One to rule them all: natural language to bind communication, perception and action

Simone Colombani[1,3], Dimitri Ognibene[2] and Giuseppe Boccignone[1]

[2]*University of Milan, Italy*
[1]*University of Milano-Bicocca, Milan, Italy*
[3]*Oversonic Robotics, Carate Brianza, Italy*

Abstract

In recent years, research in the area of human-robot interaction has focused on developing robots capable of understanding complex human instructions and performing tasks in dynamic and diverse environments. These systems have a wide range of applications, from personal assistance to industrial robotics, emphasizing the importance of robots interacting flexibly, naturally and safely with humans.

This paper presents an advanced architecture for robotic action planning that integrates communication, perception, and planning with Large Language Models (LLMs). Our system is designed to translate commands expressed in natural language into executable robot actions, incorporating environmental information and dynamically updating plans based on real-time feedback.

The Planner Module is the core of the system where LLMs embedded in a modified ReAct framework are employed to interpret and carry out user commands like *'Go to the kitchen and pick up the blue bottle on the table'*. By leveraging their extensive pre-trained knowledge, LLMs can effectively process user requests without the need to introduce new knowledge on the changing environment. The modified ReAct framework further enhances the execution space by providing real-time environmental perception and the outcomes of physical actions. By combining robust and dynamic semantic map representations as graphs with control components and failure explanations, this architecture enhances a robot's adaptability, task execution efficiency, and seamless collaboration with human users in shared and dynamic environments. Through the integration of continuous feedback loops with the environment the system can dynamically adjusts the plan to accommodate unexpected changes, optimizing the robot's ability to perform tasks. Using a dataset of previous experience is possible to provide detailed feedback about the failure. Updating the LLMs context of the next iteration with suggestion on how to overcame the issue.

This system has been implemented on RoBee, the cognitive humanoid robot developed by Oversonic Robotics, showcasing its adaptability and potential for integration across diverse environments. By leveraging LLMs and semantic mapping, the architecture enables RoBee to navigate and respond to real-time changes.

Keywords
Human-Robot interaction, Robot task planning, Large Language Models, Automated planning

1. Introduction

The integration of LLMs in robotic systems has opened new avenues for autonomous task planning and execution [2, 3]. These models demonstrate exceptional natural language understanding and commonsense reasoning capabilities, enhancing a robot's ability to comprehend contexts and execute commands [4, 5]. However LLMs are not be able to plan autonomously, they need to be integrated in architectures that enable them to understand the environment, the robot capabilities and state [6]. This research aims to empower robots to comprehend user requests and autonomously generate actionable plans in diverse environments.

However, the efficacy of these plans relies on the robot's understanding of its operating environment [7]. To bridge this gap, our work employs scene graphs [8] as a semantic mapping tool, offering a structured representation of spatial and semantic information within a scene.
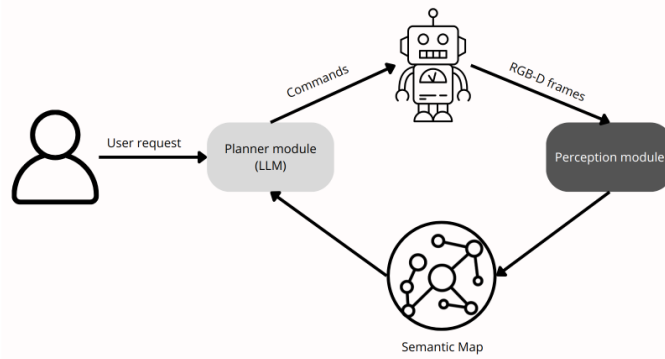
Figure 1: Architecture of the system.

In our approach, we leverage LLMs through in-context [9], which enables the models to learn and adapt based on the information provided in the context. Our work implements a modified version of the ReAct [10] framework that expand the context of LLMs with environmental information and execution feedback, allowing the model to plan and execute skills [11] translating them into physical actions.

Motivation   The primary focus of our work is to enable robot to interact flexibly and robustly in dynamic and diverse environments with limited human intervention. Traditional robotic systems usually rely on static, pre-programmed instructions or closed world predefined knowledge and settings, limiting their adapt-ability to dynamic environments. Interacting with humans in daily tasks within complex environments disrupts these assumptions. LLMs and VLM can provide open-domain knowledge to represent novel conditions without human intervention. However, these models are not informed of the trained specific robot, task and settings at hand, that define what information can be relevant and necessary to find and reason about [12]. Exceeding in the level of detail may lead to impractical computational requirements and response time. Discarding crucial information, spatial or semantic, may lead to repeated failures due to the introduced non-managed partial observability [13]. To find the relevant information may be too slow [14]. LLMs can still produce outputs that are logically inconsistent or impractical [15], expecially if they are not integrated into systems that allow them to adapt to changes in the environment and the physical capabilities of the robots. During task execution, robots may encounter unexpected situations, such as unanticipated obstacles, sensor errors, or changes in the environment that were not accounted for in the initial plan. Such scenarios necessitate robust error handling mechanisms and adaptive planning strategies that enable the system to reassess and modify its actions in real-time [16]. By introducing execution controlling and failure management into the planning process at different levels as well as retrieval of previous successful plans, we propose a solution to enhance the robustness and flexibility of LLM-based robotic systems. This approach ensures that the robot can effectively perceive changes in the environment and the failures that may arise from them, allowing it to adapt strategies in response to new challenges.

Proposed approach   Our system addresses the challenges of dynamic environments through a real-time perception module and a Planner module that integrates execution control, and failure management. It comprise a Controller that monitors the execution of tasks and detects errors, while the Explainer analyzes failures and suggests adjustments based on past experiences. This feedback loop enables adaptive re-planning, allowing the system to modify its actions as needed. Specifically, we propose the use of the ReAct [10] framework, expanding its operational space with skills, physical actions of the robot. By leveraging LLMs for natural language understanding and a perception system, the architecture supports autonomous task execution in dyanamic scenarios.

