POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Master's Degree Thesis

User-on-the-Loop for Human-Robot Interaction: LLM Code Generation and Validation in Manipulation Tasks

Supervisors

Prof. Alessandro RIZZO

PhD. Pangcheng David CEN CHENG

Candidate

Francesco DELLA CORTE

March 2026

Abstract

In many robotic applications, robots operate in a shared workspace where users may modify or supervise tasks. In Small and Medium Enterprises, the adoption of automation is still limited by the necessary robotic skills required for programming manipulators. An idea is to give focus on Large Language Models (LLMs), even if the lack of trust from the human side represents a significant obstacle. This thesis presents a "user on the loop" approach in which an LLM interprets a high-level robotic assistant. From a natural language request, the model generates a pseudocode that is validated through simulation, refined through feedback iterations, and executed on the real hardware by the user.

While natural language provides an intuitive and flexible interface, it is often ambiguous and inaccurate. In order to increase robustness reducing the risk of hallucinations, which are false and misleading information that could disorient the user, the LLM is constrained. The idea is to compose plans only through a high-level library of descriptive and task-relevant APIs, while low-level information is hidden in a black box. Vision is provided by an RGB-D camera and ArUco fiducial markers based on OpenCV. An important choice is to introduce a layer of abstraction in which the object is recognized by its ID, but it is mapped to a name and to an accurate description through the use of four YAML files. The estimation of the pose translates perception into executable actions.

For the experimental part, the proposed system employs an xArm6 from UFactory with ROS2 Jazzy, MoveIt 2, RViz, and an Intel RealSense D435i for vision. Experimental results refer to nine tasks of increasing difficulty based on: pick and place, ordering, stacking, and delivering objects to a human. The motion planner is chosen in simulation from the Open Motion Planning Library (OMPL), and the RRT-Connect is selected as the one with the best overall average planning time, with 0% of failure rate. The performance of the LLM is evaluated by comparing GPT-4.1, GPT-5, and GPT-5.2 under zero-shot, one-shot, and few-shot prompting strategies. The results show that when examples are provided to prompts is determinant; indeed, one-shot and few-shot are the best strategies. GPT-5.2 achieves the highest immediate successes, minimizing human effort.

Overall, the thesis demonstrates that an LLM system for code generation and planning, through a "user on the loop" approach, allows non-technical users to control a collaborative robot to perform manipulation tasks in a safe and repeatable manner. Future work focuses on overcoming ArUco markers dependency, on conducting a study on a larger sample of non-technical users, and on expanding the set of tasks and API library.

Table of Contents

List of Tables

List of Figures

Acronyms

HRC
Human-Robot Collaboration

HRI
Human-Robot Interaction

SMEs
Small and Medium Enterprises

LLM
Large Language Model

AI
Artificial Intelligence

NLU
Natural Language Understanding

NLP
Natural Language Processing

SSL
Self-Supervised Learning

ZSL
Zero-Shot Learning

RLHF
Reinforcement Learning from Human Feedback

ROS

Robot Operating System

RGB

Red Green Blue

RGB-D

Red Green Blue + Depth

ZSP

Zero-Shot Prompting

OSP

One-Shot Prompting

FSP

Few-Shot Prompting

CoT

Chain of Thought

OMPL

Open Motion Planning Library

TCP

Tool Center Point

IDL

Interface Definition Language

IK

Inverse Kinematics

KDL

Kinematics and Dynamics Library

FCL

Flexible Collision Library

GUI

Graphic User Interface

URDF

Unified Robot Description Format

SRDF

Semantic Robot Description Format

CHOMP

Covariant Hamiltonian Optimization for Motion Planning

RRT

Rapidly-Exploring Random Trees

Chapter 1

Introduction

Robots were always used as tools to perform tasks that require movement and navigation in environments that are hazardous for people. However, today they are becoming more social and more integrated into everyday life. Human-Robot Collaboration (HRC), or Human-Robot Interaction (HRI), focuses on how robots can be designed in order to be more interactive with humans and how this collaboration can improve the performance of the task while reducing the workload on the human operator. HRC refers to a context in which the human operator and the robot work together giving shape to a dynamic system to perform a task. In order to achieve an optimal level of mutual adaptation, robots are developed with specific features and are named Collaborative Robots (Cobots). Cobots represent a tradeoff in increasing productivity while maintaining high quality standards, and increasing the working conditions of the human operators involved in industrial activities [1][2].

Modern automation is very dependent on robotic systems; indeed, industrial growth is due mainly to automation technologies. Robots offer versatility, precision and efficiency in task execution, covering different fields, from routine activities to demanding ones. They play an important role in manufacturing, logistics, and healthcare. By focusing on Industry 4.0, represented by smart factories, robots are used in production centers in order to optimize productivity and performance [3].

Small and Medium Enterprises (SMEs) play an important role in the global economy because they represent the 99% of all enterprises and generate the 60% of overall employment [3]. SMEs are late in adopting automation with respect to larger organizations. Just by considering the manufacturing sector, there is an average of about 246 robots per 10.000 employees, while SMEs have only about 6. This difference is mainly due to high costs and limited technical expertise. Coordination between human operators and robots is really important; there is a need for an improvement in HRI systems. These systems are designed to obtain a crucial collaboration between humans and robots, enabling a smooth execution of

tasks [3]. Trust is a central challenge in HRC. Although robots are increasingly present in several sectors, the lack of trust from the human side continues to be a significant obstacle.

Nowadays, the development in the robotics practice still remains dependent by engineering expertise. Indeed, the existing pipelines depend on an expert and skilled engineer that stays "in the loop" and writes code needed to refine and improve the system. For this reason, the use of robots in real world tasks is limited by the availability of expert programmers [4].

An idea is to use a Large Language Model (LLM), like ChatGPT, to improve the collaboration using natural language [5]. Humans possess special capabilities of manipulation thanks to the combination of tactile, visual and sensory feedback. LLMs have opened a way in order to improve robot interaction and autonomy [6]. Despite robots became more capable in different environments, the main obstacle continues to be the complexity of programming them. The idea is to combine LLMs with ROS (Robot Operating System). This leads to a system in which any user, regardless of being a technical expert, can describe an objective in a simpler language and the robot helps to achieve the task [7].

Large Language Models like ChatGPT have demonstrated different abilities in text generation, machine translation and coding. These models are more and more being applied in the field of robotics, in particular in planning tasks in order to avoid the need of additional training. However, there is the need to pay attention to the direct use of the generated code because it can lead to inconsistencies and unstable behavior [8]. Code generation by LLMs could suffer of problems due to its nondeterministic behavior. Several experiments showed that these models could produce an inconsistent code with syntax errors, missing libraries and failures of execution. This happens because LLMs are "forgetful" and ignore important information from the input of the system. An idea is to impose constraints in the prompts and to pay attention to numerical context in order to reduce errors in the code that is generated [9].

The main focus of this work of thesis is to improve the behavior of collaborative robots by implementing a practical case study in which an xArm6 of UFactory performs operations of pick-and-place, so the primary objective of this thesis are:

- To use ROS2 for robot integration, MoveIt2 for the motion planning and for the simulation part, with RViz as visualization tool;
- To obtain poses and orientation directly thanks to the support of an Intel RealSense D435i camera and the object detection will be achieved by using ArUco markers in OpenCV;
- To compare the performance of GPT-4.1, GPT-5 and GPT-5.2, used as a robotic assistant in generating Python-like pseudocode, evaluating three

prompting strategies that are zero-shot, one-shot and few-shot, within a user on the loop approach. This aims to facilitate and improve Human–Robot Collaboration by sending text-based commands provided by non-expert users and evaluating the answers on the real hardware;

Despite important progress regarding industry 4.0, automation and LLMs, several key questions are still opened and one of the objective of this work of thesis is trying to answer. The main questions are:

- Can non-technical users interact with coding for robotic applications?
- LLM-generated plans and code are always safe and consistent?
- What mechanisms can be implemented to verify and constrain plans produced by an LLM so that they could be executed on the xArm6?
- Which prompting strategies can be adopted to avoid the ambiguity of the output of the LLM and to guarantee the repeatability of the robot's behavior?
- How should be structured a ROS2 architecture, regarding nodes, topics, services and actions, to integrate in the best way ChatGPT, vision, xArm6 and MoveIt2?

Thesis Structure

After the introduction, the work of thesis is organized as follows:

- Chapter 2: this chapter introduces the main concepts found and studied in literature including large language models, prompt engineering, motion planning, pick and place robots and computer vision;
- Chapter 3: this chapter introduces the elements of design of the system that were used in the research conducted. The focus was on the robot used that is an xArm6, the software to build robotics applications that is ROS 2 (Jazzy), the platform for the simulation that is MoveIt2 and the transition from an "Engineer in the Loop" to a "User on the Loop" approach;
- Chapter 4: this chapter describes the implementation of the adopted "user on the loop" approach. In particular it highlights its four fundamental steps highlighting their role in the operational flow. Furthermore, a section is dedicated to object pose estimation, which is necessary for the execution of the tasks;

- Chapter 5: this chapter describes the obtained experimental results through the evaluation of the impact of the LLM used and of the prompting strategy adopted. The analysis regards the choice of the planner and it compares the three different models, the three prompting strategies and it analyzes the successes reached;
- Chapter 6: this chapter discusses the final conclusions and highlights possible future works suggesting directions like using different techniques of object detection, expand the sample of users and extend the set of tasks and of APIs;

Chapter 2

Literature Review

This chapter provides a literature review of the main concepts and approaches on which the thesis work is built. The goal is to introduce and analyze the existing research solutions, focusing in particular on making more intuitive Human–Robot Interaction and more autonomous robot behavior. The chapter starts with the introduction of LLMs, both in general and in the robotics field. Then the focus is given to the prompt design practices that are typically used to make the outputs of the LLMs more reliable. The next step is the introduction of the main approach used to conduct the experiments. It then covers the components and the types of a pick and place robot and the motion planning methods, before concluding with an overview of computer vision that enables recognition. Overall, this chapter establishes the background used in the rest of the thesis and provides motivations for the choices made.

2.1 Large Language Models (LLMs)

LLMs are a class of models trained on huge amounts of data, which give them the ability to understand and to generate natural language. Indeed, they are able to carry out a wide spectrum of tasks. LLMs entered the public domain thanks to their central role in the field of generative AI. Lots of organizations are going to adopt AI in a lot of business functions and use cases. Many companies have spent and continue to spend years in order to integrate LLMs at different levels to improve their capabilities in Natural Language Processing (NLP)[1] and in Natural Language

[1]"Natural language processing (NLP) is a subfield of computer science and artificial intelligence (AI) that uses machine learning to enable computers to understand and communicate with human language" [10].

Understanding (NLU)[2]. This evolution obtained a big progress with machine learning models, algorithms, neural networks, and transformer architectures that provide the basis for these AI systems. LLMs are trained on vast datasets to provide necessary basic functionalities in order to support several use cases and applications, and in order to solve a huge variety of tasks. The Figure 2.1 shows how LLMs can act as a central brain that connects different inputs, like text, voice, or images, into a variety of outputs like Q&A, sentiment analysis, image captioning, and object recognition.

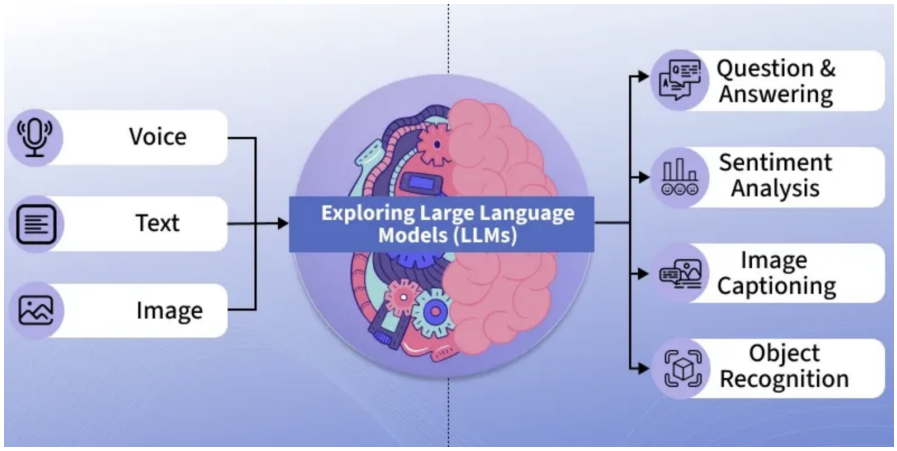

Figure 2.1: LLM General Scheme [12]

LLMs represent an important turning point in NLP and AI in general, and are accessible to everyone through interfaces like OpenAI's ChatGPT, Meta's Llama models, Google's bidirectional encoder representations (BERT/RoBERTa) derived from transformers and PaLM. In general, LLMs are designed in order to understand and to produce text in a human manner thanks to the huge amount of datasets used for training them. They can deduce from context, produce coherent answers, translate between languages, and assist with tasks like code generation. These capabilities arise thanks to the billions of parameters that are present in the models. Following its evolution, LLMs are going to reshape the way in which people interact with technology and access information [13].

Technically, LLMs work using deep learning techniques and a huge amount of text data. They are typically based on a transformer architecture. LLMs consists on different neural network layers with parameters that can be tuned during the training. During training, these models learn to predict the following token in a

[2]"Natural language understanding (NLU) is a subset of artificial intelligence (AI) that uses semantic and syntactic analysis to enable computers to understand human-language inputs" [11].

sequence based on the preceding one. The model assigns a probability score to the word occurrences. Those tokens are smaller text units that are converted into numerical representations of this context. Generally, the training of the LLMs is done thanks to Self-Supervised Learning (SSL)[3] and Zero-Shot Learning (ZSL)[4]. After the training part, the LLMs generate text by predicting the successive tokens given the input and the knowledge acquired. Model performance can be improved by using prompt engineering, prompt-tuning, fine-tuning, and other techniques like reinforcement learning from human feedback (RLHF), in order to avoid errors known as "hallucinations" that are factual errors that can arise from training on a large amount of data [13].

LLMs like OpenAI's ChatGPT have transformed AI and the way humans work with robots. An important capability of these models is their fast adaptation to new ways of communication. They can perform tasks they were never trained for ("Zero-Shot Learning") and they can rapidly learn new tasks after they saw only few examples ("Few-Shot Learning"[5]). This ability is really important to improve the performance and the efficiency of robots. A fundamental technique to achieve this solution is prompt engineering: a procedure that turns natural language instructions into executable commands for a robot. The LLM interprets the user's input and generates actions directed to the robot's context. But the real problem is effectively crafting prompts that produce a correct behavior because differences in the use of the words, for example, can lead to wrong outputs or confusing outcomes [17].

2.1.1 Benefits of LLMs in the Robotic Field

As explained in [6], the different benefits that LLMs will bring to robots are:

1. Natural language interaction: LLMs allow robots to communicate coherently with users in an intuitive way, removing the need for heavy pre-programmed commands.

2. Task execution: robots can understand and execute complex instructions

[3]"Self-supervised learning (SSL) is a machine learning technique that uses unsupervised learning for tasks that conventionally require supervised learning. Rather than relying on labeled datasets for supervisory signals, self-supervised models generate implicit labels from unstructured data" [14].

[4]"Zero-shot learning (ZSL) is a machine learning scenario in which an AI model is trained to recognize and categorize objects or concepts without having seen any examples of those categories or concepts beforehand"[15].

[5]"Few-shot learning is a machine learning framework in which an AI model learns to make accurate predictions by training on a very small number of labeled examples"[16].

divided into smaller steps, increasing versatility across different tasks.

3. Knowledge acquisition and reasoning: LLMs can extract information from vast knowledge bases, providing contextual understanding and situational reasoning.

4. Flexibility and adaptability: LLMs enable robots to adapt to dynamic environments by interpreting different circumstances.

5. Learning and improvement: feedback loops with users allow LLMs to improve robot behavior and performance in order to adapt to user preferences.

6. Multimodal interaction: robots, thanks to the support of LLMs, can combine text, voice, visual, and sensor inputs for a better understanding.

7. Education and entertainment: robots can act as tutors for children's education or also in the entertainment industry.

8. Emotional interaction: LLMs enable robots to simulate emotional responses, improving human-robot collaboration.

9. Collaboration and Cooperation: LLMs allow robots to work as cooperative partners for humans in complex tasks like problem solving or the formulation of plans.

10. Innovation and exploration: With the ability to interpret new information, robots can support research or creative avenues.

2.1.2 Applications of LLMs in the Robotic Field

As explained in [6], the applications of LLMs in robotics across various domains are:

1. Autonomous navigation and path planning: LLMs provide robots with capabilities in order to interpret commands and navigate through environments, and adapt to obstacles;

2. Speech interaction and NLP: LLMs give robots the ability to understand natural language commands, hold conversations, execute particular tasks, and provide personalized experiences;

3. Visual perception and object recognition: by combining visual data with language understanding, robots can identify objects, interpret several scenes, and recognize changes in the environment;

4. Human-robot collaboration and social robots: robots thanks to LLMs can assist humans by understanding context, predicting needs, and adapting their behavior, making them a support in fields like healthcare, education, and entertainment;

5. Humanoid robots and emotional expression: LLMs allow humanoid robots to simulate empathy and emotion through natural language generation and recognition of emotions, providing emotional support and companionship;

6. Industrial automation and robot control: robots can handle industrial tasks, adapting processes by learning and analyzing different data, improving efficiency in manufacturing and logistics;

7. Healthcare and rehabilitation robots: LLMs can be applied in the medical robotics field to support patients in diagnosis, physical assistance and treatment, helping in disease detection, surgical planning, and therapies for rehabilitation;

8. Environmental monitoring and exploration: robots equipped with LLMs can analyze environmental data, detect disasters and explore unknown territories;

9. Agriculture and farm mechanization: robots, thanks to LLMs, can assist farmers by monitoring plant health and automating intensive tasks. They can help farmers in order to improve agricultural productivity and sustainability;

10. Education and learning assistance: LLMs help robots to act as personalized tutors, adapting lessons to students, answering questions and offering learning experiences;

Figure 2.2 helps to understand that by combining robotic systems with an LLM, robots can become embodied intelligent agents, making the interaction with humans more flexible.

