outputs four separate instructions, including "move" and "rotate." However, it does not generate all the necessary movements to complete the square. Instead of using the direct "draw" command along with the "square" figure, the LLM opts for individual movements, despite being designed to select the "draw" command when appropriate.



```
command: Changecolor to red and draw a
triangle.

actions:
  number_of_actions: 2

  action1:
    action_type: change_colors
    color: red

  action2:
    action_type: draw
    figure: triangle
```



```
command: Changecolor to red and draw a
triangle.

actions:
  number_of_actions: 2

  action1:
    action_type: "Changecolor"
    color: "red"

  action2:
    action_type: "Draw"
    shape: "triangle"
```

Figure 7: YAML output of RBM in the right, LLM on the left for the command "Changecolor to red and draw a triangle".

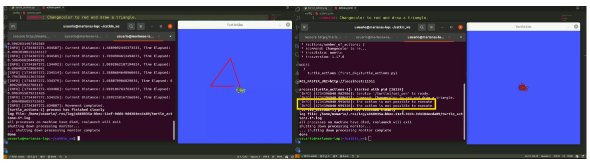


Figure 8: Comparison of results for the command "Changecolor to red and draw a triangle". The left side shows the output of the RBM, while the right side presents the output of the LLM.

In the case shown in Figure 7, the LLM output generated the term "Changecolor" with a capital letter, whereas the robot code only accepts non-capitalized words. This issue highlights a limitation of the LLM and its few-shot prompt for fine-tuning, as the model requires further extensive tuning to handle such cases properly. In contrast, the RBM addresses this problem by lowercasing the words during the pre-processing stage.

In terms of parsing accuracy, we note that on our testing subset, our model achieved a statistical accuracy of 15 successful trials over 15 attempted trials, which above the success rate reached by the LLM method (6 successful trials out of 15 attempted trials). However, these results have to be nuanced.

First of all, our implementation struggles to independently identify multiple motions within a single command when they share the same verb. For example, a command like "move left then right" consists of two distinct actions. Yet, because the sentence includes only one verb ("move"), the system processes it as a single action instead of two separate motions. This underscores a challenge in fully capturing the semantic structure of commands where multiple directions or parameters are implied but not explicitly tied to unique verbs. This is typically a task that is done better by LLMs.

Moreover, our code is currently constrained by the predefined set of words incorporated into the grammar, which does not update dynamically. Although we initially implemented a mechanism to allow the grammar to expand dynamically, this approach introduced limitations (such as the automatic parsing of non-grammatical sentences) that prevented its practical application.

5 Discussion and conclusion

5.1 Verification of the Initial Hypothesis

The hypothesis that rule-based methods could achieve competing performance with large language models was partially verified. In specific contexts where commands are straightforward and the domain is tightly constrained, rule-based methods matched or even exceeded the performance of LLMs.

5.2 Conclusions from the Experiments

Our experiments provide valuable insights into the comparative performance of rule-based methods and large language models for generating robot-compatible code from natural language commands. The results indicate that while LLMs generally offer broader comprehension and adaptability due to their extensive training datasets, rule-based methods have shown competitive performance in scenarios with well-defined command structures and limited domains.

6 Limitations

While the results are promising, the study has limitations. For example, the experiments were conducted in a simulated environment that may not fully capture the dynamic and unpredictable nature of real-world robotic scenarios. Furthermore, the current implementation may not scale efficiently with significantly larger sets of grammar rules or highly ambiguous command sets. As much as our grammar shows adaptability with many syntactic constructions, these results should still be expanded to be more reflective of the sample space of possible sentences to be written in English.

7 Future Work

Future research could focus on optimizing the algorithm for dynamically extending CFG grammars to enhance scalability and efficiency. We have attempted to create a syntax that would grow dynamically with new sentences. That is, have a syntax that would add to its grammar new syntactic sequences. However, this quickly ended up counterproductive, as our grammar would parse non-grammatical sentences.

Additionally, testing the system in more varied and unpredictable environments would help in evaluating its robustness and practical applicability. Extending the system to interpret commands in multiple languages could also vastly increase its utility in international settings.

Ethics Statement

This work involves the development and testing of systems that interpret natural language commands and convert them into executable robot actions. While our primary focus is on testing and evaluating techniques for robotic control, we are mindful of the broader ethical implications of deploying such systems in real-world environments.

We acknowledge the importance of ensuring that robots are used safely and responsibly. Our methods prioritize the generation of accurate and deterministic outputs to minimize the risks of errors in robot actions. However, as with any robotic system, there are potential concerns regarding the reliability and robustness of these systems when interacting with humans or operating in dynamic environments.

Furthermore, we recognize that the use of NLP-based systems in human-robot interaction raises issues related to privacy, data security, and bias. While this work does not directly handle personal data, we are committed to adhering to privacy best practices when collecting or processing any user input in future implementations.

8 Statement of contributions

All authors contributed equally to the development of the code, dataset generation, and report writing. The submission includes the paper, code, outputs from both models for given prompts, GIFs of robot behavior, and a detailed breakdown in the read-me file.

References

Miguel Á. González-Santamarta, Francisco J. Rodríguez-Lera, Ángel Manuel Guerrero-Higueras, and Vicente Matellán-Olivera. 2023. Integration of large language models within cognitive architectures for planning and reasoning in autonomous robots. *arXiv preprint arXiv:2309.14945*. Accessed on March 23, 2024.

Fabien Joublin et al. 2024. Copal: Corrective planning of robot actions with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8664–8670, Yokohama, Japan.

Thomas Kollar, Stephanie Tellex, Deb Roy, and Nicholas Roy. 2010. Toward understanding natural language directions. In *HRI*.

Anis Koubaa. 2023. Rosgpt: Next-generation human-robot interaction with chatgpt and ros.

Ji Liang, Weidi Huang, Feng Xia, Peng Xu, Kyle Hausman, Benjamin Ichter, Peter Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. *arXiv*.

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. 2020. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*. First published as a Review in Advance on January 31, 2020.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. *MITPROJECTMAC*.

YAML Organization. 2021. The official yaml web site. Accessed: 2024-12-17.