2. Related works

A substantial body of literature explores the utilization of LLMs for robotic task planning [4, 5].

LLM for robot planning   Recent works highlight the potential of Large Language Models (LLMs) in robotic planning [17, 18, 19]. DEPS [20] introduces an iterative planning approach for agents in open-world environments, such as Minecraft. It utilizes LLMs to analyze errors during execution and refine plans, improving both reasoning and goal selection processes. However, this approach has been primarily developed and tested in virtual environments, with notable differences in comparison to real-world settings due to the dynamic and unpredictable nature of physical environments. Additionally, DEPS does not leverage previous issues and solutions but relies solely on feedback from humans and vision-language models (VLMs).

Scene graph as environemental representation   The use of scene graphs [21] as a means to represent the robot's environment has gained traction. [22] employs 3D scene graphs to represent environments and uses LLMs to generate Planning Domain Definition Language (PDDL) files. This method decomposes long-term goals into natural language instructions and enhances computational efficiency by addressing sub-goals. However, it lacks a mechanism for replanning based on feedback during execution, which could limit its adaptability in dynamic scenarios. SayPlan [23] integrates semantic search with scene graphs and path planning to aid robots in navigating complex environments through natural language. By combining these techniques, SayPlan simulates various scenarios to refine task sequences, which helps improve overall task performance in complex environments.

Replanning   Replanning enables long-term autonomous task execution in robotics [24]. DROC [25] empowers robots to process natural language corrections and generalize that information to new tasks. It introduces a mechanism to distinguish between high-level and low-level errors, allowing more flexible plan corrections. However, DROC does not address the types of failures that may occur during plan execution, focusing instead on high-level corrections provided by users. [26] supports autonomous long-term task execution by integrating LLMs for planning and VLMs for feedback. This approach adapts to changes in the environment through a structured component system that verifies and corrects plans as needed. Yet, the feedback is limited to what is visible to the robot's camera, potentially overlooking other significant environmental changes.

3. Architecture

Our system is based on two components:

- **Perception Module**: it is responsible for sensing and interpreting the environment. It builds and mantains a semantic map in the form of a directed graph that integrates both geometric and semantic information.
- **Planner Module**: it takes the information provided by the Perception Module to formulate plans and actions that allow the robot to perform specific tasks.

Figure 1 show how these components interact to allow the robot to understand its environment and act accordingly to satisfy user requests. The Perception module uses data provided by the robot's sensors to supply the semantic map to the Planner module, which in turn processes it to generate specific action plans. In what follows we precisely address the Planner Module while details on the Perception Module will be provided in a separate article.
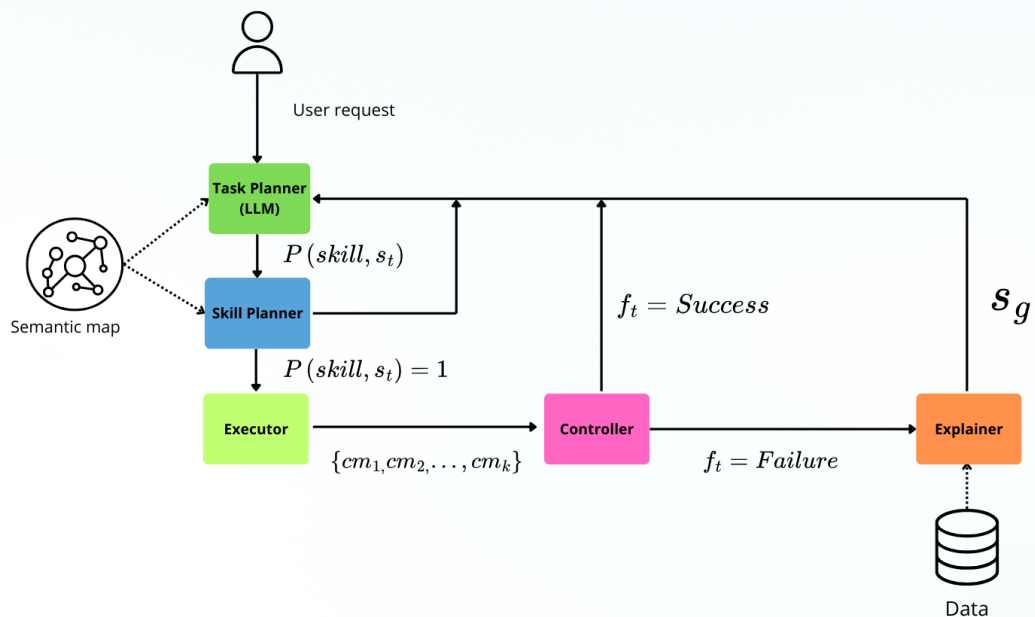
Figure 2: Architecture of the planner module.

3.1. Planner module

The architecture of the Planner module is designed to translate user requests, expressed in natural language, into specific actions executable by a robot. This module is responsible for understanding instructions, planning appropriate actions, and managing the execution of those actions in a dynamic environment. The Planning module is composed by five sub-modules:

- **Task Planner**: Translates user requests, expressed in natural language, into a sequence of high-level skills.
- **Skill Planner**: Translates high-level skills into specific, low-level executable commands.
- **Executor**: Executes the low-level actions generated by the Skill Planner.
- **Controller**: Monitors the execution of actions and manages any errors or unexpected events during the process.
- **Explainer**: Interprets the causes of execution failures by analyzing data received from the Controller and provides suggestions to the Task Planner on how to adjust the plan.

The architecture of the planner module is shown in Figure 2. The main component of the system is the Task Planner, which receives the user's request and translates it into a list of high-level "skills" that represent the robot's capabilities. These skills include actions such as "PICK" (grasp an object), "PLACE" (place an object), and "GOTO" (move to a position).