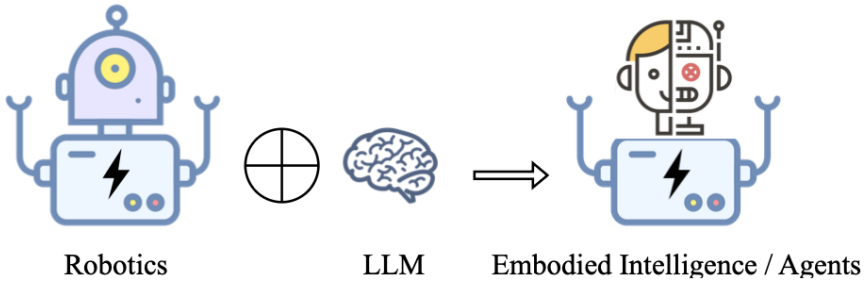

Figure 2.2: Robotics based on LLMs [6]

2.1.3 Non-Determinism of ChatGPT in Code Generation

Due to their use of predictive probabilities of a word or token based on a sample of words in the context, large language models are non deterministic for nature. The randomness of the LLM's predictions comes from the sampling method used to generate text, for example, the top-k sampling or the nucleus sampling. Due to this nature of randomness, it is possible that two or more requests made to an LLM with identical instructions and prompt will produce completely different answers. This is a critical aspect of considering how to utilize them effectively. In particular, unreliable and inconsistent code snippets can significantly interfere with the software development process, especially in applications that are safety-critical, where both consistency and reliability are really important. Moreover, the repeated presence of completely different code suggestions by an LLM can be disadvantageous for the confidence of software developers in the utility of the LLM.

The threat of non-determinism, for example, in ChatGPT, although it is also present in many of the tasks that an LLM may perform, including question answering and text summarization, it is more harmful in code generation for certain tasks. In particular, the inconsistency, especially semantic inconsistency, which exists between the actual output generated by ChatGPT and what was intended to be generated, is indicative of an error within the generated code. Therefore, it is essential to acknowledge the limitations and the associated potential risks of utilizing code generated by ChatGPT, as the variability of the generated code creates questions regarding the quality and the validity of the assessment of its ability to generate code [18].

2.2 Prompt Engineering

The Figure 2.3 shows the basic interaction with an LLM in which a user provides a prompt as input and the LLM returns a response as output.

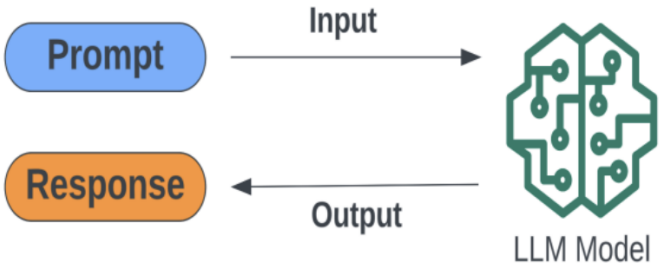

Figure 2.3: LLM's prompt and response [19]

"Prompt engineering is the practice of writing clear, purposeful inputs that guide AI models to deliver accurate and context-aware outputs" [20].
In the field of Large Language Models, prompting has become, over time, more and more an iterative approach: evaluating outputs, improving language, and fitting the structure until the obtained output matches the desired output. Considering prompt as a kind of "design tool" helps teams to model the LLM behaviors towards accurate, coherent, and aligned with human expectations results across different applications. Indeed, every interaction with an LLM involves a prompt. The model does not interpret the text as if it were a human, but it examines linguistic structures and, based on the information provided, tries to predict the best completion of a sentence. So, the way in which a prompt is structured and formulated directly affects how the model responds. If the input is short or ambiguous, the output often could be too general and confusing. However, when prompts include details, contexts, clear requirements and, sometimes, examples, the output will be more accurate and relevant.

The way a prompt is made has a real impact on how an LLM interprets a request and how it answers. Common prompt styles are:

- Instruction prompts: these are specific instructions of what the model has to do, and they are clear and operative, especially in coding, summarizing, and transforming;

- Example-based prompts: these consist of providing one or more examples to show one or different good outputs. This can help the LLM to better replicate specific patterns and to remain consistent in logic and format;

- Conversational prompts: these prompts are written to sound like a dialogue, and they promote a back-and-forth interaction. This style of prompting is really useful in chat interfaces or in documentation assistants, but also in situations in which there is a need for explanation of a piece of code;

- System prompts: these define the overall rules, constraints or personalities that guide the LLM in the interaction. For instance, this prompt could define the role of the model, define some style and safety norms or establish expectations of how all the responses should be formatted;

In order to improve performance, there exist other structured prompting strategies and some of them are:

- Zero-Shot Prompting (ZSP): this strategy gives the model a task without providing an example to follow. The model has to rely on its general knowledge to give an answer, but, in general, this strategy is enough for simple and well-defined tasks;

- One-Shot Prompting (OSP): this is a strategy where a model is given only an example or prompt to execute a task. This method provides an important advantage because it allows models to generalize from a single prompt in cases in which the input space is limited, for example;

- Few-Shot Prompting (FSP): this strategy is helpful when the structure and consistency of the output are important because it works by providing a few examples to guide the model to reach the desired output;

- Chain-of-Thought (CoT) Prompting: with this strategy, the model is asked to reason step by step in order to improve accuracy in tasks that require logic, such as debugging, analysis, or planning;

In the end, the choice of the "best" prompt depends on the objectives because sometimes a structured instruction could satisfy all the needs, in other cases, there is a need of examples or step-by-step reasoning to reach the goal [20].

2.2.1 Elements of a Prompt

The process of constructing a good prompt is a process that relies on different important elements: instruction, context, input data and output indicator. These elements, all together, help the large language model to produce responses that are accurate and relevant. Their importance becomes relevant in contexts like the field of programming or software engineering. A description of the elements is:

1. Instruction: this element represents the command the model is expected to perform. An example in the case of programming could be: "Write a Python function";

2. Context: this element adds information like the background that helps the model to better understand the task. In the case of programming, an example could be to specify any requirements of a certain function;

3. Input Data: this element represents the concrete information that the model needs in order to execute the task. An example could be a necessary mathematical formula to achieve a certain objective;

4. Output Indicator: this element specifies how the response should be structured or presented;

From the combination of these elements, prompts can be crafted in order to guide large language models to produce relevant and accurate outputs [19].

2.3 User on the Loop in Code Generation

As explained in [4], the use of LLMs for robotic control introduces a number of complex challenges. These include the need to describe in an accurate and clear way the task through the prompt, to identify and select the appropriate functions and APIs, to guide the structure of the outputs given by the model through the correct constraints and arguments, and so on. As also presented in Figure 2.4, the pipeline used to enable the effective use of ChatGPT in robotic applications is:

1. A high-level library of robot functions is defined. This library could be adapted to a specific robot form factor or operational scenario and should be strictly related to concrete implementations on the robotic platform, while remaining descriptive enough for the language model to interpret correctly;

2. A prompt for ChatGPT is constructed and it has to highlight the desired objective and has to specify the set of allowable high-level functions from the library. This prompt may also include other constraints or instructions regarding how the model should organize and present the output;

3. A user remains on the loop to assess the code generated by ChatGPT through direct inspection or simulation, and to provide feedback on different aspects like correctness, safety, and quality;

4. The code can be directly tested on the physical robot, once the generated implementations have been refined through iterative feedback of the human operator;

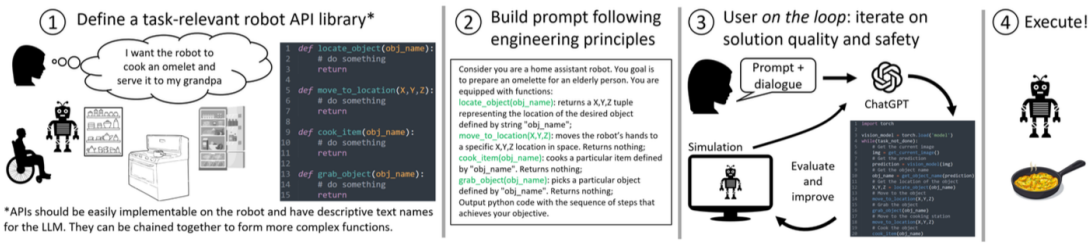

Figure 2.4: "Robotics pipeline employing ChatGPT with the user on the loop to evaluate the output's quality and safety" [4]

The study focused on the evaluation on the ability of ChatGPT to manage more complex tasks while keeping the user on the loop who provides feedback in natural language. This interaction was found to support complex strategies, such as curriculum learning, in which the system is taught simpler skills that can be

combined to achieve more complex objectives. Furthermore, the study showed that ChatGPT can interpret non-technical textual feedback about the quality or behavior of the generated code and translate it into appropriate low-level code changes. This capability could help even users without strong technical backgrounds to interact with the system in an efficient way, guiding its behavior through intuitive and natural language feedback.

2.4 Pick and Place Robots

In Figure 2.5, there is an example of Pick and Place robots, which are a type of industrial robots that are developed to move objects from one position to another with high efficiency. The first robots were introduced in the 1960s and, now, have evolved into a fundamental component in modern warehouses because they can handle various tasks, from the simplest, highly repetitive actions to the more complex ones. Thanks to their speed, precision and reliability, pick and place robots are now of paramount importance in sectors such as manufacturing, packaging and logistics. Because of their autonomous ability to perform potentially dangerous activities for human workers, like lifting heavy boxes, they made an important contribution in reducing the risk of injuries and improving workplace safety [21].

Figure 2.5: Example of Pick and Place Robot [22]

Although the basic idea behind pick and place robots appears to be a very simple system, the systems themselves are extremely sophisticated. There are a lot of interconnected components that operate in a few seconds to locate, pick up, and relocate objects with high accuracy. Through the use of sensors and vision systems,

the pick and place robot is able to calculate an object's position, dimensions, and orientation. This set of information allows the robot to determine the optimal movement path, through motion planning algorithms, ensuring efficiency while avoiding potential obstacles. Thanks to the presence of the end-effectors, the robot is able to grasp the object, move it to the desired location, and position it where needed with precision [21].

2.4.1 Main Components of a Pick and Place Robot

As presented in [21], the software part is a fundamental element in pick and place robots, but alone it does not have a high value, so without capable hardware and perfect coordination between the two. The real effectiveness of these systems arrives in the moment in which there is an integration of the physical components with the intelligent software. The main components of a pick and place robot are:

1. Robotic arm: this component is constructed to perform fast and efficient movements and its main focus is to reduce the cycle duration, which is an indicator that estimates the total duration needed to start a task, complete it and return to the initial position. The robotics arm represents only one part of the overall solution. The choice of the right arm depends on several factors like available space, required capabilities, and workload volume. Once the mechanical design is ended, the robotic arm is programmed and integrated with sensors, controllers and end-effectors, in order to obtain efficiency and velocity in executing tasks;

2. End-effectors: these components represent the interface of the robot with the physical world, allowing it to directly interact with objects. Their design varies according to the objects that are going to be handled. Common solutions include:

    - Vacuum grippers: these types of grippers are based on suction, and they are used in cases of smooth surfaces, for example, because they are less effective in the case of open or porous items;
    - Mechanical grippers: these grippers can adapt to different shapes and sizes, thanks to their finger-like grips, even if they require sufficient space around the object that is going to be manipulated;
    - Magnetic and adhesive grippers: these grippers are typically used for heavy metal components in industrial production environments;

3. Sensors: they are really important for the pick and place robots in order to guarantee maximum precision. Several types of sensors include:

- Vision sensors: these types of sensors are equipped with 2D or 3D cameras, they allow the robot to identify objects and determine what and how to pick;
- Force sensors: these sensors are usually implemented in the end-effector and they regulate the force of the grip in order to avoid damaging objects that could be fragile. Moreover, they can also measure weight to confirm that the correct number of objects is being handled;
- Proximity sensors: these types of sensors are important for safety, because its aim is to detect obstacles or human operators in the workspace to prevent collisions;

4. Controllers and movement control: controllers represent the central core of the pick and place robots because they ensure that hardware and software operate as a unique system in order to execute pick and place tasks. They receive input data from sensors and determine the next moves of the robot.

2.4.2 Types of Pick and Place Robots

As can be seen in [23] pick and place robots exist in a variety of forms, each designed to satisfy certain needs and environments. The main types of pick and place robots are:

- Robotic arm: these types of robots are the most used solution for pick and place operations. A 5-axis arm is typically sufficient for standard tasks where objects are picked and placed in a single plane. On the other hand, 6-axis robotic arms are used for more advanced applications, as they allow items to be rotated or reoriented before being placed in their final position;

- Cartesian: these types of robots that are similar in capability to six-axis robotic arms, operate in multiple planes, moving in the three linear and orthogonal axes X, Y, and Z (Cartesian coordinates). They are built with linear actuators and driven by mechanisms such as belts, balls, or lead screws. Cartesian robots often have a more accurate positioning with respect to 6-axis robotic arms;

- Delta: these robots are commonly used in contexts in which there is a need to pick objects in groups and place them in assembly patterns or containers. Equipped with complex vision systems, they can recognize objects with different sizes, shapes, and colors. Most delta robots are composed of three arms that operate on four axis and they have heavy motors attached to a frame, with lighter arms connected to linking rods with ball joints at either end of each arm to allow movement;

- Fast pick: these types of robots are designed for medium to high-performance environments that involve high-speed product handling. By fully automating the picking process, they reduce the need for manual labor and allow workers to focus on other tasks;

- Collaborative (Cobot): these robots are built to work directly near to human operators, improving productivity rather than replacing manual labor. They guide workers to pick locations, optimize movement paths in real time and ensure that tasks are completed in the best way;

- Industrial robots: these robots differ from collaborative robots because they are not intended to share workspace with humans. For safety reasons, they are usually closed within protective fences. They are known for their ability to manage heavy loads at high speeds and, for this reason, they are ideal applications such as depalletizing in logistics or precision assembly in manufacturing environments [21];

- SCARA: this term stands for Selective Compliance Assembly Robot Arm. These robots are rigid along three axes while remaining flexible along the fourth so they have a controlled movement without sacrificing stability. This robot is mainly used for small components manipulation like pick and place tasks, sorting operations, and assembly ones [24];

2.4.3 Applications of Pick and Place Robots

The applications of pick and place robots are usually employed in packaging operations, bin picking and quality inspection. Some of the most frequent use cases and how these robots are typically applied is presented as follows:

- Assembly: pick and place robots grab incoming components from a source, such as a conveyor belt and place or affix them on another part of the product. Once the two parts are combined, the assembled components are transferred to the next stage of the assembly process;

- Packaging: During this operation, the pick and place robots pick objects from a predefined area or incoming flow and accurately place them into packaging containers;

- Bin picking: In these tasks, robots select items directly from bins that may contain randomly mixed parts. These types of robots are equipped with advanced vision systems, which can recognize differences in size, shape, and color, enabling them to pick the correct item before sending it to assembly or packaging location;

- Inspection: Mainly for quality control, pick and place robots use complex vision systems to detect products according to their defects or irregularities. Faulty items are identified, picked, and separated by placing them into designated areas;

2.5 Motion Planning

Motion planning is an important problem in robotics and consists of, given a robot and a description of the goal, finding a motion that approaches the goal and satisfies the constraints given by the problem. In the simplest form, the solution is to find a path between two states with the constraint of avoiding collisions. Adding differential constraints that are dynamical or more complex goal specifications makes the problem harder to solve. Given this complexity, a lot of motion planning algorithms renounce having optimality and completeness by going towards the way of weaker notions like resolution completeness, probabilistic completeness and asymptotic optimality. Sampling-based motion planning algorithms are the most used class of probabilistically complete algorithms [25].

The Open Motion Planning Library (OMPL) is a library for sampling-based motion planning algorithms that implements many state-of-the-art planning algorithms. OMPL is designed so users can solve challenging planning problems with minimal input [26].

