Human Robot Interaction project

# Introduction

Recently the advances in Artificial Intelligence (AI) have achieved increasingly notable gains in all AI related fields, with particular attention to Natural Language Programming and Large Language Models (LLM). The availability of increased computing power and larger, more comprehensive datasets, have spearheaded impressive results in LLMs. Among the many things LLMs can assist with, an interesting advantage is their ability to generate code. Due to syntax and logic, writing computer code was considered a difficult task, yet LLMs seem to have an easy time generating code. Large Language Models are known to sometimes hallucinate, generating content that is not factual. However, considering the professional nature in which the LLMs provide answers, and the confidence in generating content, it can be challenging understanding how helpful LLMs can be when generating code.

This project investigates how effective LLMs are in generating computer code for an abstract robot class, by using one-shot techniques. This was done by generating a “system instruction” document that contained a detailed description of the environment, requirements and expectations of the LLM. The “system instruction” would also specify with examples the type of input that is expected, and the type of output the LLM would need to generate.

The project used a set of 20 inputs, which were asked to three different LLMs: ChatGPT 3.5, Gemini 1.5 Pro and Claude 3.5 Sonnet.

# Environment

For a project of this nature to be specific I employed a closed environment of an abstract robot class, specifying the abilities of the robot instances and methods in which a robot instance could manifest those. Instances of the robot could move on the x, y and z axis in 3D space and use a gripper to hold objects – which are instances of an abstract object class. In addition, a method to check collision was executed before the robot instance would move to a destination to check for logic errors, such as placing an object in the same coordinates where another object would be.

## Implementing collision

Implementing collision. One of the problems that I am having is implementing a way to check if running the generated method will create collision between robot and object as it would arrive at object destination before it would open the gripper. Therefore I would like to implement a Boolean gripperIsOpen which will be used in the methods. If the robot arrives at the coordinates of the object while gripper is closed, the method fails. One of the ways I could implement this is by making all of the methods a Boolean, returning false or true. Another way of implementing this is by making a condition where an error is raised if the condition ( gripperIsOpen == false and robot coordinates == object coordinates) is true.

After considering the various ways in which to implement this functionality, I decided to implement a checkCollision() method, which will perform a check before moving the robot and when closing the gripper. If a collision is detected, it will throw an exemption. This implementation allows for higher quality tests to be performed, asserting both the object and the robot’s position.

# Prompt engineering

A significant factor in obtaining relevant results when working with LLMs is prompt engineering. The quality of the prompts almost always reflects the quality of the generated outputs.

After investigating different ways of using LLMs to assist with robotics, it was decided the project would instruct the LLMs to generate methods which to interact and manipulate instances of the object class. The project will also use a test class to perform tests on the functions generated by the LLMs.

# Related work

The recent increase of AI based solutions to robotics have shown significant increase in using LLMs to overcome challenges and provide solutions for hand exoskeletons control (Chen et al, 2024), Human-Robot Collaboration (Liu et al, 2024) or personalised robot assistants (Wu et al, 2023).

Various other studies have attempted to investigate the practical usability of using LLMs to generating code for specific tasks(Singh, 2023), and have used zero-shot techniques (Liang et al, 2024) to instruct LLMs into providing code for robotic tasks.

# Testing methodology

A series of 20 inputs of varying difficulty was employed, each prompting the LLM to generate code for instances of the abstract robot class. The prompts were designed to challenge the LLMs in producing code that would be used for movement and/or interaction with objects. Some prompts would be a simple task, requiring the robot to move a few units on the X axis, others would have increased complexity requiring the robot to move to a position, pickup an object and place it to a different position in the cartesian space, with the most challenging prompts asking the LLM to generate code that would manipulate multiple objects across many different steps, such as picking two different objects from different locations and stacking them on top of each other at specific coordinates.

Each of the models studied was presented with the same inputs in the same order, with the purpose to investigate how well the LLMs could conceptualise and formulate coding problems when provided with identical instructions.

Testing was done by implementing a test class containing tests for all the methods the LLM generated. The same set of tests was used for all three LLMs evaluated in this study, thus ensuring a standardised approach. After a method was generated by the LLM, such as a method that moved the robot forward 2 centimetres, tests were performed to evaluate the expected position of the robot instance with the actual position. The same tests would also evaluate the object’s position, thus allowing for a standardised platform to check if a generated coding method was successful or not.

# Results

Different methods have shown very different results, with more recent LLMs performing much better compared with older models. After testing the generated code for the 20 input prompts, the results are:

* Gemini 1.5 Pro
  + Passed - 12
  + Failed - 8
* ChatGPT 3.5
  + Passed - 4
  + Failed – 16
* Claude 3.5 Sonnet
  + Passed – 19
  + Failed – 1

Analysing the results showed a clear indicating that recent models perform better than older ones, with ChatGPT 3.5 failing the majority of the tests because it did not follow the instructions clearly specified in the “system instruction” one-shot input.

While performing better, Gemini 1.5 Pro generated many functions which caused collision between objects, this happening even after there were clear instructions on how to manipulate objects with the gripper, the object’s dimensions and the positions in cartesian space.

Claude 3.5 Sonnet had the most promising results, with the only error given to a method that employed manipulating objects at different heights.

## Future analysis

Importantly, this project analysed the effectiveness of three LLMs and with increased research and funding allocated to Large Language Models, future models will likely have improved results in human-robot interactions. Secondly, the project used Java as the programming language of choice. Considering LLMs are trained on existing code, there is a clear connection between the popularity of a programming language (and the abundance of code in that particular language available to use for training) and the performance of LLMs. This suggests that popular languages such as Java or Python will have increased chances of obtaining well working code from an LLM compared with less popular languages like Assembly.

# Conclusion

In conclusion, large language models can be used to generate code used for human-robot interaction.

References

Liang, Y. et al (2024), “Taxonomy-Guided Zero-Shot Recommendations with LLMs”, online, available at: https://arxiv.org/pdf/2406.14043 (Accessed: 23.07.2024)

Singh, I. et al (2023), “ProgPrompt: Generating Situated Robot Task Plans using Large Language Models”, online, Available at: https://progprompt.github.io/ (Accessed: 23.07.2024)

Chen, W. et al (2024), “LLM-Enabled Incremental Learning Framework for Hand Exoskeleton Control”, online Available at: https://ieeexplore.ieee.org/document/10489910 (Accessed: 23.07.2024)