3.1.1. Task Planner

The decision-making process of the Task Planner is driven by a **policy**, which is implemented as a LLM. A policy is a strategy or rule that defines how actions are selected based on the current state or

context,[27].

Task Planner is implemented using the ReAct framework [10], which alternates between reasoning and action phases during the process. In the reasoning phase, the Task Planner can access various **"perception" actions** to gather information from the environment, such as the semantic map and the current state of the robot, and can execute one or more **"skill" actions** to perform physical actions. The classical idea of ReAct is to augment the agent's action space to $\hat{A} = A \cup L$, where L is the space of language-based reasoning actions. An action $\hat{a}_t \in L$, referred to as a *"thought"* or reasoning trace, does not directly affect the external environment but instead updates the current context $c_{t+1} = (c_t, \hat{a}_t)$ by adding useful information to support future decision-making [10]. In the classical idea there could be various types of useful thoughts, such as decomposing task goals and creating action plans, injecting commonsense knowledge relevant to task solving, extracting important parts from observations, tracking progress and transitioning action plans, handling exceptions and adjusting action plans, and so on, but always without modifying the physical environment, only embedding it within the context. Interestingly, this approach mixes reasoning and action in a flexible manner. In the future, we will analyse the potential of this approach also connecting to the planning-to-plan [28, 29] and meta-reasoning [30, 31, 32] concepts.

In our work, we augment the agent's [33] action space with two types of actions:

- A *skill* action $a_t \in A_{\text{skill}}$, which involves physically interacting with the environment, such as manipulating objects or navigating. The result of a skill action provides new feedback that updates the current context.
- A *perception* action $a_t \in A_{\text{perception}}$, which involves accessing information from the environment, such as querying the semantic map or sensors, and integrating that information into the context.

The augmented action space is defined as:

$$\hat{A} = A_{\text{skill}} \cup A_{\text{perception}} \cup L$$

Thus, the LLM serves as the **policy** π that selects different types of actions from the augmented action space and dynamically adapting the current context c_t used to plan based on real-time information and reasoning.

Formal Description: The Task Planner's policy π, represented by the LLM, can be formalized as a function that maps the current context c_t to an action $\hat{a}_t$ from the augmented action space $\hat{A}$:

$$\pi : C \rightarrow \hat{A}, \quad \pi(c_t) = \hat{a}_t$$

Where:

- C is the set of all possible contexts.
- $\hat{A}$ is the augmented action space $\hat{A} = A_{\text{skill}} \cup A_{\text{perception}} \cup L$.
- c_t represents the current context at time t, which includes the state of the robot, the environment, and any past actions or thoughts.
- $\hat{a}_t \in \hat{A}$ is the action chosen by the policy, which can be a skill action $a_t \in A_{\text{skill}}$, a perception action $a_t \in A_{\text{perception}}$, or a reasoning trace $\hat{a}_t \in L$.

The context c_t is updated based on the chosen action:

- If $\hat{a}_t \in L$ (a reasoning action), the context updates to:

$$c_{t+1} = (c_t, \hat{a}_t)$$

This represents the thought process, where reasoning contributes new information without affecting the external environment.

- If $\hat{a}_t \in A_{\text{perception}}$ (a perception action), the result of querying the environment updates the context:

$$c_{t+1} = (c_t, f_{\text{perception}}(\hat{a}_t))$$

Here, $f_{\text{perception}}$ represents the function that gathers information and modifies the context based on the perception action's outcome.

- If $\hat{a}_t \in A_{\text{skill}}$ (a skill action), the robot interacts with the environment, and the context updates based on feedback from the physical action:

$$c_{t+1} = (c_t, f_{\text{skill}}(\hat{a}_t))$$

Where f_{skill} is the function that captures the result of executing a physical skill, such as manipulating an object or moving to a location.

3.1.2. Skill Planner

Once a high-level request for the execution of a skill is made, the Skill Planner is responsible for translating the high-level skills, provided by the Task Planner, into sequences of low-level commands executable by the robot. While the Task Planner focuses on understanding natural language and creating a general plan, the Skill Planner deals with the specific details of how each skill should be executed, considering the robot's state and the environment.

Let a skill be represented in the following general form, defined by the Task Planner with specific syntax:

$$SKILL\_NAME(param_1, param_2, \dots, param_N)$$

Where:

- SKILL_NAME is the name of the skill to be executed (e.g., PICK, PLACE, GOTO).
- param_1, param_2, …, param_N are parameters for the skill, such as the object to manipulate or the destination to navigate to.

Using a strict syntax ensures that the Skill Planner can correctly interpret the high-level commands without ambiguity. For instance, a natural language command like *"Move near the table and grab the bottle"* would lack precision. The Skill Planner needs concrete parameters for the robot to act effectively.

Skill Planner workflow: The Skill Planner operates by performing three functions:

1. **Precondition Verification:** Before translating a skill into low-level commands, the Skill Planner verifies that the necessary preconditions for execution are met. Let s_t represent the current state of the robot and the environment at time t, and $P(\text{skill}, s_t)$ denote a function for every skill that evaluates the preconditions for a given skill. The precondition check can be expressed as:

$$P(\text{skill}, s_t) = \begin{cases} 1, & \text{if all preconditions are met} \\ 0, & \text{otherwise} \end{cases}$$

For example, before executing the PICK skill, the following checks may be performed:

- The object is visible by the robot.
- The object is reachable for the robotic arm.
- The robotic arm is free.

If any of these conditions are not met ($P(\text{skill}, s_t) = 0$), the Skill Planner reports a failure to the Task Planner.

2. **Target nodes extraction:** Based on the parameters of the skill, the Skill Planner extracts the target nodes from the semantic map $\mathcal{M}$, which contains geometric and semantic information about

the environment. Every node provides geometric information such as object's position and relevant context, which is then used to generate low-level commands.