2.5.1 Definitions of Sampling-Based Motion Planning

As presented in [26], sampling-based planners neglect strict completeness for practical efficiency and demonstrated that several planning problems can be solved despite the theoretical complexity. The key idea of sampling-based motion planning is to approximate the connectivity of the search space with a graph structure. The search space is sampled in different ways, and the samples that are selected will represent the vertices of the approximating graph. The edges represent feasible path segments. Two important considerations have to be made in the construction of the graph approximation and are the probability distribution in order to sample the states and the strategy used to generate edges. The components on which the sampling-based algorithms depend are:

- State Space: the points in the state or configuration space describe the system's state. For a free-flying rigid body, the state space includes all translations and rotations, while for a manipulator with n rotational joints, the state space can be modeled by an n-dimensional torus;
- Control Space: in the case of a system with dynamics, the control space parametrizes the space of controls. For many practical systems with m

controls, this can be thought of as a subset of $\mathbb{R}^m$. Geometric planning do not require a control space;

- Sampler: it is needed in order to generate different states from the stare space. In the case of control-based systems, a separate sampler in necessary for sampling different controls. Some algorithms need only a control sampler and not a state sampler;
- State Validity Checker: it is a routine that distinguishes the valid part of the state space from the invalid part, for example, by testing collisions, checking joint limits, or velocity/acceleration bounds;
- Local Planner: when planning with controls, the local planner is used to compute the evolution of the robotic system forward, and sometimes backward, in time. In geometric constraints planning, the local planner performs interpolation between states in the state space;

2.5.2 Overview of OMPL

OMPL is used in different fields and the central design criteria are:

- Clarity of concepts: OMPL was designed to consist of a set of components so that each part corresponds to a known concept in sampling-based motion planning;
- Efficiency: OMPL is implemented in C++ and is thread-safe;
- Simple integration with other software packages: OMPL offers abstract interfaces to facilitate integration with other software libraries. Instead of C++, OMPL can be compiled with Python bindings, which facilitates integration with Python modules;
- Straightforward integration of external contributions: OMPL keeps the planner API minimal so that new components can be added;

OMPL focuses only on motion-planning algorithms. Thanks to this minimalist approach, the OMPL's library can be used for a generic search in high-dimensional spaces under complex constraints. OMPL leaves the state validity entirely to the user or to the software package in which OMPL is integrated. This design gives users a lot of flexibility. The user can delegate the check of collisions to a physics engine, implement a state sampler that produces only valid states or define state validity in arbitrary ways that are dependent or not on geometry. OMPL also computes lots of tuning parameters, required by sampling-based planners, automatically [26].

2.6 Computer Vision

Vision is a human sense that provides us with a lot of information without physical contact. Artificial vision has achieved important results regarding industrial applications where tasks are well defined, and the environment is in some way structured. A vision system takes as input images and provides their description. Fundamental requirements are:

- It must have a relationship with the input image;
- It must include all the information to achieve a certain task;
- It must operate a compression of the information;

Let's consider, for example, a robotic arm that is equipped with a camera and that is required to pick an object with a certain gripper. So, the camera provides the information to the robot about the position and the orientation, relative to the gripper, of the object. Having this information, the control system can plan a trajectory that brings the arm into a suitable grasping configuration, knowing both the pose and the shape of the object. When this configuration is reached, the system can send the command to close the gripper and perform the execution of the picking.

The geometric relationship between one or more 2D images of the observed scene and the 3D world is the core of all the techniques for the estimation of the pose of objects in the robot's workspace or of the end-effector with respect to the objects that are around. For this reason, the process of calibration of the camera is necessary in order to calculate the intrinsic and extrinsic parameters [27]. Extrinsic camera parameters describe where the camera is located in the space and how it is oriented, and they are independent of the internal characteristics of the camera, such as focal length or field of view. Intrinsic parameters, instead, depend on how the camera captures an image. The intrinsic matrix of the model of the camera is composed by parameters such as focal length, aperture, field of view, resolution, etc. As can be seen in Figure 2.6, both intrinsic and extrinsic parameters can be represented as transformation matrices that convert points from one coordinate system to another, in particular: the extrinsic matrix performs a transformation between the world and the camera coordinate system, the intrinsic matrix allows the transformation between the camera and the pixels' image representation [28].

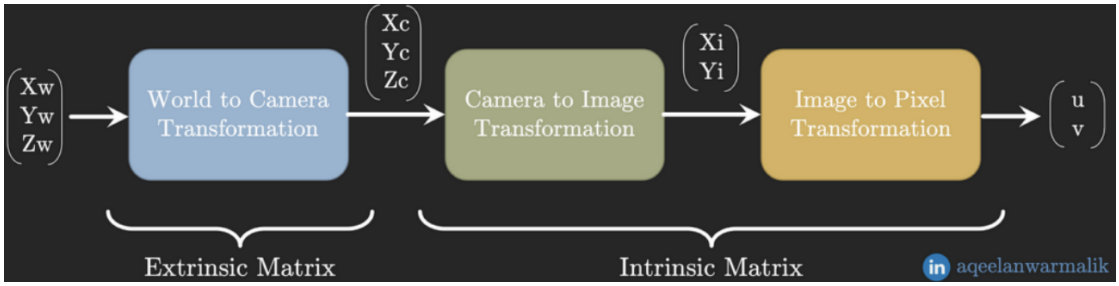

Figure 2.6: Flow of coordinate systems in computer vision [28]

A vision system can be implemented using a single camera or multiple cameras. If there two or more cameras looking at the same object, it is possible to estimate the depth by calculating its distance with respect to the visual system, so the cameras. It is possible to determine 3D data also with just one camera. The necessary requirement is to have two different pictures of an object taken from two different camera poses. In the case of having just one picture of the object, it is anyway possible to determine the depth by having a certain knowledge of the geometry of the object previously. This is the reason why, in most of the cases, there is a preference for the mono camera systems with respect the multi camera ones being easier to calibrate and also less expensive, although characterized by lower accuracy.

Another characteristic regarding vision in robotic manipulation is where the camera is placed. In mono-camera systems, there are two configurations:

- Eye-to-hand: the camera is fixed in the environment and it is looking at the robot's workspace from an external point of view;
- Eye-in-hand: the camera is mounted on the robot and, in the case of robot manipulators, it is placed near the end-effector, so it moves with the arm;

In multi-camera systems, these configurations are combined together into a hybrid configuration, with one or more eye-to-hand cameras and one or more eye-in-hand cameras [28].

2.6.1 Pinhole Camera Model

The main reference of this section is the article [29] in which there is a detailed description of the pinhole camera model.

To convert pixel measurements into coordinates in the real world, there is a need for a model of the process by which the images are captured by the camera. In other words, the idea is to map 3D points in the space into 2D points in the image. In order to achieve this objective, there is the pinhole camera model, which

describes a camera made like a box with a very small hole on one side. This hole is also called a pinhole aperture, and it represents the camera's optical center. Light rays pass through the pinhole aperture and project onto the opposite wall of the chamber, which is also called the image plane. In this way, an inverted projection of the observed scene is formed. The distance between the camera's optical center and the image plane is usually denoted by f and is known as the principal distance or focal length.

As also shown in Figure 2.7, in this model, two coordinate frames are used:

1. Camera coordinate frame $(x_{\text{camera}}, y_{\text{camera}}, z_{\text{camera}})$: is has the origin at the optical center and the $(z_{\text{camera}}$ is perpendicular to the image plane;

2. Image coordinate frame $(x_{\text{screen}}, y_{\text{screen}}$: it has the origin at the top left corner of the image plane, furthermore x_{screen} and y_{screen} correspond to pixels' columns and rows;

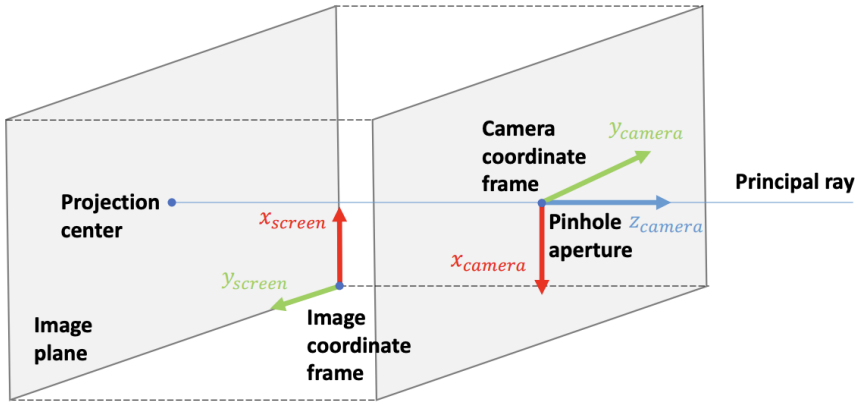

Figure 2.7: Coordinate frames of the pinhole camera model [29]

Knowing that the camera frame $(x_{\text{camera}}, y_{\text{camera}}, z_{\text{camera}})$ is right-handed and that the image plane is inverted, the axes of x_{camera} and y_{camera} with respect to the respective axes x_{screen} and y_{screen}. In order to avoid complicated calculations, the pinhole plane and the image plane are positioned virtually one in front to the other. The final result maps a 3D point expressed in world coordinates to pixel coordinates on the image plane and can be represented as:

$$x_{\text{screen}} = f_x \frac{x_{\text{camera}}}{z_{\text{camera}}} + c_x \tag{2.1}$$

$$y_{\text{screen}} = f_y \frac{y_{\text{camera}}}{z_{\text{camera}}} + c_y \tag{2.2}$$

In general, the position of a point in the space is known in the world coordinate frame and not in that of the camera. The world and the camera are strictly related by the rigid transformation $\mathrm{T}_{\mathrm{world}}^{\mathrm{camera}}$. So, the pinhole camera model is represented in matrix form as:

$$\begin{bmatrix} x_{\mathrm{screen}} \cdot w \\ y_{\mathrm{screen}} \cdot w \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_1 \\ r_{21} & r_{22} & r_{23} & d_2 \\ r_{31} & r_{32} & r_{33} & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{world}} \\ y_{\mathrm{world}} \\ z_{\mathrm{world}} \\ 1 \end{bmatrix} \tag{2.3}$$

The short form of the (2.3) equation is:

$$\begin{bmatrix} x_{\mathrm{screen}} \cdot w \\ y_{\mathrm{screen}} \cdot w \\ w \end{bmatrix} = \mathbf{MT}_{\mathbf{world}}^{\mathbf{camera}} \begin{bmatrix} x_{\mathrm{world}} \\ y_{\mathrm{world}} \\ z_{\mathrm{world}} \\ 1 \end{bmatrix} \tag{2.4}$$

In the end, the objective of the camera calibration is to find two parameters that are the intrinsic parameters of the camera, that are represented by the camera matrix M, and the extrinsic parameters of the camera, that are represented by the rigid transformation $\mathrm{T}_{\mathrm{world}}^{\mathrm{camera}}$ [29].

2.6.2 Detection of ArUco Markers

As presented in the OpenCV's official tutorial on ArUco marker detection [30], the estimation of the pose is one of the most important in lots of computer vision tasks, such as robot navigation, augmented reality and other applications. It is based on matching points in the physical world with their corresponding projections in a 2D image. This step is really complex, and for this reason, synthetic or fiducial markers are usually used to simplify the process. A solution that is largely used is based on binary square fiducial markers. Their main advantage is that a single marker alone provides enough information, through its four corners, to compute the camera pose. Moreover, the internal binary pattern makes these markers robust, enabling the use of error detection and correction methods.

As can be seen in Figure 2.8, an ArUco marker is a synthetic square marker characterized by a black border surrounding an inner binary matrix that specifies its identifier (ID).

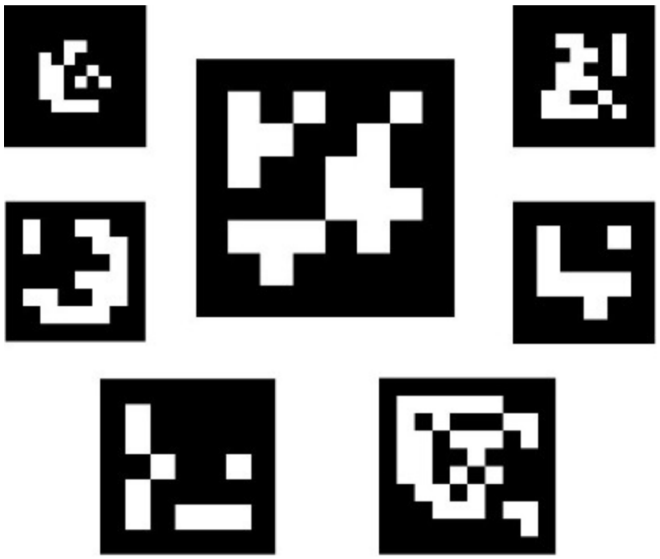

Figure 2.8: Example of ArUco Markers [30]

The black border is used to allow fast detection, while the binary codification enables identification and also error handling. The marker size defines the dimensions of the internal matrix; for example, a 4×4 marker contains 16 bits. Since markers could be rotated in the real scene, the detection algorithm has to recover the correct orientation so that each corner can be uniquely identified. This orientation recovery is achieved by the identification of the binary codification. Markers are grouped into dictionaries, which are collections of predefined binary codification used in an application. Each dictionary is characterized by two elements:

- The dictionary size, which is the number of markers it contains;
- The marker size is the size of each marker, so the number of bits;

Each marker is assigned an ID based on its position within the dictionary. For example, the first five markers in a dictionary are labeled with IDs from 0 to 4. When processing an image that contains ArUco markers, the detection algorithm outputs a list of detected markers. Given an image of an environment in which there are different markers, the detection has the objective of providing a list of the detected markers, in which each of them includes:

- the position of the four corners;
- the marker's ID;

The detection pipeline consists of two main phases:

1. Potential marker candidates are detected. The image is thresholded to segment the markers, after which contours are extracted. Shapes that are not convex or do not approximate the shape of a square are discarded, and also additional filtering steps to remove contours that are too small or too large or too close to each other;

2. Each candidate is analyzed by inspecting its internal binary code in order to confirm if it is actually a marker. A perspective transformation is applied to normalize the marker in order to extract the marker bits of each marker. This image is then thresholded to separate black and white bits. The marker is subdivided into cells according to the size of the border and of the marker itself, and the pixel distribution of black and white bits within each cell is analyzed to determine the corresponding bit value. The resulting bits are compared in the dictionary to confirm if one matches a valid marker, with the help of error correction techniques when required;

Chapter 3

System Design

The engine that is at the basis of all the robotic applications carried out and that powers the entire system based on this thesis is ROS2 Jazzy. In this work, all simulations were conducted using MoveIt2 and RViz, while all experimental results were obtained by actually executing pick and place tasks in the laboratory using a Ufactory xArm6 collaborative robot. All the part regarding computer vision was handled with OpenCV, and the object recognition in the scene was achieved thanks to the support of ArUco Markers.

3.1 Manipulator xArm6 of UFactory

As shown in Figure 3.1, the UFactory xArm6 is a 6-axis robotic arm manipulator. It has 6 degree of freedom, a 5 kg payload and ±0.1 mm repeatability.

Figure 3.1: xArm6 of UFactory [31]

Regarding the working space, it is able to reach a maximum distance of 762 mm along the horizontal axis and a maximum distance of 1029 mm along the vertical axis. The maximum speed is 1 m/s. For what regards the joint parameters, the maximum speed admissible is of 180°/s, and the working range of each joint is represented in Table 3.1.

Joint 1	$\pm 360°$
Joint 2	$-118° \sim 120°$
Joint 3	$-225° \sim 11°$
Joint 4	$\pm 360°$
Joint 5	$-97° \sim 180°$
Joint 6	$\pm 360°$

Table 3.1: Working range of each joint of the xArm6

The range of various motion parameters of the xArm6 is represented in the Table 3.2.

	TCP Motion	Joint Motion
Speed	$0 \sim 1000\,\mathrm{mm/s}$	$0 \sim 180°/\mathrm{s}$
Acceleration	$0 \sim 50000\,\mathrm{mm/s^2}$	$0 \sim 1145°/\mathrm{s^2}$
Jerk	$0 \sim 100000\,\mathrm{mm/s^3}$	$0 \sim 28647°/\mathrm{s^3}$

Table 3.2: Motion parameters' range

An important thing to avoid is singularities. They occur when the axes of two joints of a robotic arm are aligned along the same straight line. At the singularity point, the robot loses some degrees of freedom, which will cause the angular velocity of some joints to be extremely fast, leading to a loss of control. A typical example occurs when the wrist joint moves on or very close to the axis of the first joint. The characteristic of the singularity is that the planning movement cannot be performed correctly. When the robot performs motion planning near the singularity point, it will stop in order to avoid the high instantaneous speed of the joint.

Then an important role is played by the gripper, shown in Figure 3.2, that is the robotic arm's end-effector and is responsible for the dynamic grasp of objects. Its opening/closing range goes from -10 to 850, in which higher values correspond to a larger opening stroke, while lower values result in a smaller stroke. The presence of negative values is due to the increase in the clamping force until a specific object is

securely held. The gripper speed should typically be set between 1000 and 5000, also, the opening speed must be greater than or equal to the closing speed. [31]

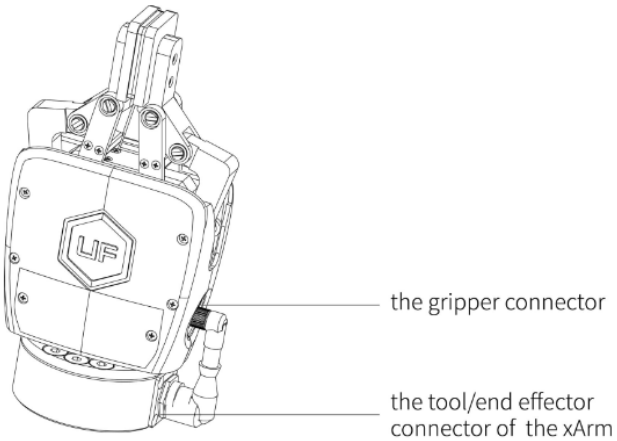

Figure 3.2: Gripper of the robotic arm [31]

3.2 ROS2

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications [32]. ROS 2 is a middleware framework that is built around an anonymous publish/subscribe model, which allows different processes to exchange messages in a reliable way. Every ROS2 system is organized around its heart that is known as the ROS graph. This represents the collection of nodes and the connections between them with which they communicate.

The basis of the ROS system is the "node". A node is a participant in the graph that uses a ROS2 client library to talk with other nodes. Nodes can communicate in the same process, in separate processes, or even on different machines over a network. Nodes are the main unit of computation in ROS and each of them is designed to handle a specific task. Nodes can publish data on named topics or subscribe to certain topics to receive data from other nodes. They can also act as a service client to request a computation from another node and obtain a response for themselves or to act as a service server to provide functionalities to other nodes. When the requested operation needs a longer time, nodes can act as action clients to request a computation from another node and obtain a response for themselves or to act as action servers to provide functionalities to other nodes. Moreover, nodes can provide parameters that allow their behavior to be configured at runtime. So, a single node often combines multiple roles at once; for example, it could publish and subscribe to several topics while also acting as a service server and an action

client.

ROS2 applications exchange information using one of three interface types: topics, services or actions. In ROS2, these interfaces are described using the "Interface Definition Language" (IDL). This description of the interfaces allows an easier automatic generation of the source code for the interface type in different target languages for the ROS tools. The supported types are:

- `.msg`: this file is a text definition that describes the fields of a ROS message. These text files are used for the generation of source code for messages in several languages;
- `.srv`: this file defines a service. It is composed of two parts that are a request and a response and both of them are message declarations;
- `.action`: this file defines an action and it is composed of three parts: goal, result and feedback, and each of them is a message declaration;

A message is one of the basic units of data exchange in ROS2. It allows a node to send data to other nodes over the network without expecting any response back.

Topics are one of the core communication mechanisms in ROS, and they should be used for continuous streams of data, such as sensor data, robot state, etc. In the publish/subscribe system, there are data producers (publishers) and data consumers (subscribers). Publishers and subscribers get in communication through a topic. When a publisher is created, there is a need to specify the topic name as a string, and the same name is given to the subscriber. Any publisher and subscriber using the same topic can exchange information. There can be zero or more publishers and zero or more subscribers on any specific topic. Whenever a publisher publishes data on a topic, every subscriber listening to that topic receives it. This system is one of the reasons why ROS2 is so flexible, because publishers and subscribers can come and go as necessary, meaning that debugging and introspection are an important part of the system.

In ROS 2, a service is essentially a remote procedure call in which one node sends a request to another node that will do the computation and return a response. Services are used for operations that complete quickly because the client typically waits for the response. They are not good for tasks that take a long time; in fact, they are better handled with actions. Services are identified by a service name that is similar to a topic name, even if it is in a different namespace. A service is composed of two parts: a service server, which receives remote procedure requests and produces responses, and a service client, which sends remote procedure requests and waits for the result.

An action follows a similar client/server structure but is designed for longer procedures. Actions are identified by an action name, similar to a topic name but

in a different namespace, and are composed of an action server and an action client. [32][33]

3.3 MoveIt 2

MoveIt 2 is the platform of robotic manipulation for ROS 2 and it deals with motion planning, manipulation, 3D perception, kinematics, control and navigation [34]. MoveIt2 is composed of a set of ROS libraries and it implements different functions; its pipeline is represented in Figure 3.3. It also uses RViz that is a visualization tool to effectively visualize what is happening in the scene during simulation.

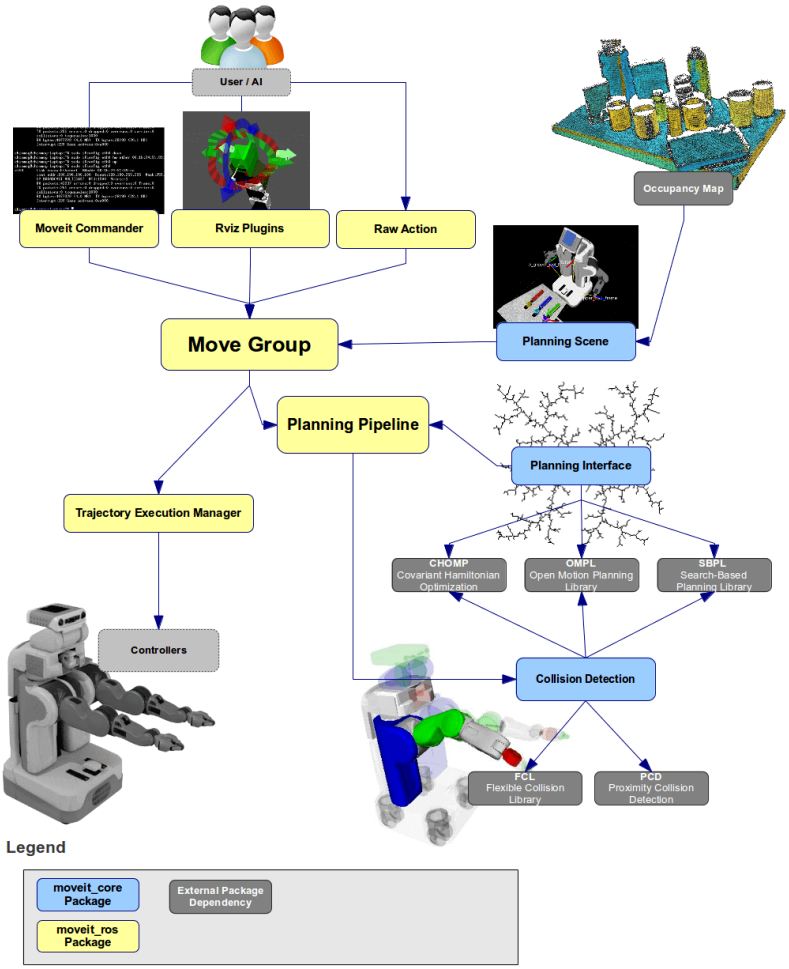

Figure 3.3: Pipeline of MoveIt [35]

As can be seen in the documentation of MoveIt2 [35], the core functions of MoveIt 2 are:

- Inverse Kinematics (IK): MoveIt 2 relies on a plugin infrastructure that is designed especially to allow users to customize their own kinematics algorithms. Forward kinematics and finding the Jacobian are directly provided through the `RobotState` class. By default, the inverse kinematics solution for MoveIt2 is based on the KDL (Kinematics and Dynamics Library) numerical Jacobian-based solver;

- Collision Checking and Avoidance: in MoveIt 2, collision checking is handled in the planning scene through the `CollisionWorld` component. In practice, MoveIt 2 allows users to never have the need to manage the low-level details of the collision checking and avoidance because it is carried out automatically by the FCL (Flexible Collision Library) package;

- Control and Execution: MoveIt 2 offers a motion control interface that connects in a strict manner with the robot's hardware, so the trajectories it plans can be executed in a reliable and accurate way;

- Motion Planning: MoveIt 2 provides motion planning functionalities based on OMPL and other planning libraries, giving access to many different algorithms and allowing planners to operate under various possibilities of constraints;

3.3.1 The `move_group` Node

The `move_group` node is the key node in MoveIt 2 and in Figure 3.4 the high-level architecture of it can be seen. This node serves as an integrator because it connects together the various subsystems in order to provide a consistent set of ROS services and actions that can be used by the user. As shown in Figure 3.4, users can interact with the capabilities provided by `move_group` in two ways:

- C++: through the `move_group_interface` package, which offers a convenient way to setup C++ interface to `move_group`;

- GUI (Graphic User Interface): using the Motion Planning plugin in RViz, which lets to plan, visualize and execute motions in an interactive way;

`move_group` is a standard ROS node and, from the ROS param server, it retrieves most of what it needs that is:

- URDF: it reads the `robot_description` parameter to load the URDF model of the robot;

- SRDF: it reads the `robot_description_semantic` parameter to load the SRDF model of the robot;
- MoveIt configuration: it loads additional configuration such as joint limits, kinematics, motion planning, perception and other parameters;

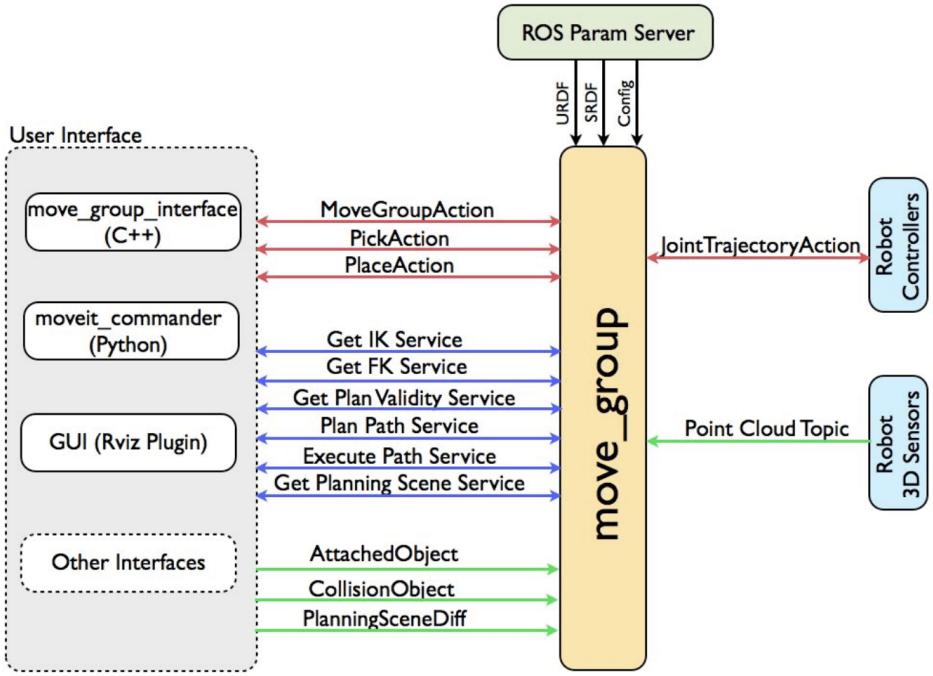

Figure 3.4: `move_group` node [35]

The node communicates with the robot in order to read the current robot state, like the joint positions, to receive sensor data such as point clouds and to talk to the controllers on the robot. To track the configuration of the robot, `move_group` subscribes to the `/joint_states` topic. The node is capable of listening to multiple publishers on this topic, even when each publisher provides only a portion of the information on the robot state. For global information about the pose of the robot, `move_group` relies on the ROS TF library. By listening to TF updates, it can solve transforms between coordinate frames. [35]

3.3.2 Motion Planning in MoveIt 2

As was already explained in Section 2.5, motion planning is an important problem in robotics and consists of, given a robot and a description of the goal, finding a motion that approaches the goal and satisfying the constraints given by the problem.

As explained in [35], MoveIt 2 interacts with motion planning algorithms via a plugin interface, so the user can choose and change planners from different libraries without changing the rest of the system. The interface to the motion planners is through ROS actions or services exposed by the `move_group` node that sends motion planning requests and receives planned trajectories. By default, `move_group` is typically set up to use OMPL; in addition, there are also default planners that are the Pilz industrial motion planner and CHOMP.

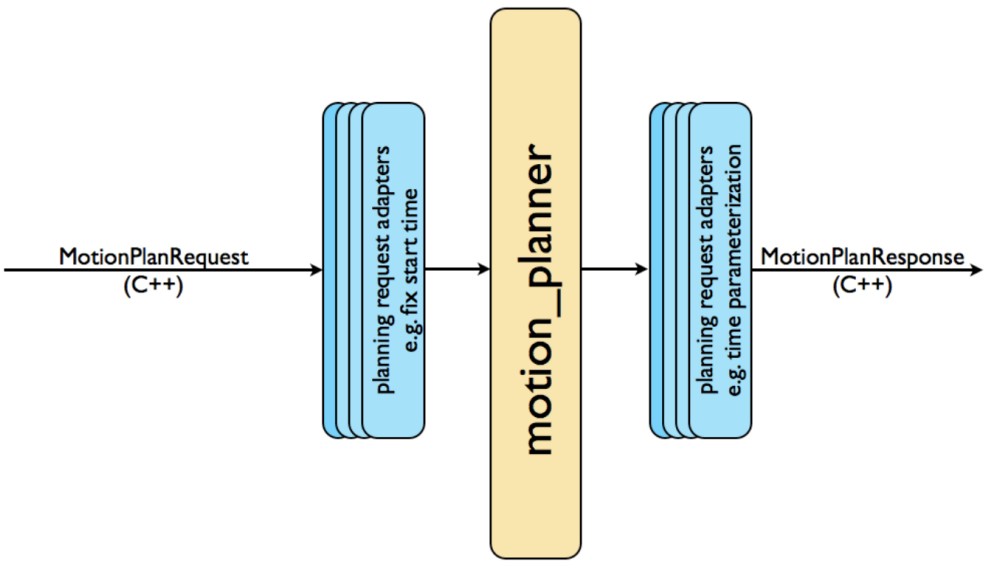

Figure 3.5: Motion planning flow in MoveIt 2 [35]

As can be seen in Figure 3.5, three elements in motion planning in MoveIt 2 are fundamental and are:

- Motion Plan Request: it tells the planner what kind of movement to do that means either moving a set of joints to a new configuration (joint space) or driving the end-effector to a target pose. By default, MoveIt 2 makes plans with collision avoidance. Moreover, the planner can be chosen by setting the parameter `planner_id`, which in this work of thesis was imposed to be the RRT-Connect, and also kinematic constraints can be added, like:
  - Position constraints: they force a link to remain in a specific portion of space;
  - Orientation constraints: they force the link's orientation (roll, pitch, yaw) within given limits;
  - Visibility constraints: they limit a point on a link to always lie in the field of vision of a certain sensor;

  - Joint constraints: they keep the joint between two imposed values;
  - User-specified constraints: these are rules imposed by the user on its own;

- Motion Plan Result: in response to a certain motion plan request, the `move_group` node will generate a trajectory that will move the arm to the desired location. The motion plan result is not a simple path but a particular trajectory that respects not only all the constraints listed before but also velocity and acceleration constraints at the joint level;

- Motion Planning Adapters: they are a sort of adjustments that are run before motion plan requests and after motion plan responses. Pre-processing can fix common issues like a start state that is slightly outside the joint limits of the robot, while post-processing can, for example, convert paths generated for a robot into trajectories that are time-parameterized;

3.3.3 Overview on Planners Tested in Simulation

As described on the OMPL website [36], let's have a general focus on the planners used in the simulation part conducted on MoveIt 2:

- RRT (Rapidly-exploring Random Trees): it is a tree-based motion planner that uses the idea of sampling a random state q_r in the state space, finding the state q_c, that is closest to q_r, among the previous states and expanding from q_c towards q_r, until a state q_m is reached. This state is then added to the exploration tree;

- RRT Connect: it's a bidirectional version of RRT but it usually outperforms the original RRT algorithm. The idea is to grow two RRTs, one from the start and one from the end, and try to connect them;

- RRT* (Optimal RRT): it is an asymptotically-optimal incremental sampling-based motion planning algorithm. This algorithm converges to an optimal solution of the path as a function of time;

- EST (Expansive Space Trees): it is a tree-based motion planner that tries to find the less explored area of the space by measuring the density of the explored space. The exploration is oriented to the lowest density part of the space;

- KPIECE (Kinematic Planning by Interior-Exterior Cell Exploration): it is a tree-based planner that uses discretization to guide the exploration of the space. This implementation uses a single level of discretization: one grid. The grid is imposed on a projection of the state space. When exploring the space

the preference is given to the boundary of this grid. It is important to set the projection used by the algorithm. If no projection is set, the planner will try to use the default projection associated to the state space;

- LBKPIECE (Lazy Bi-Directional KPIECE): this implementation is used if it is no default projection available, so two trees of exploration with lazy collision checking are used;

- PRM (Probabilistic RoadMap): it is a planner that builds a roadmap of milestones that approximate the connectivity of the state space. The milestones are valid states in the state space. Near milestones are connected by valid motions. Finding a motion plan that connects two given states is reduced to a discrete search in the roadmap;

- SPARS2 (SPArse Roadmap Spanner Version 2.0): SPARS is a planner that provides asymptotic near-optimality. SPARS2 uses a different approach in order to identify interfaces and compute the shortest paths through these interfaces;

3.4 `Explanation of xarm_ros2` Repository

In order to simulate, control and perform real tasks on the hardware of the xArm6 of UFactory in the ROS2 environment, there was a need to use its GitHub repository that can be found in [37]. The most important packages considered in this work of thesis are:

- `xarm_description`: this folder comprehends the URDF and XACRO models that describe the physical structure of the xArm, so the robot's links, joints, physical dimensions, mass, inertia and so on. It is the core of MoveIt simulation. By loading this model in RViz, it is possible to have a look at the robot and to customize it by adding grippers, sensors, or different end-effectors;

- `xarm_moveit_config`: this folder includes the configuration files of MoveIt 2 that are used for motion planning and manipulation. It is possible to set up planning groups for the robot, like the arm or the gripper. It is possible to use this package in order to plan and execute trajectories by using MoveIt 2 through RViz or coding (Python in this thesis work). Moreover, solver parameters can be changed for certain applications. Thanks to the use of this package, different and advanced motion planning scenarios were tested;

- `xarm_msgs`: it defines ROS 2 service and message definitions for the xArm. The format of the files is `.srv` and `.msg`;

3.5 From "Engineer in the Loop" to "User on the Loop"

Nowadays, as represented in Figure 3.6, the ability of the robot to adapt and improve is constrained to the presence of a technical user, an engineer, who closes the loop between failure and correction. In this context, the engineer is "inside" the cycle, modifies code and necessary parameters, collects data, and performs the test another time. The idea behind this thesis work, as can also be seen in [4], is to have the user "on" the loop that, now, is not the technical programmer but a "supervisor". The user provides prompts, objectives, and access to tools and APIs, and the model becomes the interpreter that tries to transform the user's request into operational steps. The cycle of deploying and improving does not disappear but there is a change in the nature because it iterates with high level instructions, feedback and corrections that are a sort of conversation like: "You were wrong, I told you to do first X and then Y". Therefore, the idea is to shift the focus from "writing a software" to "asking objectives and guiding the execution".