3. **Generation of Low-Level Commands:** When $P(\text{skill}, s_t) = 1$, the Skill Planner translate the skill into a sequence of low-level commands to control the robot behavior. In this system, we represent skill decomposition in commands as Hierarchical Task Networks (HTNs) that contains low-level commands executable by the robot. Let $CM(\text{skill}, \text{node}, s_t)$ denote the function that translates the given skill into low-level commands based on the target nodes extracted from the semantic map and current state. The output is a sequence of pre-modeled commands parameterized with the information of the robot state and the target nodes, $\{cm_1, cm_2, \ldots, cm_k\}$, where each command cm_i directs specific components of the robot. Our implementation use HTNs solely on the breakdown of skills into commands without using them with advanced features like re-planning or error recovery of the commands. In this case, if any command fails, the entire skill fails, with no attempt at re-planning at the skill planner level. The process can be represented as:

$$\{cm_1, cm_2, \ldots, cm_k\} = CM(\text{skill}, \text{node}, s_t)$$

The Skill Planner is designed to be flexible and extendable. The skill functions P, and CM can be adapted or extended to accommodate new skills, hardware, or environments.

3.1.3. Executor

The Executor is responsible for directly interacting with the robot's hardware to execute the commands provided by the Skill Planner. It translates the low-level commands into physical actions by controlling various hardware elements such as motors, robotic arm grippers, and other actuators required for task execution.

Let the set of low-level commands generated by the Skill Planner be represented as above, i.e., $cm_1, cm_2, \ldots, cm_k\} = CM(\text{skill}, \text{node}, s_t)$, where $CM(\text{skill}, \text{node}, s_t)$ defines the sequence of commands based on the skill, the target node, and the current state of the robot and the environment.

The Executor is tasked with executing these commands on the physical robot. Let the state of the robot at time t be denoted by h_t, and the function that maps a low-level command c_i to an effect on the robot's state be denoted as $H(cm_i, h_t)$. The execution of a command at time t can be described as:

$$h_{t+1} = H(cm_i, h_t)$$

where h_{t+1} is the updated state after executing the command cm_i. This process is repeated for each command in the sequence $\{cm_1, cm_2, \ldots, cm_k\}$ until the entire skill is executed.

Executor workflow:

- **Command reception:** The Executor receives a set of low-level commands $\{cm_1, cm_2, \ldots, cm_k\}$ from the Skill Planner. Each command specifies a concrete action to be performed by the robot's hardware components.
- **Hardware interaction:** For each command cm_i, the Executor interacts with the robot's hardware, adjusting the motors, grippers, and other actuators. This interaction can be represented by the function $H(cm_i, h_t)$ that determines the effect of a command on the robot's state h_t.
- **Command execution:** The Executor executes each command cm_i in the sequence, ensuring that the robot's state transitions from state h_t to h_{t+1}. Formally:

$$h\_t + 1 = H(cm\_i, h\_t), \quad \forall i = 1, 2, \ldots, k$$

After executing all commands, the robot's reaches the final state h_{t+k}, corresponding to the completion of the skill.

- **Real-Time feedback:** During execution, the robot's provides feedback on its current state. Let f_t denote the feedback at time t, and f_{t+1} be the updated feedback after executing command cm_i:

$$f_{t+1} = F(cm_i, h_t)$$

where F is the feedback function. If unexpected feedback f_{t+1} is received, the Executor can trigger adjustments to the plan or inform the Skill Planner of a potential issue.

Different robots may use different communication protocols, and hardware configurations. Therefore, the Executor must be adapted for each specific robot system, ensuring that it correctly interacts with the robot's hardware.

3.1.4. Controller

The Controller is responsible for monitoring the robot's status and the environment during command execution, ensuring that they are carried out as planned. After each command is executed, the Executor sends feedback indicating either success or failure. If a failure occurs, it results in the failure of the entire skill. Upon the completion of all commands, a success feedback will indicate the successful execution of the skill.

Denote f_t the feedback from the Executor at time t. The Controller processes f_t to determine the outcome of the executed skills. The feedback can be classified into two categories: success and failure.

Feedback processing:

- **Success:** If the feedback f_t indicates successful execution of a command and it is the last command to execute, the skill is considered successfully completed, the Controller sends a positive acknowledgment to the Task Planner to continue the planning process. However, if the feedback indicates success but the command is not the last one, the Controller waits for the execution of the next command:

$$\text{if } f_t = \text{Success} \implies \text{Task Planner continues}$$

- **Failure:** If a failure occurs during the execution of any command, the planned skill fails and the Controller generates a failure message m_f that includes the reason for the failure. This message is sent to the Explainer. Let e_t represent the specific error detected at time t. The failure message can be represented as:

$$m_f = \text{Failure}(e_t)$$

where e_t can include various error reasons such as obstacles detected, non-executable trajectories, or environmental changes.

The Controller's operation is highly dependent on the specific robot system in use, as it relies on the characteristics of the robot and the employed software system. In a ROS environment, for example, the Controller interacts with ROS nodes that control the robot's hardware. In our work, RoBee, described in section 5, has a system that allows to obtain feedback on the execution of commands.

3.1.5. Explainer

The Explainer component plays a critical role in enhancing the planning process by providing insights to the Task Planner when failures occur during the execution phase. After receiving the failure reason, the Explainer searches a dataset $\mathscr{D}$ for previous instances of similar failures. This dataset comprises records of failures associated with specific skills and user requests. Let $\mathscr{D}_{r_f}$ denote the subset of the dataset containing records of failures and solutions related to the same skill and error message. The dataset has been manually built based on previous experiences, desired behaviors, and expected failures.

The search can be expressed as:

$$\mathscr{D}_{r_f} = \{(s_k, u_r, r_f) \in \mathscr{D} \mid s_k = \text{skill\_name}, e_r = r_f, u_r \sim \text{user\_request}\}$$

where:

- s_k is the skill being executed (e.g., PICK).
- u_r represents the specific user request associated with the failure.
- r_f is the failure reason provided by the Controller
- $u_r \sim$ user_request indicates that the user request in the dataset is similar to the current user request.

Rather than searching for an exact match to the user's request, the Explainer assesses the similarity of the user's request (u_r) to the instances in the dataset linked to the suggestion, using cosine similarity in our approach [34]. This method enables the system to identify the most relevant past instances, even when the user's requests are not identical.

Once relevant instances are identified, the Explainer analyzes these cases to generate a suggestion s_g for the Task Planner on how to proceed. The suggestion is structured as follows:

$$s_g = \text{Suggest}(\mathcal{D}_{r_f})$$

For instance, if the Controller reports the failure reason:

$$r_f = \text{"Cannot execute the approach movement for the PICK skill, object too far"}$$

The Explainer analyzes this failure and may find a previous instance where the robot successfully resolved a similar issue. It could recommend a command to the Task Planner:

$$s_g = \text{"Use the GOTO skill to move near the object to pick"}$$

This suggestion enables the Task Planner to adjust its strategy effectively, moving the robot closer to the object before attempting the PICK action again.

The suggestions provided by the Explainer can be tailored to accommodate specific behaviors of the robot. This adaptability can be achieved by modifying the parameters of the data utilized to generate the suggestions. Thus, the Explainer enhances the resilience of the system, facilitating its ability to adapt to changing conditions and recover from errors efficiently.

4. Behavior example of the system

To illustrate the proposed system's behavior in a dynamic environment, consider a scenario where the robot is tasked with picking up a bottle from a table in a room and placing it in a designated area in other room. In this example, the system must react to unexpected changes, such as the bottle being moved by an external agent during the task.

Example   User Request: The user provides the instruction: "Go to the table in the kitchen, pick up the bottle, and place it on the table in the bedroom." **Perception Module:** The robot senses the environment and generateds a semantic map, which includes the pose and types of objects. **Task Planner:** The request is interpreted, and the Task Planner begins the planning procedure. It starts by calling perception actions to gather information about the environment and the robot's state. Using the results of these perception actions, the Task Planner determines the next steps in its execution by calling skill actions and incorporating the outcomes of each skill into the planning process. LLM interprets the request and reasons as follows.

User Request: "Go to the table in the kitchen, pick up the bottle, and place it on the table in the bedroom."

Robot State: The robot is currently in the bedroom and has both the right and left arms empty.

Thought: The task is to fetch the bottle from the kitchen and place it on the table in the bedroom. I need to know all the rooms where the robot can go to.
Perception action: `GetMapRooms()`
Observation: The robot identifies the rooms: `[kitchen, bedroom]`.

Thought: Now, I will navigate to the kitchen to find the bottle.
Skill action: `GOTO(kitchen)`
Observation: The robot moves to the kitchen.

Thought: I should identify the specific location of the bottle within the kitchen.
Perception action: `GetObjectInRoom(kitchen)`
Observation: The robot finds the following objects in the kitchen: `[sink on the table_1, stove on the table_1, bottle, table_1, table_2]`.

Thought: I will now move to the location of the bottle.
Skill action: `GOTO(bottle)`
Observation: The robot moves to the bottle's location in the kitchen.

Thought: I will pick up the bottle now.
Skill action: `PICK(bottle, right)`
Observation: The robot can't see the bottle to pick, find a bottle in the semantic map and try again to pick it.

Thought: The bottle is not visible where expected, I need to check the location of the bottle again.
Perception action: `GetObjectInRoom(kitchen)`
Observation: The robot finds the following objects in the kitchen: `[sink on the table_1, stove on the table_1, bottle on the table_2, table_1, table_2 ]`.

Thought: I will go to the table to pick up the bottle instead.
Skill action: `GOTO(table_2)`
Observation: The robot moves to the table.

Thought: I will pick up the bottle from the table now.
Skill action: `PICK(bottle, right)`
Observation: The robot successfully picks up the bottle.

Thought: Next, I need to navigate to the bedroom to place the bottle.
Skill action: `GOTO(bedroom)`
Observation: The robot moves to the bedroom.

Thought: I need to locate the table in the bedroom where I will place the bottle.
Perception action: `GetObjectInRoom(bedroom)`
Observation: The robot sees: `[table, bed, lamp on the table]` in the bedroom.

Thought: I will approach the table to place the bottle.
Skill action: `GOTO(table)`
Observation: The robot approaches the table.

Thought: Finally, I will place the bottle on the table.
Skill action: `PLACE(bottle, right)`
Observation: The robot places the bottle on the table successfully.

Skill Planner: For each high-level skill, once the preconditions are met, the Skill Planner translates the choosen skill in a sequence of low-level commands, such as motor movements for navigation,

arm articulation for picking, and placing actions. For example once the `PICK(bottle, right)` skill is planned, it can be translated and divided into the following phases with relative commands:

- Approach: The robot arm moves towards the object's position, making any necessary adjustments to align correctly, and opens the gripper.
- Grasp: The robot activates the gripping mechanisms to seize the object. This phase includes closing the gripper and verifying the grasp.
- Lifting: The robot lifts the object from the surface it is on.

Execution: The Executor begins executing the planned skill, which is composed of a sequence of commands by the Skill Planner. The Executor follows the ordered steps to achieve the goal. For example with the skill `PICK(bottle, right)`, the Executor receive the list of command and execute:

- Execute approach: The robot arm moves towards the object's position and open the gripper.
- Execute grasp: This phase includes closing the gripper and verifying the grasp.
- Execute lifting: The robot lifts the object from the surface it is on.

Thus, when an unexpected event occurs, such as the bottle being moved or is not reachable the executor may raise a failure message.