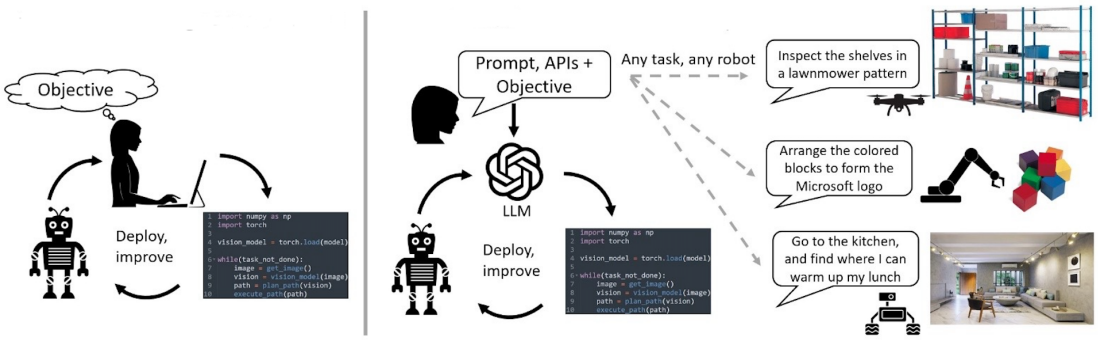

Figure 3.6: Engineer in the loop and user on the loop pipelines [4]

The user on the loop approach does not represent a total autonomy of the model but a mediation in which the user remains part of the system because of the ambiguity that could arise from several task requests. The LLM can fill the gap between natural language and robotic action, but without eliminating the necessity of supervision from a human user.

Chapter 4

Workflow Implementation

This thesis has the objective to demonstrate the potential of different GPT models (in this context, case GPT-4.1, GPT-5, GPT-5.2) when faced with robotic applications under three different prompting strategies (zero-shot, one-shot, and few-shot). To evaluate GPT's capabilities in planning and code generation, a "user on the loop approach" was adopted, where, for each trial, a new chat has to be opened with the chosen model, and a human user, not necessarily an expert, should be on the loop to do refinement and validation by giving text-based feedback.

4.1 End-to-End Workflow

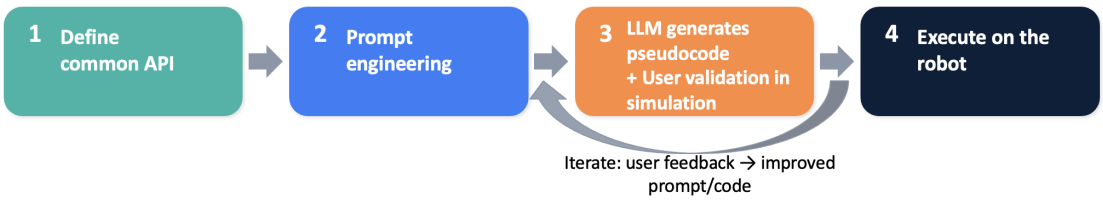

Figure 4.1: General workflow

As already presented in Section 2.3 and also as can be seen in Figure 4.1, the general workflow used in this work to enable the effective use of ChatGPT in robotic applications was:

1. Definition of a common API library;
2. Construction of a prompt for ChatGPT to specify objectives and constraints;
3. User on the loop for code validation through simulation;

4. Execution on the xArm6;

The idea of the "user on the loop" works if the LLM is considered as a robotic assistant and planner that receives a natural language request and converts it into a combination of APIs.

In this thesis work, the intention was to insert a sort of black box in which all the information that the LLM cannot see and know is inserted, so there exist:

- Visible for the LLM: the text-based message in natural language that requests the specific task and the names of the high-level APIs, with the explanation of their role;
- Hidden for the LLM: the part regarding the robot core, so controllers, scene parameters, configurations, motion primitives, all the details regarding the construction of the APIs, and so on;

4.2 High-Level Function Library

The first step of the work was to define a high-level API function library that is a sort of bridge between natural language requests and effective operational capabilities of the robot. An example is provided in Figure 4.2. The intention was: instead of allowing each model to generate code or low-level controls in a free manner, there is a need to constrain the three different models of GPT (GPT-4.1, GPT-5 and GPT-5.2) to operate only with the available functions of the created library. In this way, the role of the LLM is not that of the programmer, but rather it composes an action plan by using predefined functions. It is of paramount importance the fact that APIs has to be interpretable both for the LLM and for the non-technical user. They should be composed by descriptive and coherent names (e.g., locate_location(), locate_object(), pick_object(), place_object()) and the set of actions covers the main use cases without entering into details of implementation.

```
601  def home():
602      # do something
603      return
604
605  def observe_scene():
606      # do something
607      return
608
609  def pick_object(name_or_alias: str):
610      # do something
611      return
612
613  def place_to(location_name: str):
614      # do something
615      return
```

Figure 4.2: Example of a part of the task-relevant robot API library

So, the API library becomes the boundary of the black box because the LLMs see the list of available functions, but what happens within each of them remains hidden. There are two main advantages:

- This way to proceed increases robustness and safety of execution because the concrete action always passes under controlled functions;
- This modality reduces the risk of hallucination because the model cannot invent new functions or manipulate low-level details;

Another fundamental aspect is that the APIs can be chained together to form more complex functions. This allows different models to be compared in a fair manner because if each of them is bound to the same set of functions, performance differences arise in the ability to plan, decompose the goal and handle particularities. The common API library plays also a key role in reproducibility. With a defined and stable API set, system behavior becomes more traceable because a task can be described as a sequence of known and precise functions.

4.3 Construction of the Prompt

In the second step of this thesis work, the focus was given to the prompt, starting from the definition of a common baseline prompt that was divided into the three prompting strategies: zero-shot, one-shot, and few-shot. The idea is to build a stable framework that allows the LLM to act as a high-level planner, respecting the constraints and the black-box intent presented in Section 4.1. The baseline prompt is given to the LLM before the user's request. In this prompt, what is specified is:

- The system prompt that represents the role of the model. In this work, the fixed system prompt was:

  "From now on, we're working with a robot manipulator, specifically an xArm6 robotic arm, so a 6 DoF robotic arm, with a gripper attached to the end effector, and your role is to be a robotic assistant and planner";

- Constraints of safety and behavior, like collision avoidance and asking clarifying questions when information is missing;
- The requirement to use only the set of allowed APIs;
- The required output format in this work was a pseudocode in Python style;

This baseline prompt is kept the same through all the experiments and is provided ready-made, so that even a non-technical user can interact with the system without knowing implementation details or without writing complex instructions.

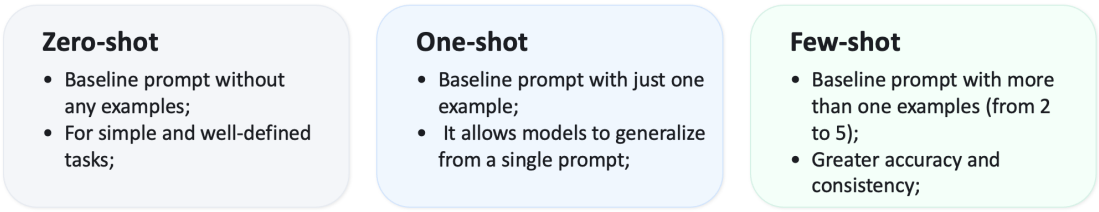

Figure 4.3: Prompting strategies adopted

As presented in Figure 4.3, the three prompting strategies implemented starting from the baseline prompt are:

- Zero-Shot: with this strategy, the LLM has to generate the pseudocode for the requested task by relying only on the rules of the baseline prompt without having access to examples that could help to reason. The zero-shot prompting measures how well the model can respect the constraints and the format without relying on previous solutions;
- One-Shot: with this strategy, the LLM has to generate the pseudocode by having access to the fixed baseline prompt and just one example of pseudocode about a specific task. In this case, the example is important to show to the model, for example, how to decompose a task, how to chain APIs, how to guarantee safety, and how to keep the required format of the output. The idea is that a single example increases consistency and reduces errors like omitted steps or improper API use;
- Few-Shot: with this strategy, the LLM has to generate the pseudocode by having access to the fixed baseline prompt and two examples of pseudocode about two particular tasks. In this case the model receives a larger sample of resolution patterns, like two different ways to decompose and validate a plan. The goal is that one of achieving greater stability and robustness in the output. With the few-shot approach, there is more coherence and consistency because the model can better understand the structure of the reasoning;

4.4 User on the Loop Implementation

In the third step of the workflow, as shown in Figure 4.4, the “user on the loop” approach was made explicit, in which even the non-technical user actually enters the development and validation cycle of the behavior of the robot. The idea is to transform the interaction with the LLM into a validation and refinement cycle that tries to guarantee the correctness of the task and safety in the execution on the real hardware.

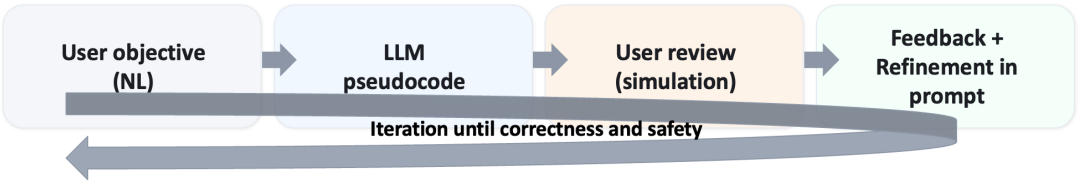

Figure 4.4: Flow of the "user on the loop" approach

In the operational flow, the user makes a text-based request in natural language, and the LLM translates it into pseudocode based on the high-level APIs defined in Section 4.2 and according to the rules of prompting defined in Section 4.3. However, at this point, there is no execution on the robot because there is first the user review, which is the central phase of this step, through simulation on MoveIt 2, as shown in Figure 4.5, or through direct inspection of the pseudocode itself.

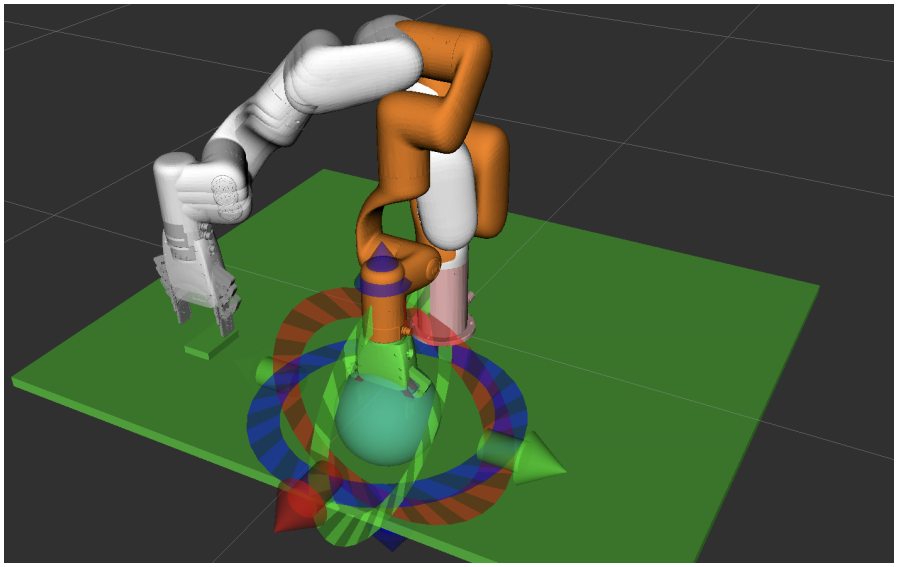

Figure 4.5: Example of simulation on MoveIt 2

Now, the user does not necessarily have to understand controls, planners, or parameters, but has to avoid potentially dangerous or ambiguous behaviors. The user should evaluate if the plan actually makes sense, like: "Is it picking the right object?"; "Is it placing the object in the right location?"; "Are there missing steps?"; "Is the order of the actions reasonable?". In other words, the responsibility of the user is to approve or to correct the idea of the plan generated by the LLM.

If there are errors or ambiguities, the "refinement phase" plays its role: the user provides natural language feedback that is used to refine the goal and regenerate the

plan and the pseudocode. This cycle of goal → pseudocode → review → feedback, is repeated until correctness and safety of the result. This is where the user on the loop approach differs from the idea of a chatbot controlling a robot, because the entire system is able to correct errors before real deployment and to fix real problems that could arise, like incomplete specifications, unspoken constraints, and so on.

In order to increase the reproducibility, the guideline followed was to open a new chat for each new task, reducing accumulated context and conversational drifts, which could amplify errors and hallucinations. The checks performed through simulation or inspection are of fundamental importance because they act as a filter for the actual implementation. Iterations improve not only the quality of the plan but also intercept possible failures before they become real failures on the robot. It is important to notice that in this work the maximum number of iterations of refinement was three.

Finally, in this step, models and prompting strategies are compared systematically. The "user on the loop" approach was applied to the models GPT-4.1, GPT-5, and GPT-5.2, each under the zero-shot, one-shot, and few-shot prompting strategy defined in Section 4.3. In this way, the comparison is not simply "which model generates the best plan first" but also "how quickly a correct solution is got". So, the entire pipeline becomes a controllable iterative process until reaching safety and correctness.

4.5 Execution on the Robot

In the fourth step, the workflow moves to the implementation on the physical robot. This is the step in which all the design choices made by the engineer in the first two steps and all the parts regarding the user on the loop approach, are stressed by real-world conditions like mechanical tolerances, friction, small movements and so on. So now it is not a matter of generating good code but of demonstrating that the plan that was created before can be executed in a consistent and robust way through the API library and the supervision of a non-technical user. With the execution on the real robot, three main objectives were demonstrated:

1. Accuracy in the execution of the task: accuracy refers, in addition to precision in the process of pick for example, to the ability of the system to translate the high level plan into physical actions that satisfy the task with precision, like: picking the corrected object and manipulating it, placing the object in the intended location, avoiding inconsistent placements or incomplete sequences. In this sense, the accuracy does not regard only the single sequence of the pick and place, but in particular, the correspondence between the intent of the user for the task and the observable final result. In Figure 4.6, there is an

example of a high-accuracy multiple pick and place with the following initial message given by the non-technical user:

"Take all the objects in the center of the table and arrange them according to size within the locations";

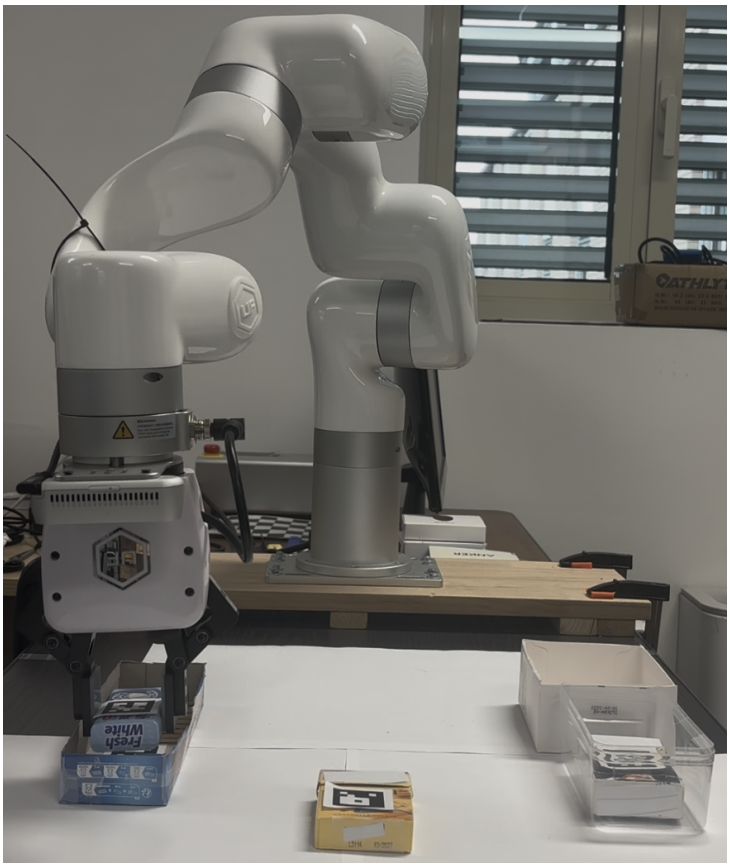

Figure 4.6: Example of accuracy in multi pick and place

2. Dynamicity of the system: dynamicity refers to the ability of the system to perceive and react to possible continuous movements of objects in the scene during execution. In other words, the system does not assume that the world remains static between planning and execution because if an object is moved, if it appears or disappears from the workspace or if its pose is accidentally altered, the system must be able to recognize it and adapt the sequence of actions, for example: avoiding grasping an object already in the desired position, or updating the picking pose. In Figure 4.7, an example of dynamic stacking of different objects can be noticed, with the following initial message provided by the non-technical user:

"Take all the objects dynamically present within the scene and place them in the center of the table by stacking them";