Controller and **Explainer** interaction:

- The Controller detects that the object is no longer in the expected location and sends a failure message to the Explainer.
- The Explainer analyzes the failure, referencing previous instances where objects were moved unexpectedly. It suggests the Task Planner to re analyse the semantic map and update the object's location.

Re-planning: Based on the suggestion, the Task Planner issues a new plan:

- Execute `GOTO(table)` to go near the identified bottle.
- After locating the bottle on the table, the robot updates its actions and proceeds to execute the remaining tasks.

This example demonstrates how the system adapts in real-time, allowing for continuous task execution even in dynamic and unpredictable environments.

Planning algorithm   We now formalize this process in the form of an adaptive planning algorithm. In this algorithm, the used LLM is a generalist model such as *Llama 3 70B Instruct* [35], whose behavior we influence through in-context learning [9].

Figure 3: Robee, humaniod robot developed by Oversonic Robotics.

Algorithm 1 Planning with extedend ReAct Framework

1: **Input:** User request r, Robot state R_s
2: **Output:** Execution of user request
3: **procedure** PLANNING(r, M)
4:     $C_0 \leftarrow$ InitializeLLMContext(r, M, R_s)
5:     **while** not goal achieved **do**
6:         $action \leftarrow$ TaskPlanner(r, C_0)   ▷ Get first skill
7:         **if** $action =$ "Skill" **then**
8:             $commands \leftarrow$ SkillPlanner($skill$, C_t)   ▷ Translate skill into low-level commands
9:             $success \leftarrow$ Executor($commands$)   ▷ Execute commands
10:             **if** $success =$ False **then**
11:                 $failureMsg \leftarrow$ Controller(C_t)   ▷ Detect failure
12:                 $c_t \leftarrow$ Explainer($failureMsg$)   ▷ Generate suggestion
13:             **else**
14:                 $c_t \leftarrow$ Skill succesfully executed
15:             **end if**
16:         **else**
17:             $c_t \leftarrow$ CallPerceptionAction()   ▷ Reading semantic map from Perception Module
18:         **end if**
19:         $C_{t+1} \leftarrow$ UpdateContext(C_t)   ▷ Update context
20:         $skill \leftarrow$ TaskPlanner(r, C_{t+1})   ▷ Get next skill based on updated context
21:     **end while**
22: **end procedure**

This algorithm shows the adaptive behavior of the system by incorporating feedback loops that facilitate real-time re-planning. By alternating between action and reasoning phases, the robot can continuously adapt to changes, ensuring task success even in unpredictable environments.
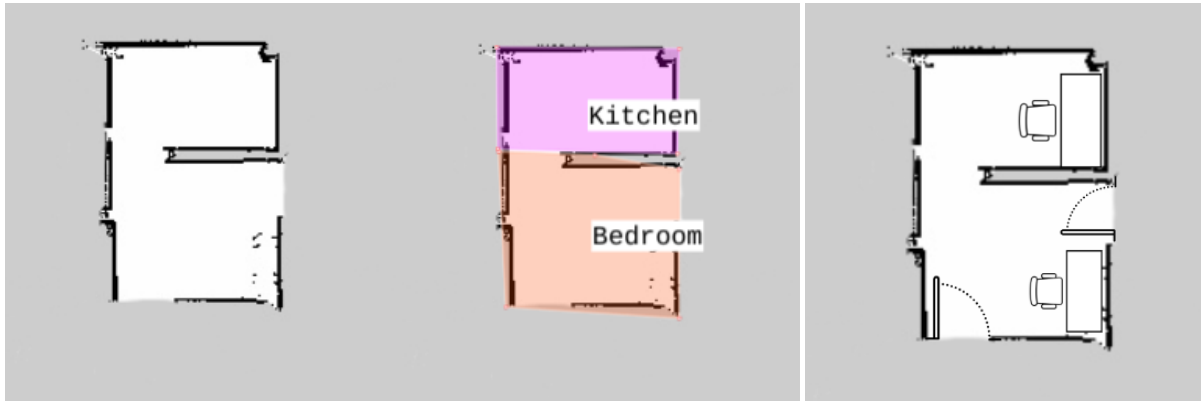
Figure 4: Environment used during the execution of experiments.

5. Robot Hardware

The system was implemented using RoBee, a cognitive humanoid robot developed by Oversonic Robotics. RoBee measures 160 cm in height and weighs 60 kg. It has 32 degrees of freedom, enabling highly flexible movement. The robot is equipped with multiple sensors, including cameras, microphones, and force sensors.

The cameras provide real-time visual data, supporting navigation and object recognition tasks. The microphones facilitate audio input, enabling speech recognition and interaction through natural language processing. The force sensors are used for handling objects, allowing RoBee to adjust grip force based on the characteristics of the item being manipulated, enhancing precision and safety during interactions.

RoBee's mechanical structure includes two arms capable of bimanual manipulation, each capable of handling objects weighing up to 5 kg. The system includes a torso and leg system designed for balance and mobility. RoBee is equipped with LIDAR sensors for real-time environment mapping and obstacle detection. These LIDAR sensors enable the robot to navigate autonomously through complex environments, ensuring safe operation in shared spaces. The combination of autonomous navigation technologies and LIDAR-based detection enhances the ability of RoBee to move efficiently and avoid collisions in dynamic industrial environments.

In addition to its physical capabilities, RoBee integrates with cloud-based systems, allowing for remote monitoring, task scheduling, and data analytics.

The Planner-module takes into account RoBee's embodiment, ensuring that the system is aligned with the robot's capabilities such as its degrees of freedom, sensor suite, and ability to perform manipulation and navigation.

6. Preliminary results

Preliminary experiments were conducted in a simulated environment replicating two main rooms: a *kitchen* and a *bedroom*, as illustrated in Figure 6.

During the experiments, three types of requests were tested, each varying in complexity:

- **Simple requests**: direct commands that involve only one skill. For example, *"Pick up the bottle in front of you"*, where the task planner needs only to identify the parameters and activate the appropriate skill.
- **Moderately complex requests**: tasks that require the robot to perform multiple skills in sequence, as explicitly described in the command. An example is *"Go to the kitchen, pick up the bottle, and bring it to the table in the bedroom"*, which involves multiple skills. These tasks require a higher level of complexity, with planning across several steps and handling potential failures.

- **Complex requests**: such as *"I'm thirsty, can you help me?"*, which were more open-ended and required the robot to interpret the task and break it down into multiple steps.

The results in table 1 showed that the system performed well with simple requests, followed by moderately complex ones. However, the success rate for complex requests was significantly lower, with only 25% of the tasks completed correctly. This lower performance was attributed to the system's difficulty in understanding and managing ambiguous or under-specified instructions.

It is important to note that these are preliminary results, and further analysis is ongoing. A thorough evaluation of the data is currently underway, including a comparison with the state of the art in robot task execution and natural language understanding. This will allow for a deeper understanding of the system's strengths and areas for improvement.

Request type	Number of attempts	Success rate
Simple requests	30	90%
Moderately complex requests	20	75%
Complex requests	10	25%

Table 1
Number of attempts and success rate for each request type

7. Conclusions

The proposed planning system exhibits notable strengths, particularly its adaptability and seamless with the robot's diverse set of skill for executing complex tasks. The system's core advantage lies in its ability to interpret user commands through natural language processing, converting them into high-level actions that are further refined into low-level, executable tasks. By integrating real-time environmental feedback from the Perception Module through an extended version of ReAct framework, the system can dynamically adjust to unexpected situations, such as obstacles or execution failures. This adaptability is supported by an architecture, where the Task Planner, Skill Planner, Controller, and Explainer components work in harmony to ensure smooth task execution even in changing environments.

One of the system's key strengths is its ability to manage error recovery through feedback loops, allowing the robot to adapt quickly to failures during task execution. The Explainer module provides on the fly suggestions to modify the plan based on past errors, enhancing the system's validity. The use of semantic maps and scene graphs provides the robot with a structured understanding of its environment, ensuring that actions are contextually accurate and responsive to real-world conditions.

The integration of LLMs, perceptual feedback, and flexible task planning mechanisms makes the system highly versatile for complex, dynamic environments. Its implementation on RoBee, the humanoid robot developed by Oversonic Robotics, has demonstrated its practical potential, positioning it as a valuable tool for applications requiring advanced human-robot interaction and adaptability in unpredictable settings.

In the future, other than extending the low level skill set available, we will investigate the possibility to autonomously expand the Explainer dataset as well as providing similar information directly to the Task Planner, increasing flexibility and reliability and reducing the number of re-planning events. We will also study capability of the system to proactively acquire information about the environment [14] and human partners both through sensors [36] and communication strategies, leveraging the potential for proactive information gathering behaviours of LLMs [37, 38, 39]. Moreover, it will be crucial to assess the reliability of the system both at the planning level as well as the communication level, considering the introduction of embodiment and environment while the limitation in pragmatic understanding of LLM are still to be understood [39, 40, 41].

Acknowledgments

References

[1] D. Aineto, R. De Benedictis, M. Maratea, M. Mittelmann, G. Monaco, E. Scala, L. Serafini, I. Serina, F. Spegni, E. Tosello, A. Umbrico, M. Vallati (Eds.), Proceedings of the International Workshop on Artificial Intelligence for Climate Change, the Italian workshop on Planning and Scheduling, the RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and the Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (AI4CC-IPS-RCRA-SPIRIT 2024), co-located with 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024), CEUR Workshop Proceedings, CEUR-WS.org, 2024.

[2] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al., Palm-e: an embodied multimodal language model, in: Proceedings of the 40th International Conference on Machine Learning, 2023, pp. 8469–8488.

[3] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al., Do as i can, not as i say: Grounding language in robotic affordances, arXiv e-prints (2022) arXiv–2204.

[4] J. Wang, Z. Wu, Y. Li, H. Jiang, P. Shu, E. Shi, H. Hu, C. Ma, Y. Liu, X. Wang, et al., Large language models for robotics: Opportunities, challenges, and perspectives, arXiv preprint arXiv:2401.04334 (2024).

[5] F. Zeng, W. Gan, Y. Wang, N. Liu, P. S. Yu, Large language models for robotics: A survey, arXiv e-prints (2023) arXiv–2311.

[6] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Bhambri, L. Saldyt, A. Murthy, Llms can't plan, but can help planning in llm-modulo frameworks, arXiv preprint arXiv:2402.01817 (2024).

[7] S. Tellex, N. Gopalan, H. Kress-Gazit, C. Matuszek, Robots that use language, Annual Review of Control, Robotics, and Autonomous Systems 3 (2020) 25–55.

[8] G. Zhu, L. Zhang, Y. Jiang, Y. Dang, H. Hou, P. Shen, M. Feng, X. Zhao, Q. Miao, S. A. A. Shah, et al., Scene graph generation: A comprehensive survey, arXiv e-prints (2022) arXiv–2201.

[9] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, Z. Sui, A survey on in-context learning, arXiv preprint arXiv:2301.00234 (2022).

[10] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: International Conference on Learning Representations (ICLR), 2023.

[11] L. Heuss, D. Gebauer, G. Reinhart, Concept for the automated adaption of abstract planning domains for specific application cases in skills-based industrial robotics, Journal of Intelligent Manufacturing (2023) 1–26.

[12] M. Shanahan, Frame problem, the, Encyclopedia of Cognitive Science (2006).

[13] L. P. Kaelbling, M. L. Littman, A. R. Cassandra, Planning and acting in partially observable stochastic domains, Artificial intelligence 101 (1998) 99–134.