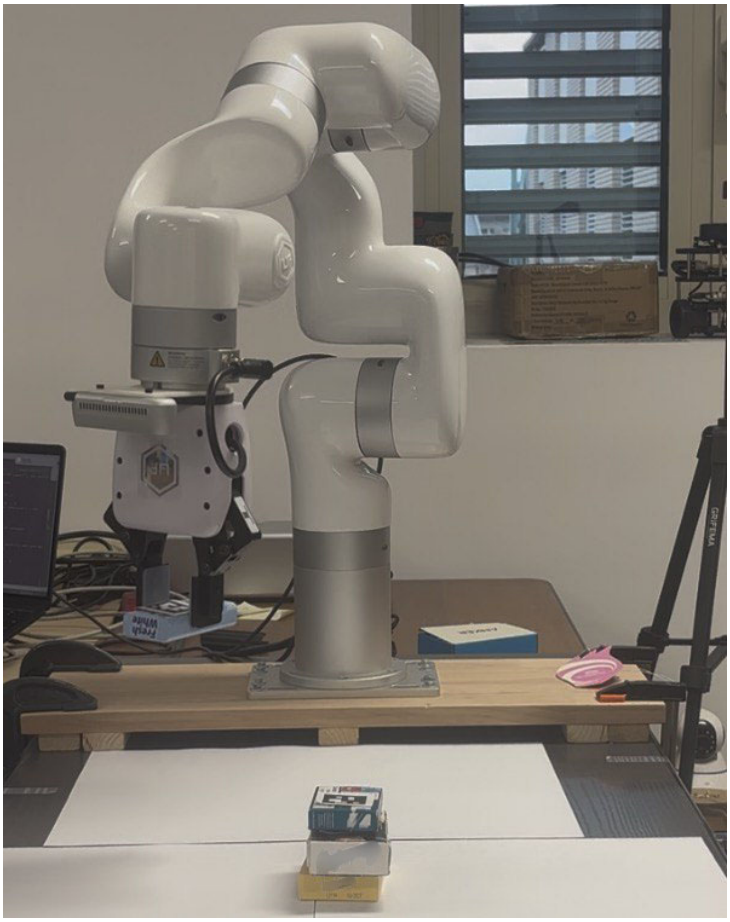

Figure 4.7: Example of dynamic stacking of multiple objects

3. Human-Robot Collaboration (HRC): this objective refers to the interaction in a shared space. The robot has to operate in collaboration with the non-technical user, respecting constraints like maintaining a sequence that makes sense also in collaboration with the human. The HRC is important because now the user is not a spectator of the scene but an active participant in the process. In Figure 4.8, an example of HRC can be seen, in which there is a box, with inside two ordered objects, that is given by the robot to the non-technical user. The initial message sent to the LLM was:

   "Take the two items and arrange them in the compartments of the box, then bring the box to me placing it in my hand";

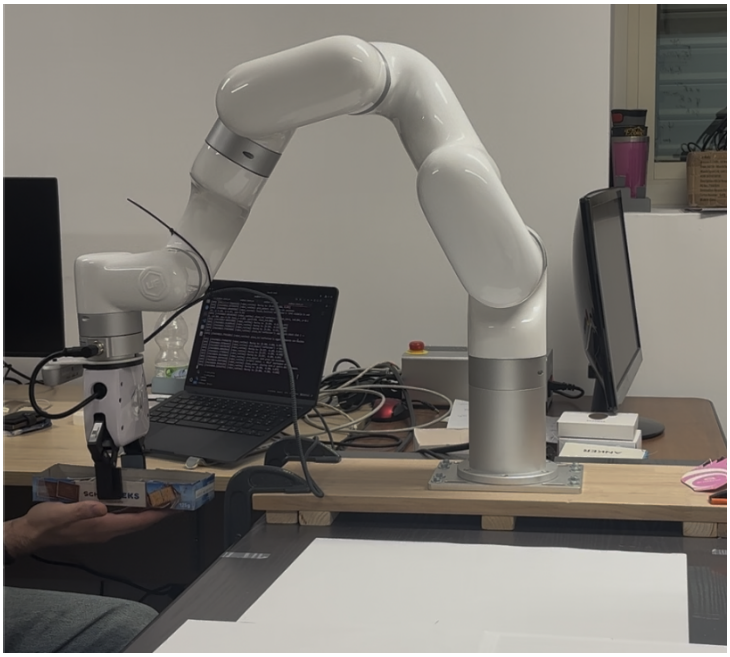

Figure 4.8: Example of HRC by providing an object to the user

4.6 Pose Estimation

In order to obtain the poses of the different objects that are in the scene, ArUco Markers played a fundamental role. This information is also present in a more theoretical key in the Section 2.6, but in particular in the Subsection 2.6.2.

First of all, instead of relying directly on the number of the marker's ID, which is not descriptive or robust enough, a layer of abstraction was introduced in which the object is recognized by its ID, but it is mapped to a name and to an accurate description. To achieve this, the information was organized into four YAML files, separating the data and making it more readable, extensible, and easy to modify without modifying the low-level code.

1. Catalog of manipulable objects: in this catalog, each object is indexed by its marker ID and contains:
   - `name`: the name of the object that is then used by the non-technical user;
   - `size(x,y,z)`: the dimensions of the object, which is useful for operations like planning, collision checking, or gripping constraints;
   - `gripper_pick_svc`: this part is dedicated to the gripper parameters, where the aperture values needed to correctly grasp the object, based on its orientation, were specified. This is important because the same

geometry could require different apertures if the picking process comes from different sides;

In general, what really matters for the manipulation is these files;

2. Aliases of the objects: in this file, the main aliases of the catalog object were added to recognize them. The idea is that the same object can be called in different ways, but the system has to connect all these variants to a single ID. This makes the interaction more flexible because, instead of forcing those who use the system or the LLM to remember an exact name, there are accepted variants that can map correctly to the desired object;

3. Catalog of place locations: In this file, a separate catalog was created for locations that are, in this work of thesis, objects that are not necessarily pickable, but their main purpose is to represent an area of storage. For example, there are boxes with internal compartments used for sorting tasks. Also in this case, each location is directly linked to a marker ID, and it is described as:

   - Name of the location;
   - Information about its dimensions and, above all, its internal structure, like, as said before, information about internal compartments or geometry useful for understanding where to place something;

4. In this file, the same logic is as in the second file, but for the locations. There was a definition of the aliases of the locations, so that each of them can be recalled in a natural way and is tolerant to variations;

In Figure 4.9, an example of ArUco Markers placed on different objects used during the experimental activities is presented.

Figure 4.9: Example of ArUco Markers placed on different objects

Subsequently, the detection of the objects was reached thanks to the support of OpenCV, and the following main instructions were used:

- `aruco_dict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_5X5_100);`
- `detector_params = cv2.aruco.DetectorParameters();`
- `corners, ids, _ = cv2.aruco.detectMarkers(gray, aruco_dict, detector_params);`

In order to obtain the pose estimation of the object, the most important step is the transformation from the reference frame of the object to the base reference frame of the robot through the following matrix multiplications:

$$T_{base}^{obj} = T_{base}^{tcp} \cdot T_{tcp}^{camera} \cdot T_{camera}^{obj} \tag{4.1}$$

In order to obtain the homogeneous transformation matrix, T_{tcp}^{camera}, between the tool center point (TCP) and the `camera_color_optical_frame`, the following command on the terminal was launched:

`ros2 topic echo -once /camera/camera/color/camera_info`

In particular, Figure 4.10 shows the TCP, the camera, and the base frames of the robot model through the support of RViz.

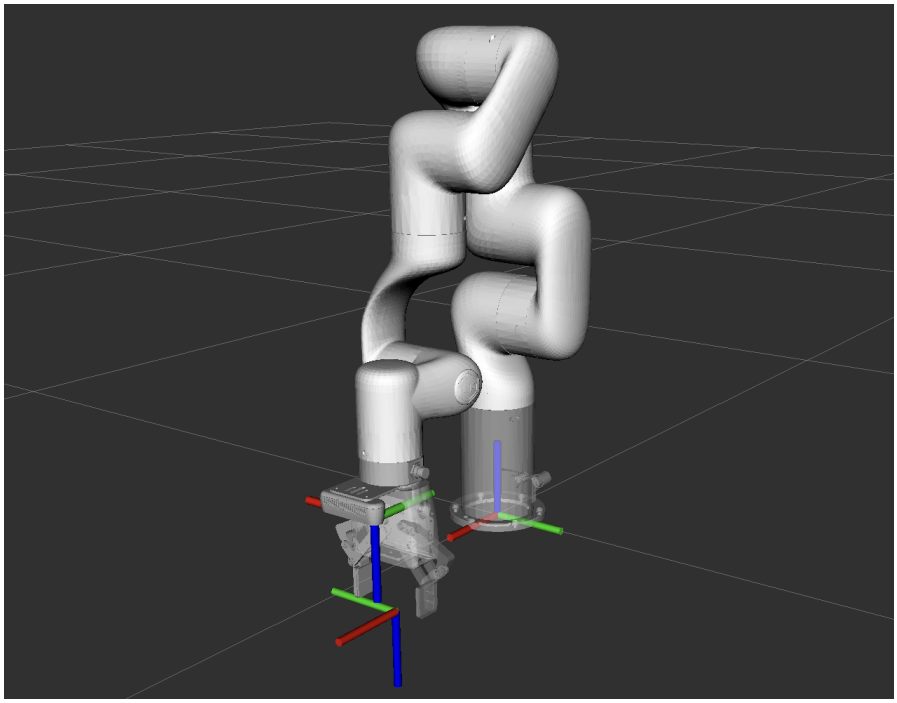

Figure 4.10: Base, camera and TCP frames of the robot model

Chapter 5

Experimental Results

The experiments were designed to evaluate the impact of the model of LLM used and the prompting strategy adopted. The evaluation is divided into three parts:

- A comparison of the three different models that are GPT-4.1, GPT-5 and GPT-5.2;
- A comparison of the three different prompting strategies that are zero-shot, one-shot and few-shot;
- An analysis of the successes achieved;

Validation was conducted by executing 9 tasks designed with increasing difficulty. For each task, the system received a maximum of 3 refinement feedback, in which, if execution did not lead to the correct execution of the task within this limit, the test was classified as failed (Fail). Another result that is presented regards the choice of the planner.

5.1 Choice of the Planner

The choice of the planner influences the overall quality of the execution, in particular in the manipulation tasks presented in this work, such as pick and place, ordering, stacking, and delivering objects to a human collaborator or in a workspace shared with the human operator. For this reason, the main objective is to guarantee rapid and repeatable planning with the ability to plan another time when the scene changes, for example, if objects are dynamically moved or if there are changes in the shared workspace.

For this reason, the idea was to choose sampling-based approaches available in OMPL, which tend to provide rapidly feasible solutions, making them suitable for tasks like multi-pick and place and scenes with frequent updates of the scene.

Several analyses were performed on the 8 proposed planners in Section 3.3.3. For each of the 3 simulations conducted, 20 tests for every planner were carried out. The simulations consisted of the pick and place of rectangular prisms with the following characteristics:

- A prism measuring 7.4 × 4.8 × 1.5 cm was moved from position A (x=0.40 m; y=0.01 m) to a position B (x=0.37 m; y=-0.20 m);
- The same prism as the previous was moved from a position A (x=0.35 m; y=0.25 m) to a position B (x=0.25 m; y=-0.35 m);
- A prism measuring 5.5 × 8.8 × 2.3 cm was moved from position A;(x=0.25 m; y=-0.35 m) to a position B (x=0.35 m; y=0.25 m);

By looking at Table 5.1, 5.2 and 5.3, it can be noticed that the success rate of all of these simulations was more or less 100%, so the idea is to make a comparison with the mean planning time for each planner in each simulation:

OMPL Planner	Mean Planning Time	Failure Rate
EST	23.52 s	0%
KPIECE	23.49 s	0%
LBKPIECE	83.95 s	5%
PRM	22.14 s	0%
RRT Connect	22.05 s	0%
RRT	23.14 s	0%
RRT*	66.73 s	0%
SPARS2	66.3 s	0%

Table 5.1: Results of the first simulation on planners

OMPL Planner	Mean Planning Time	Failure Rate
EST	29.37 s	0%
KPIECE	29.94 s	0%
LBKPIECE	90.74 s	10%
PRM	27.96 s	0%
RRT Connect	27.94 s	0%
RRT	29.19 s	0%
RRT*	72.36 s	0%
SPARS2	72.52 s	0%

Table 5.2: Results of the second simulation on planners

OMPL Planner	Mean Planning Time	Failure Rate
EST	29.95 s	0%
KPIECE	32.49 s	0%
LBKPIECE	89.93 s	10%
PRM	28.30 s	5%
RRT Connect	28.29 s	0%
RRT	29.72 s	5%
RRT*	72.86 s	0%
SPARS2	72.71 s	0%

Table 5.3: Results of the third simulation on planners

The previous data are the results obtained from the different simulations. From these outputs, the final choice for performing the tasks fell on RRT Connect because it results the best in overall performance, as can be seen from the tables.

5.2 Models' Comparison: GPT-4.1 vs GPT-5 vs GPT-5.2

This part of the experimental analysis is dedicated to comparing the three models GPT-4.1, GPT-5 and GPT-5.2. The comparison is performed with the same prompting strategy, so the models are evaluated separately under zero-shot, one-shot, and few-shot. For each strategy, a graph of comparison is constructed. For example, in the zero-shot case, GPT-4.1 vs GPT-5 vs GPT-5.2 are directly compared under the same conditions. On the graphs:

- X-axis: the 9 tasks are shown and are ordered in increasing order of difficulty;
- Y-axis: on this axis, the metric is "Correct Steps" (%), which measures the percentage of correct steps in the pseudocode generated by the model compared to a "golden code", which is a reference code.

The golden code represents a target solution, a pseudocode considered correct, consistent, secure and functional, used as a baseline to evaluate how the generation of the LLM is coherent with respect to the expected sequence of actions. So, Correct Steps (%) quantifies how well the model reproduces the logical structure of the solution, task by task, as complexity increases.

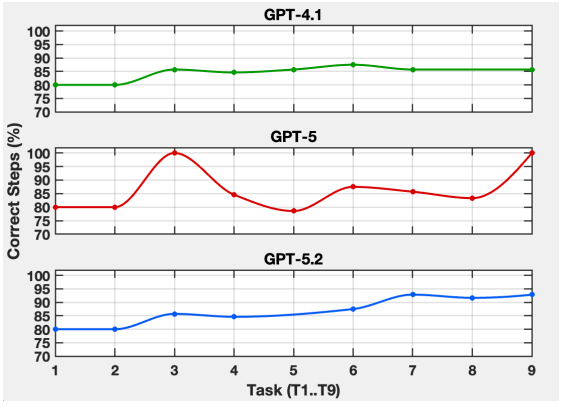

Figure 5.1: Comparison of the models under ZSP

In the zero-shot case as can be seen in Figure 5.1, the three models show overall performance that are similar regarding the correctness of the steps but with the following considerations:

- GPT-4.1 achieves an overall correct steps rate of 85%, calculated only on successfully completed tasks by considering that 1 out of 9 tasks is failed. This indicates that there is a good ability to generate a correct pseudocode for the completion of the task, but an imperfect robustness on the general set;
- GPT-5 achieves 87% overall correct steps, showing an improvement over GPT-4.1 in average accuracy and in reliability, taking into consideration that all tasks were completed without failures;
- GPT-5.2 achieves 88.5% but with 1 out of 9 fails. Therefore, despite the highest percentage of overall correct steps among the three models by considering that had a failure like GPT-4.1;

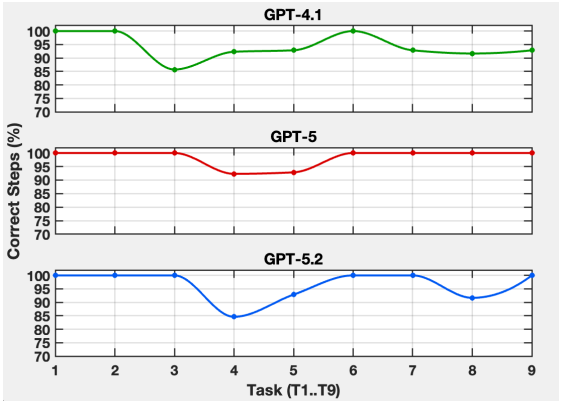

Figure 5.2: Comparison of the models under OSP

In the one-shot case, as can be seen in Figure 5.2, all the models completed the 9 tasks without any failures, so the focus of the comparison was directed exclusively on step correctness:

- GPT-4.1 achieves an overall correct step of 93.5%, indicating a high adherence to the golden code;
- GPT-5 achieves the highest value, an overall correctness of the steps of 97.8%, highlighting a greater ability to produce pseudocode more aligned with the reference solution;
- GPT-5.2 scores 95.7%, over GPT-4.1 but below GPT-5, while maintaining a high and consistent performance with no fails;

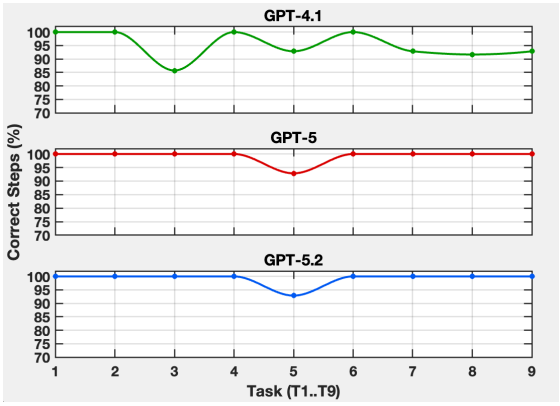

Figure 5.3: Comparison of the models under FSP

Even in the few-shot case, as can be seen in Figure 5.3, no failures were observed, so 0 fails for all the models. The differences, therefore, arise only between the generated pseudocode by the LLM and the golden code:

- GPT-4.1 achieves 94.65%, improving the overall quality of the correctness of the steps;
- GPT-5 achieves an overall correctness of the steps of the 98.9%;
- GPT-5.2 also achieves 98.9%, as well as GPT-5 in overall accuracy;

In general, considering the three scenarios, the findings that arise comparing the models are: GPT-4.1 is the least performing model regarding the accuracy among the other two models, and also it runs into a failure, showing that it is not the most robust. GPT-5 is the most stable model in the comparison; in fact, it is the only one with zero failures and maintains high performance. GPT-5.2 tends to reach very high correct steps (%) and therefore has very high accuracy, even if it shows one fail, like with GPT-4.1.