[14] D. Ognibene, G. Baldassare, Ecological active vision: four bioinspired principles to integrate bottom–up and adaptive top–down attention tested with a simple camera-arm robot, IEEE transactions on autonomous mental development 7 (2014) 3–25.

[15] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, P. Fung, Survey of hallucination in natural language generation, ACM Computing Surveys 55 (2023) 1–38.

[16] O. Ruiz, J. Rosell, M. Diab, Reasoning and state monitoring for the robust execution of robotic

manipulation tasks, in: 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2022, pp. 1–4.

[17] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, A. Zeng, Code as policies: Language model programs for embodied control, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2023, pp. 9493–9500.

[18] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, P. Stone, Llm+ p: Empowering large language models with optimal planning proficiency, arXiv e-prints (2023) arXiv–2304.

[19] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, Y. Su, Llm-planner: Few-shot grounded planning for embodied agents with large language models, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2023, pp. 2998–3009.

[20] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, Y. Liang, Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, arXiv e-prints (2023) arXiv–2302.

[21] I. Armeni, Z.-Y. He, J. Gwak, A. R. Zamir, M. Fischer, J. Malik, S. Savarese, 3d scene graph: A structure for unified semantics, 3d space, and camera, in: Proceedings of the IEEE/CVF international conference on computer vision, 2019, pp. 5664–5673.

[22] Y. Liu, L. Palmieri, S. Koch, I. Georgievski, M. Aiello, Delta: Decomposed efficient long-term robot task planning using large language models, arXiv e-prints (2024) arXiv–2404.

[23] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, N. Suenderhauf, Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning, in: 7th Annual Conference on Robot Learning, 2023.

[24] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, W. Ruml, Replanning for situated robots, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 29, 2019, pp. 665–673.

[25] L. Zha, Y. Cui, L.-H. Lin, M. Kwon, M. G. Arenas, A. Zeng, F. Xia, D. Sadigh, Distilling and retrieving generalizable knowledge for robot manipulation via language corrections, in: 2024 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2024, pp. 15172–15179.

[26] M. Skreta, Z. Zhou, J. L. Yuan, K. Darvish, A. Aspuru-Guzik, A. Garg, Replan: Robotic replanning with perception and language models, arXiv e-prints (2024) arXiv–2401.

[27] H. Geffner, Non-classical planning with a classical planner: The power of transformations, in: European Workshop on Logics in Artificial Intelligence, Springer, 2014, pp. 33–47.

[28] D. Ognibene, G. Pezzulo, H. Dindo, Resources allocation in a bayesian, schema-based model of distributed action control, in: NIPS-Workshop on Probabilistic Approaches for Robotics and Control, 2009.

[29] M. Ho, D. Abel, J. Cohen, M. Littman, T. Griffiths, People do not just plan, they plan to plan, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, 2020, pp. 1300–1307.

[30] S. Russell, E. Wefald, Principles of metareasoning, Artificial intelligence 49 (1991) 361–395.

[31] S. Zilberstein, S. J. Russell, Anytime sensing, planning and action: A practical model for robot control, in: IJCAI, volume 93, 1993, pp. 1402–1407.

[32] R. Ackerman, V. A. Thompson, Meta-reasoning: Monitoring and control of thinking and reasoning, Trends in cognitive sciences 21 (2017) 607–617.

[33] S. J. Russell, P. Norvig, Artificial intelligence: a modern approach, Pearson, 2016.

[34] F. Rahutomo, T. Kitasuka, M. Aritsugi, et al., Semantic cosine similarity, in: The 7th international student conference on advanced science and technology ICAST, volume 4, University of Seoul South Korea, 2012, p. 1.

[35] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al., The llama 3 herd of models, arXiv. org (????).

[36] D. Ognibene, Y. Demiris, Towards active event recognition., in: IJCAI, 2013, pp. 2495–2501.

[37] S. Patania, E. Masiero, L. Brini, G. Donabauer, U. Kruschwitz, V. Piskovskyi, D. Ognibene, Large language models as an active bayesian filter: information acquisition and integration, in: Proceedings of the 28th Workshop on the Semantics and Pragmatics of Dialogue - Full Papers, SEMDIAL, Trento, Italy, 2024. URL: http://semdial.org/anthology/Z24-Patania_semdial_0006.pdf.

[38] A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, et al., Robots that ask for help: Uncertainty alignment for large language model planners, Proceedings of Machine Learning Research 229 (2023).

[39] B. Magnini, Toward collaborative llms: Investigating proactivity in task-oriented dialogues, in: Proceedings of the 28th Workshop on the Semantics and Pragmatics of Dialogue - Invited Talks, SEMDIAL, Trento, Italy, 2024. URL: http://semdial.org/anthology/Z24-Magninini_semdial_0003a. pdf.

[40] A. Martinenghi, C. Koyuturk, S. Amenta, M. Ruskov, G. Donabauer, U. Kruschwitz, D. Ognibene, Von neumidas: Enhanced annotation schema for human-llm interactions combining midas with von neumann inspiredsemantics, in: Proceedings of the 28th Workshop on the Semantics and Pragmatics of Dialogue - Poster Abstracts, SEMDIAL, Trento, Italy, 2024. URL: http://semdial.org/ anthology/Z24-Martinenghi_semdial_0045.pdf.

[41] A. Martinenghi, G. Donabauer, S. Amenta, S. Bursic, M. Giudici, U. Kruschwitz, F. Garzotto, D. Ognibene, Llms of catan: Exploring pragmatic capabilities of generative chatbots through prediction and classification of dialogue acts in boardgames' multi-party dialogues, in: Proceedings of the 10th Workshop on Games and Natural Language Processing@ LREC-COLING 2024, 2024, pp. 107–118.

8. Online Resources

More information about RoBee and Oversonic Robotics are available:

- RoBee,
- Oversonic Robotics