5.3 Prompting Strategies' Comparison: Zero-Shot vs One-Shot vs Few-Shot

The second part of the experimental analysis is dedicated to comparing the adopted prompting strategies that are zero-shot, one-shot and few-shot. In this case, the comparison is made by fixing the model and, for each LLM (e.g., GPT-4.1), it is analyzed how performance varies from zero-shot to one-shot and few-shot. This allows to isolate the effect of the prompting strategy on the results. As also seen in Section 5.2, in the graph, on the x-axis are shown the 9 tasks in an increasing order of difficulty, while on the y-axis is shown the metric of the correct steps (%), representing the percentage of correct steps in the pseudocode generated by the LLM compared to a reference code called golden code.

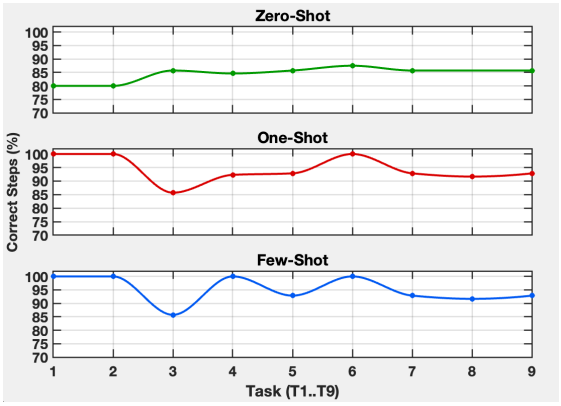

Figure 5.4: Comparison of the prompting strategies under GPT-4.1

As can be seen in Figure 5.4 for GPT-4.1, the transition from zero-shot prompting (ZSP) to one-shot prompting (OSP) shows a clear improvement:

- In ZSP, the overall correctness of the steps result 85%, and one case of failure is observed. This is the most challenging case for the model because no pseudocode example in the prompt is provided, so there is less guidance in the solution structure;
- In OSP, the accuracy increases to 93.5% with 0 fail and this demonstrates a marked improvement; in fact, introducing a single pseudocode example in the prompt brings the model to be more correct and robust, like in the case of the golden code;
- In FSP, the overall correctness of the steps is 94.6% with no failures, so it results in a further improvement with respect to the OSP. The gain exists but is smaller with respect to the large gap observed between the ZSP and the

OSP. It results in adding a second example, consolidating the quality of the pseudocode related to the execution of the task;

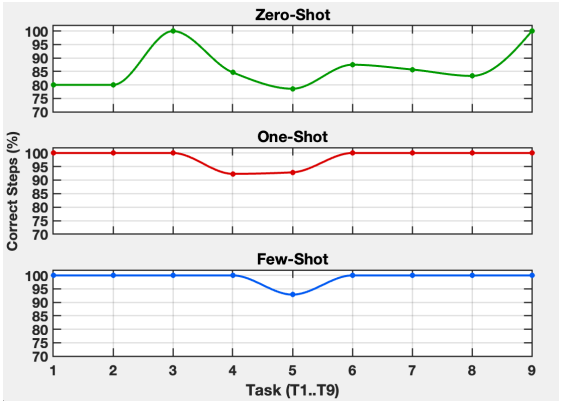

Figure 5.5: Comparison of the prompting strategies under GPT-5

As can be seen in Figure 5.5 for GPT-5, the behavior of the prompting strategies is even clearer with respect to the previous case because:

- In ZSP, the overall correctness of the steps is 87%. Even without examples, the model is solid, but there still exists a significant difference compared to strategies with examples;
- In OSP, a very marked jump to 97.8% is observed, so the introduction of just one example produces a huge improvement in the alignment of the steps with the reference code;
- In FSP, the value increases further to 98.9%, so a smaller increase compared to the large gap that exists passing from ZSP to OSP;

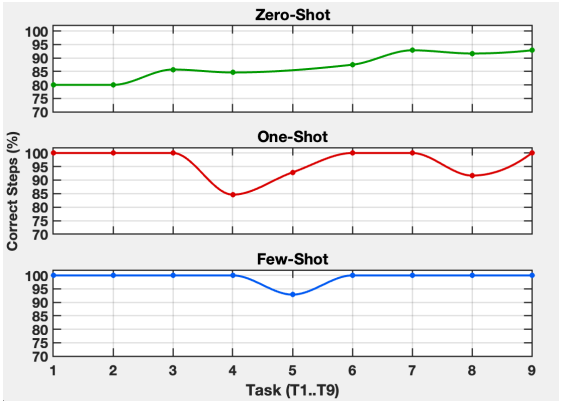

Figure 5.6: Comparison of the prompting strategies under GPT-5.2

As can be seen in Figure 5.5, for GPT-5.2, a similar dynamic as before is observed, but with a more gradual progression between one-shot and few-shot:

- In ZSP, the overall correct steps rate is 88.5%, so the model is competitive anyway, but remains far from the maximum performance achievable with explicit examples;
- In OSP, it rises to 95.7%, confirming the evident gap between the absence of examples and the presence of at least one example. The model tends to adhere better to the golden structure;
- In FSP, it reaches 98.9%, showing a more significant improvement with respect to the increase observed in the other two cases between OSP and FSP. The second example provides an additional context that is useful to reach a sequence of steps that is certainly safe and robust;

Overall, the analyses show a consistent behavior of the prompting strategies. ZSP (0 examples) is the weakest scenario because the LLM has to deduce the form and structure of the pseudocode from scratch with greater variability. It is also important to notice that it is the only case in which there are failures. OSP (one example) introduces an important gap in the overall correctness of the steps in the pseudocode. The gap between ZSP and OSP is the most marked because even a single example works as an implicit template that guides the generation of the pseudocode for the LLM. Indeed, it improves the sequence's consistency, reduces ambiguity, and increases the percentage of correct steps. FSP (2 examples) is an advanced system because the two examples make the recurring patterns and constraints more explicit. However, the improvement over OSP is generally smaller than the large jump from ZSP to OSP. It can be noticed that once minimal guidance is provided, additional examples improve refinement rather than radically changing performance.

5.4 Analysis of the successes achieved

The third part of the experimental analysis has the objective not only to determine if a task has been completed, but above all to determine "when" a success is achieved within the "user on the loop" cycle. For each task, the LLM can receive up to three user feedback composed by refinement and validation. If the success is not achieved within these iterations, the execution is considered failed. This choice is intentional because beyond a certain threshold, even if the success could be reached by trying, the human cost increases too much in terms of time, attention, and the need to understand why the LLM is failing. In the diagrams:

- Success@0: immediate success with the input message of the user;

- Success@1: success achieved after an iteration of validation and refinement;
- Success@2: success achieved after two iterations of validation and refinement;
- Success@3: success achieved after three iterations of validation and refinement;
- Fail: the task does not converge within 3 iterations or converges beyond the threshold, but it is considered a failure anyway because of the human effort constraints;

Therefore, the goal is not to measure only the ability to achieve a correct execution of the task but the ability to do so with the least possible effort for the user.

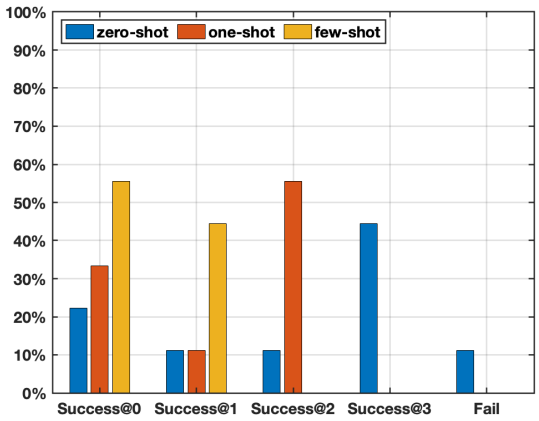

Figure 5.7: Analysis of the success with GPT-4.1

As can be seen in Figure 5.7, for what regards GPT-4.1:

- In the case of zero-shot, a significant portion of successes arrive late, many cases concentrated in Success@3, and also a portion of fail appears. This means that without examples in the prompt, the user is more often forced to interact multiple times to guide the model towards a correct solution;
- With one-shot, there are no fails anymore but a significant portion of cases converge after multiple iterations (Success@2). This shows that the model benefits from the example but often it still requires corrections;
- With few-shot, the distribution shifts toward early successes that are Success@0 and Success@1. The interaction becomes more and the human effort is reduced a lot because the user intervenes less and the model converges faster;

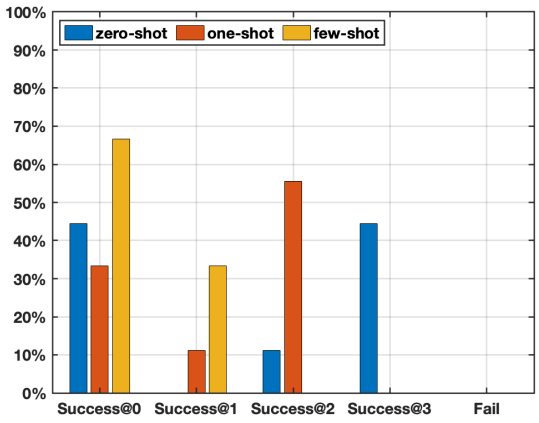

Figure 5.8: Analysis of the success with GPT-5

As can be seen in Figure 5.8, for what regards GPT-5:

- In zero-shot, there are no fails, but a marked presence of late successes is noted (Success@3). This indicates that the model is robust, but it does so at the cost of several correction cycles;
- In one-shot, the convergence is mainly concentrated in Success@0 and Success@2, so the example reduces uncertainty even if some tasks continue to require more steps of feedback to be aligned with the expected result;
- In few-shot most executions converge very early with Success@0 and Success@1, so the number of iterations required is reduced significantly;

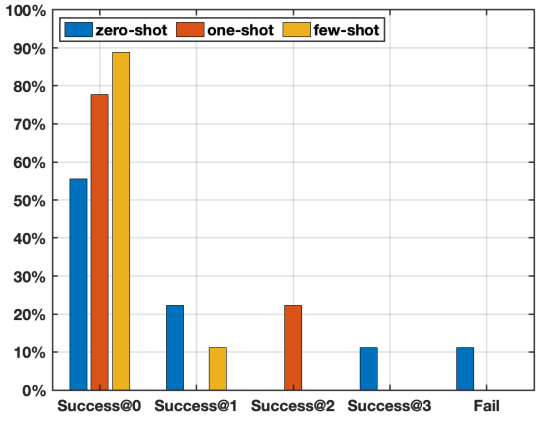

Figure 5.9: Analysis of the success with GPT-5.2

As can be seen in Figure 5.9, for what regards GPT-5.2:

- In zero-shot, a significant portion of successes occur immediately; in fact, there are many Success@0, but a proportion of Fails also appear. This suggests that the behavior is often very effective and fast but not always stable without guidance;
- In one-shot, the distribution shifts more toward immediate successes; in fact, there are many Success@0 and no fails are observed. The example provided to the prompt makes the behavior more consistent and reduces the need for iterations;
- In few-shot success is almost entirely concentrated in Success@0 and a very small portion in Success@1. This scenario remember the idea of effective collaboration because the user almost always obtains the correct solution with minimal intervention;

In general, from the comparison of the models, the important point that emerges is that GPT-5.2 is the one that best minimizes human effort because it tends to concentrate a higher proportion of successes on Success@0, or generally within few iterations, in particular when it is supported by one-shot and few-shot methods. In other words, GPT-5.2 demonstrates the best ability to collaborate with the user because it arrives in a quick manner to a correct solution, reducing the need for repeated corrections, even if some failures could be present. GPT-4.1 is more often associated with late successes like Success@2 and Success@3 and also presents failures under more difficult conditions, thus requiring more human intervention. GPT-5 is in an intermediate position because it is robust and never falls in failure, but in some cases it tends to require more iterations than GPT-5.2. The successes, in fact, are moved forward, for example, towards Success@3 in zero-shot methods with respect to GPT-5.2.

Chapter 6

Conclusions and Future Works

This thesis work focused on the study and implementation of a "user on the loop" approach for the execution of robotic tasks in which a large language model (LLM), after a text-based input message, generates a pseudocode written in Python and a non-technical user interacts with an iterative process of validation and refinement. The primary objective is not to make the system totally autonomous but rather to achieve an effective collaboration between the LLM and the user that allows tasks to be completed in a correct and safe way, minimizing as much as possible the effort required by the non-technical user.

Giving a look on the implementation, the first thing to notice is that the idea was to separate the technical identification (ID of the marker) from the semantic representation of an object or location through name and attributes. This means that observations obtained through ArUco markers are converted into entities with a certain meaning, like objects or locations, through catalogs structured in YAML files. In particular, the definition of catalogs and aliases allows for separating the ID of the marker from the information regarding the manipulation, like dimensions or gripping parameters, and positioning, like information about the internal structure and location compartmentalization. Furthermore, an important part of the system was dedicated to pose estimation, which is necessary to translate recognition into executable robotic actions like ordering objects, stacking objects or simple pick and place.

The experimental analyses were conducted on nine tasks that are ordered in an increasing order of difficulty, comparing three different models (GPT-4.1, GPT-5, GPT-5.2) and three prompting strategies (zero-shot, one-shot, few-shot). Performance was analyzed through two complementary metrics: (i) "Correct Steps" (%), which measures the percentage of correct steps in the pseudocode generated by

the model compared to a "golden code" which is a reference code; (ii) "Success@k", which highlights when success is achieved within a maximum of three (k = 1,2,3) refinement iterations. The results showed that:

- The introduction of examples in the baseline prompt produces a significant improvement in the quality of the pseudocode, with a huge gap in the transition from zero-shot to one-shot;

- The analysis through Success@k highlights that it is not important only to consider if a task is completed, but above all "when", so in how many iterations, since each additional iteration leads to an increase in the load of work and the time required by the non-technical user;

- The analyzed models have different strengths depending on the metric considered. In particular, GPT-5.2 results to be the most effective in reducing the effort required by the non-technical user, showing a greater tendency to achieve success early, around Success@0 and Success@1, and therefore to minimize the number of iterations of validation and refinement. At the same time, GPT-5 results in the model with the highest accuracy with respect to the "correct steps" (%) metric, obtaining the highest in average values for overall correctness of the steps and showing a generation of the pseudocode that is on average more coherent with the golden code;

In general, this thesis work demonstrates that a system that performs code generation, thanks to an LLM, can successfully support the execution of robotic tasks, provided that it measures and optimizes both the correctness of the generated solution and the speed of convergence to an acceptable solution through the help of a non-technical user.

6.1 Limitations and Future Works

Although the results obtained confirm the validity of the approach, there are different directions for development that could reinforce robustness.

6.1.1 Overcoming ArUco Markers Dependency

A fundamental limitation of the implementation explained in this thesis is represented by the use of the ArUco markers because they require that the tag be physically put on the object, reducing the capacity of generalizing real-world scenarios. A solution to this limitation is to replace with more advanced recognition techniques like object detection with bounding boxes or instance segmentation.

6.1.2 Study on a Large Sample of Non-Technical Users

In this thesis, the figure of the non-technical user was represented by the author that, being also the author of all of the core implementations, was the one who cared about debugging, even if this represents a limitation for drawing general conclusions about usability. An important future work is to extend the conducted analysis to a larger sample of people (e.g., $\sim$100 non-technical users) that enter in the "user on the loop approach", evaluating:

- Distribution of Success@k and failure rates;
- Quality and type of prompts or feedback provided by the non-technical users;
- Time required to achieve a task and workload;

6.1.3 Extension of the Set of Tasks and of the API Library

The nine tasks performed in this work represent a limited subset of the possible tasks that could be performed in real-world contexts. An evolution of the work could be to expand both the set of tasks used for experimental validation and the library of API that the LLM can use to build the pseudocode.

Appendix A

Configuring a Type-C to LAN Adapter for MacOS using Ubuntu

This guide refers to the use of the operating system developed by Apple, macOS. Among the prerequisites, the installation of UTM, a virtual machine emulator specifically designed for this operating system, is required. Once you have installed UTM, you will need to set up a virtual machine using Ubuntu. After creating the virtual machine, configure the network settings to use bridge mode (Figure A.1). In this mode, there is a direct connection between the Wi-Fi interface of the host system and the virtual machine, effectively creating a network bridge.

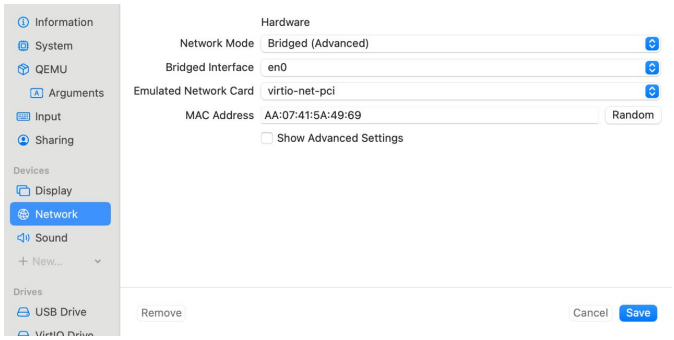

Figure A.1: Network settings interface

A.1 Steps

1. Start the virtual machine.

2. To ensure that the cable is correctly recognized by the Linux operating system, it is essential to connect the 'Type-C to LAN' adapter directly to the virtual machine. To do so, click on the icon highlighted in red at the top-right corner of the Figure A.2. From the drop-down menu, select the name of the adapter (e.g. USB 10/100/1000 LAN).

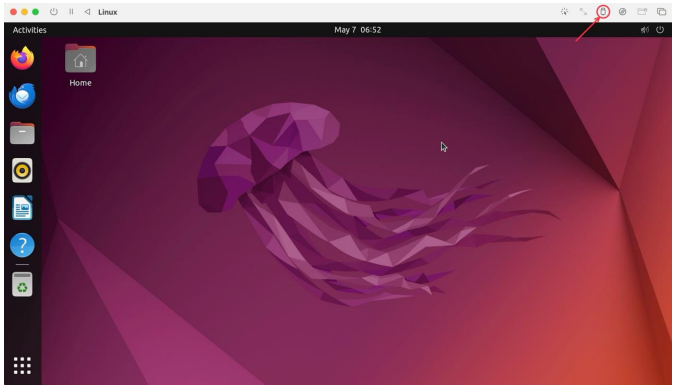

Figure A.2: Icon needed to select for the adapter reading

3. Once the adapter has been successfully connected and recognized by the system, open the terminal and enter the following command: `"ip link show"`. This command will display a list of the available network interfaces (Figure A.3). At least three interfaces should appear, including: LOOPBACK, BRIDGE and ADAPTER. Please note that the interface corresponding to the connected adapter typically begins with the prefix `"enx"`.

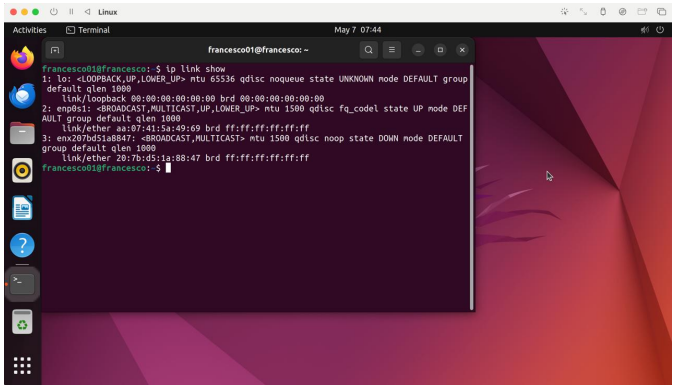

Figure A.3: Interfaces on the terminal

4. To activate the network interface, execute the following command in the terminal: `sudo ip link set <interface_name> up`. Replace `<interfa-`

`ce_name>` with the actual name of the adapter interface (e.g. `enx...`) as identified in the previous step.

5. An IP address compatible with the robot must be assigned to the network interface. It is crucial that this address is not the same as the robot's IP address. To do this, let's use the following command: `"sudo ip addr add <IP_address> <netmask> dev <interface_name>"`. Replace `<IP_address>`, `<netmask>` and `<interface_name>` with the appropriate values. For example, in the case of the xArm 6 by UFACTORY, the PC's IPv4 network segment must fall within the range 192.168.1.1 to 192.168.1.255, ensuring that the end number of IP address shall be different from that of the Control Box (e.g. 192.168.1.2 in Figure A.4). The `<netsmak>` in this case is the following: 255.255.255.0 (Figure A.4) and the `<interface_name>` is the same of the previous step (e.g. `enx...`).

6. At this stage, the configured network adapter should now appear among the list of network interfaces. To verify this, use the following command, as in Figure A.4, in the terminal: `"ifconfig"`. Check the output to confirm that the adapter is listed and correctly configured.

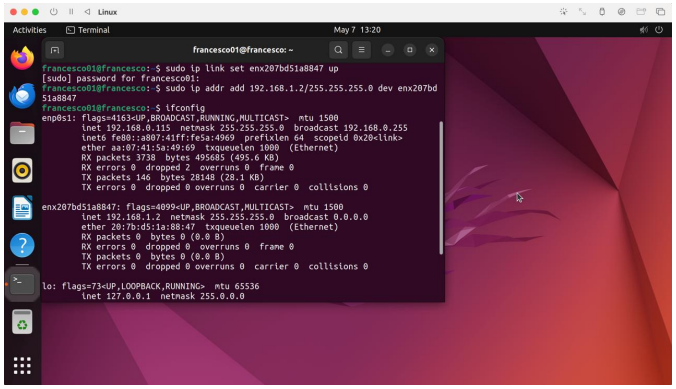

Figure A.4: Full view of the terminal

A.2 Setting Up the ROS 2 Environment Automatically

Each time the virtual machine is started, trying to run the `"ros2"` command in the terminal results in an error that is: `"command not found"`. This happens because the ROS 2 environment has not yet been set up. To manually configure it, it's necessary to execute the following command: `"source /opt/ros/humble/setup.bash"`. However, performing this step repeatedly can become boring.

To avoid this, a more efficient method is to configure the environment setup to occur automatically. First, open the terminal and enter the following command: `"gedit ~/.bashrc"`. In this context: `"gedit"` represents the home directory, while `".bashrc"` is a system configuration file that sets up the starting environment. After running this command, a graphical interface will open, as shown in Figure A.5.

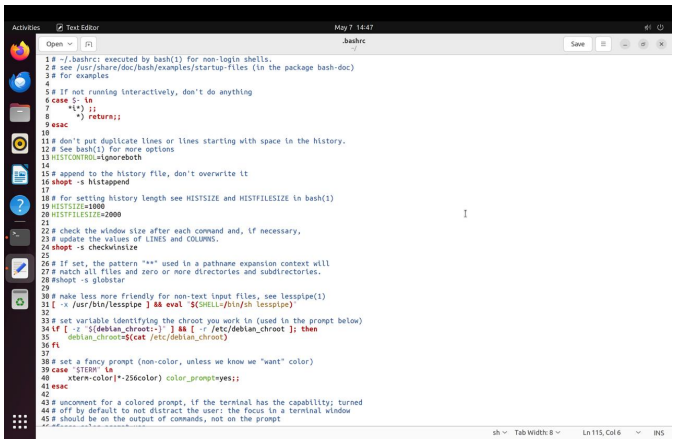

Figure A.5: System configuration file "`.bashrc`"

Now it is necessary to add a new line of code at the end of the `".bashrc"` file, that is: `"source /opt/ros/humble/setup.bash"`. Once this line has been added, save the changes, close the file and restart the terminal. It is possible to notice that, by entering the command `"ros2"`, the issue is resolved, as demonstrated in Figure A.6.

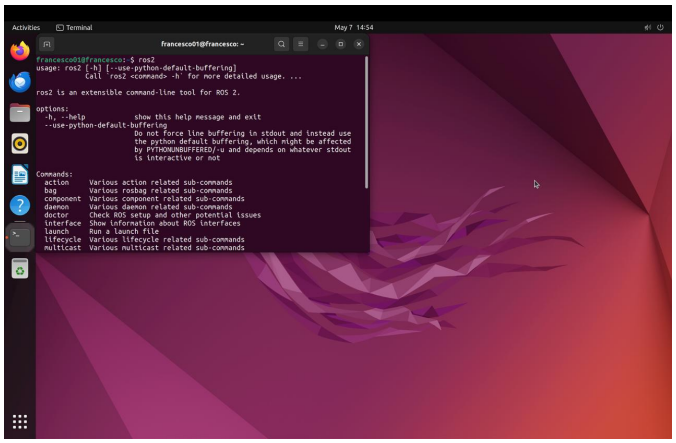

Figure A.6: Command "`ros2`" on the terminal

Appendix B

Intel RealSense™ D435i

B.1 Intel RealSense D400 Series

Figure B.1: RealSense™ Depth Camera D435i [38]

The RealSense™ D400 family is a stereo vision depth camera system. Each subsystem combines a stereo depth module with a vision processor and there is a connection to a host processor via USB 2.0, USB 3.1 Gen 1 or MIPI[1]. Thanks to its compact form and ease of integration, the camera subsystems offer flexibility for the system integrators in order to design a wide range of products. The series offer also complete depth cameras that integrate the vision processor, stereo depth module, an RGB sensor with signal processing of color image and an IMU[2]. The D400 series is supported by the open-source platform RealSense™ SDK 2.0. [38]

B.2 Key Charateristics of Intel RealSense™ D435i [38]

The main features of the RealSense™ Depth Camera D435i are the following:

[1]MIPI (Mobile Industry Processor Interface) is an organization that defines interface standards in the field of mobile devices.

[2]IMU (Inertial Measurement Unit) is an electronic device that measures the specific force of a body, the angular rate and, sometimes, the magnetic field around the body.

- RealSense™ Vision Processor D4;
- Stereo depth resolution up to 1280 x 720;
- RGB resolution up to 1920 x 1080;
- Depth diagonal field of view exceeding 90°;
- Dual global-shutter sensors enabling depth streaming at up to 90 FPS;
- Operation range from 0.2 m to beyond 3 m but subjected to lighting conditions;
- Integrated IMU for 6 Degree of Freedom (6 DoF) data;

Bibliography

[1] Balasubramaniyan Chandrasekaran and James M. Conrad. «Human-robot collaboration: A survey». In: *SoutheastCon 2015*. 2015, pp. 1–8. DOI: 10.1109/SECON.2015.7132964 (cit. on p. 1).

[2] Alessio Baratta, Antonio Cimino, Maria Grazia Gnoni, and Francesco Longo. «Human robot collaboration in industry 4.0: a literature review». In: *Procedia Computer Science* 217 (2023), pp. 1887–1895 (cit. on p. 1).

[3] Muhammad Umar Farooq, Geon Kang, Jiwon Seo, Jungchan Bae, Seoyeon Kang, and Young Jae Jang. «DAIM-HRI: A new Human-Robot Integration Technology for Industries». In: *2024 IEEE International Conference on Advanced Robotics and Its Social Impacts (ARSO)*. IEEE. 2024, pp. 7–12 (cit. on pp. 1, 2).

[4] Sai H Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. «Chatgpt for robotics: Design principles and model abilities». In: *Ieee Access* 12 (2024), pp. 55682–55696 (cit. on pp. 2, 13, 36).

[5] Yang Ye, Hengxu You, and Jing Du. «Improved trust in human-robot collaboration with ChatGPT». In: *Ieee Access* 11 (2023), pp. 55748–55754 (cit. on p. 2).

[6] Fanlong Zeng, Wensheng Gan, Yongheng Wang, Ning Liu, and Philip S Yu. «Large language models for robotics: A survey». In: *arXiv preprint arXiv:2311.07226* (2023) (cit. on pp. 2, 7–9).

[7] Christopher E Mower et al. «Ros-llm: A ros framework for embodied ai with task feedback and structured reasoning». In: *arXiv preprint arXiv:2406.19741* (2024) (cit. on p. 2).

[8] Yixiang Jin, Dingzhe Li, Jun Shi, Peng Hao, Fuchun Sun, Jianwei Zhang, Bin Fang, et al. «Robotgpt: Robot manipulation learning from chatgpt». In: *IEEE Robotics and Automation Letters* 9.3 (2024), pp. 2543–2550 (cit. on p. 2).

[9] Juo-Tung Chen and Chien-Ming Huang. «Forgetful large language models: Lessons learned from using llms in robot programming». In: *Proceedings of the AAAI Symposium Series*. Vol. 2. 1. 2023, pp. 508–513 (cit. on p. 2).

[10] IBM. *Natural Language Processing*. https://www.ibm.com/it-it/think/topics/natural-language-processing. Accessed: 2025-09-25. 2025 (cit. on p. 5).

[11] IBM. *Natural Language Understanding*. https://www.ibm.com/think/topics/natural-language-understanding. Accessed: 2025-09-25. 2025 (cit. on p. 6).

[12] GeeksforGeeks. *What is a Large Language Model (LLM)*. https://www.geeksforgeeks.org/artificial-intelligence/large-language-model-llm/. Accessed: 2025-09-25. 2025 (cit. on p. 6).

[13] IBM. *Large Language Models*. https://www.ibm.com/it-it/think/topics/large-language-models. Accessed: 2025-09-25. 2025 (cit. on pp. 6, 7).

[14] IBM. *Self-Supervised Learning*. https://www.ibm.com/think/topics/self-supervised-learning. Accessed: 2025-09-25. 2025 (cit. on p. 7).

[15] IBM. *Zero-Shot Learning*. https://www.ibm.com/think/topics/zero-shot-learning. Accessed: 2025-09-25. 2025 (cit. on p. 7).

[16] IBM. *What Is Few-Shot Learning?* https://www.ibm.com/think/topics/few-shot-learning. Accessed: 2025-09-26. 2025 (cit. on p. 7).

[17] Anis Koubaa, Adel Ammar, and Wadii Boulila. «Next-generation human-robot interaction with ChatGPT and robot operating system». In: *Software: Practice and Experience* 55.2 (2025), pp. 355–382 (cit. on p. 7).

[18] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. «An empirical study of the non-determinism of chatgpt in code generation». In: *ACM Transactions on Software Engineering and Methodology* 34.2 (2025), pp. 1–28 (cit. on p. 10).

[19] Satish Kathiriya, Mahidhar Mullapudi, and Abhishek Shende. «The Power of Prompt Engineering: Refining Human-AI Interaction with Large Language Models in The Field of Engineering». In: *International Journal of Science and Research (IJSR)* 12.11 (2023), pp. 2319–7064 (cit. on pp. 10, 12).

[20] GitHub. *What is prompt engineering?* https://github.com/resources/articles/what-is-prompt-engineering. Published: Dec. 19, 2025. Accessed: Jan. 11, 2026. 2025 (cit. on pp. 11, 12).

[21] Smart Robotics. *What Is a Pick & Place Robot? Types, Benefits, and Uses*. https://smart-robotics.io/what-is-a-pick-and-place-robot/. Accessed: 2026-01-20. 2025 (cit. on pp. 14, 15, 17).

[22] Tecno4Industry. *Robot Pick and Place: cosa sono, le tipologie e i vantaggi*. https://tecno4industry.it/robot-pick-and-place/. Accessed: 2026-01-20. 2022 (cit. on p. 14).

[23] Ocado Intelligent Automation. *What is a Pick and Place Robot? Uses and Types*. https://ocadointelligentautomation.com/insights/what-is-a-pick-and-place-robot/. Accessed: 2026-01-20. 2019 (cit. on p. 16).

[24] RoboDK. *What Is a SCARA Robot? The Background and Benefits*. https://robodk.com/blog/what-is-a-scara-robot/. Accessed: 2026-01-20. 2023 (cit. on p. 17).

[25] Mark Moll, Ioan A Sucan, and Lydia E Kavraki. «Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization». In: *IEEE Robotics & Automation Magazine* 22.3 (2015), pp. 96–102 (cit. on p. 18).

[26] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. «The open motion planning library». In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82 (cit. on pp. 18, 19).

[27] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2008. ISBN: 9781846286421. URL: https://books.google.it/books?id=VsTOQOnQjCAC (cit. on p. 20).

[28] Aqeel Anwar. *What are Intrinsic and Extrinsic Camera Parameters in Computer Vision? Towards Data Science*. Accessed: 2026-01-17. 2022. URL: https://towardsdatascience.com/what-are-intrinsic-and-extrinsic-camera-parameters-in-computer-vision-7071b72fb8ec (cit. on pp. 20, 21).

[29] Oleksandr Semeniuta. «Analysis of camera calibration with respect to measurement accuracy». In: *Procedia Cirp* 41 (2016), pp. 765–770 (cit. on pp. 21–23).

[30] OpenCV Development Team. *Detection of ArUco Markers*. https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Consulted on January 22, 2026. 2025 (cit. on pp. 23, 24).

[31] *UFACTORY xArm User Manual (Translated Version)*. V2.0.0. Accessed: 2026-01-30. Shenzhen UFactory Co., Ltd. URL: https://www.ufactory.cc/wp-content/uploads/2023/05/xArm-User-Manual-V2.0.0.pdf (cit. on pp. 26, 28).

[32] *ROS – Robot Operating System*. https://www.ros.org/. Accessed: 2026-01-29 (cit. on pp. 28, 30).

[33] *Basic Concepts — ROS 2 Documentation: Jazzy*. https://docs.ros.org/en/jazzy/Concepts/Basic.html. Accessed: 2026-01-29 (cit. on p. 30).

[34] PickNik Robotics. *MoveIt 2 Documentation (Rolling)*. https://moveit.picknik.ai/main/index.html. Accessed: 2026-01-31. 2025 (cit. on p. 30).

[35] PickNik Robotics. *MoveIt 2 Documentation (Rolling) — Concepts*. https://moveit.picknik.ai/main/doc/concepts/concepts.html. Accessed: 2026-01-31. 2025 (cit. on pp. 30–33).

[36] Kavraki Lab. *OMPL Available Planners*. https://ompl.kavrakilab.org/planners.html. Accessed: 2026-02-01. 2026 (cit. on p. 34).

[37] xArm-Developer. *xarm_ros2 (jazzy branch): ROS 2 developer packages for robotic products from UFACTORY*. GitHub repository. Accessed: 2026-02-01. URL: https://github.com/xArm-Developer/xarm_ros2/tree/jazzy (cit. on p. 35).

[38] Intel RealSense. *Intel RealSense D400 Series Datasheet*. Accessed: 2025-09-24. Aug. 2025. URL: https://realsenseai.com/wp-content/uploads/2025/08/Intel-RealSense-D400-Series-Datasheet-August-2025.pdf (cit. on p. 66).