Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

AutoGPT+P: Affordance-based Task Planning using Large Language Models

Timo Birr, Christoph Pohl, Abdelrahman Younes and Tamim Asfour
Karlsruhe Institute of Technology, Germany
{timo.birr, asfour}@kit.edu

***Abstract*—Recent advances in task planning leverage Large Language Models (LLMs) to improve generalizability by combining such models with classical planning algorithms to address their inherent limitations in reasoning capabilities. However, these approaches face the challenge of dynamically capturing the initial state of the task planning problem. To alleviate this issue, we propose *AutoGPT+P*, a system that combines an affordance-based scene representation with a planning system. Affordances are the action possibilities of an agent on the environment and the objects present in it. Thus, deriving the planning domain from an affordance-based scene representation allows symbolic planning with arbitrary objects. *AutoGPT+P* leverages this representation to derive and execute a plan for a task specified by the user in natural language. In addition to solving planning tasks under a closed-world assumption, *AutoGPT+P* can also handle planning with incomplete information, such as tasks with missing objects, by exploring the scene, suggesting alternatives, or providing a partial plan. The affordance-based scene representation combines object detection with an *Object Affordance Mapping* that is automatically generated using *ChatGPT*. The core planning tool extends existing work by automatically correcting semantic and syntactic errors leading to a success rate of 98% on the *SayCan* instruction set. Furthermore, we evaluated our approach on our newly created dataset with 150 scenarios covering a wide range of complex tasks with missing objects, achieving a success rate of 79%. The dataset and the code are publicly available at https://git.h2t.iar.kit.edu/sw/autogpt-p.**

I. Introduction

The effectiveness of natural language interaction between humans and robots has been empirically confirmed as highly efficient [29]. For example, Kartmann et al. [24, 23] demonstrate the incremental learning of spatial relationships through demonstrations and reproducing the learned relationships through natural language commands, providing an intuitive way to manipulate scenes semantically. Despite the recent notable advancements in Natural Language Processing (NLP) and understanding, particularly with the emergence of *Large Language Models* (LLMs), these models still face limitations. Specifically, LLMs currently lack the ability to directly translate a natural language instruction into a plan for executing robotic tasks, primarily due to their constrained reasoning capabilities [38, 1]. Recently, *LLM+P* [28] demonstrated the capacity for enhancing the planning capabilities of LLMs by combining them with classical planners, grounding them with the planning domain and objects within the scene. However, the system is restricted by the closed-world assumption of classical planners. Thus, it can only generate plans if all objects needed to complete the task are available. Furthermore, *LLM+P* has no automated error correction and is vulnerable to contradictory goal definitions of the LLM.

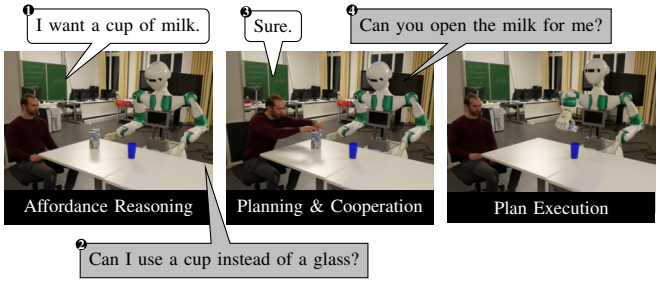

Fig. 1: ARMAR-DE solves the user task given in natural language by detecting the objects within the scene, reasoning about their affordances, planning how to solve the task including asking for help and finally executing the plan.

To overcome these restrictions, we introduce *AutoGPT+P*, a system that enables users to command robots in natural language, derive and execute a plan to fulfill the user's request even if the objects needed to perform the task are missing in the immediate environment. *AutoGPT+P* exhibits dynamic responsiveness by searching the environment for missing objects, proposing alternatives, or progressing towards a subgoal when faced with such constraints.

For instance, if a user requests a glass of milk but no glass is detected in the scene, *AutoGPT+P* proposes replacing the glass with a cup, ensuring task completion by considering alternative objects suitable for the task. When all task-relevant objects are accessible, *AutoGPT+P* can address the closed-world planning problem by extending the *LLM+P* approach with automated semantic and syntactic error correction and dynamic planning domain generation based on the agents' capabilities. Moreover, *AutoGPT+P* endows the robot with the ability to seek assistance from humans when executing an action needed to reach the goal surpasses the robot's capabilities, such as requesting help with opening a milk box.

To formulate and execute plans effectively, *AutoGPT+P* relies on affordances, which represent the action possibilities that an object or environment offers to an agent [15]. For instance, a knife affords cutting, grasping, or stirring. Leveraging the concept of affordances enables the dynamic deduction of viable actions within a given scene, facilitating the formation of a plan to achieve the user's objective. Moreover, affordances allow for reasoning about how to substitute a missing object

with suitable alternatives based on their functionality within the given task.

AutoGPT+P consists of two stages: the first involves perceiving the environment as a set of objects and extracting the scene affordances based on visual data. This is achieved by combining object detection and an *Object Affordance Mapping* (OAM), which describes the relations between object classes and the set of affordances associated with instances of those classes. Subsequently, in the second stage, task planning is conducted based on the established affordance-based scene representation and the user's specified goal. Here, *AutoGPT+P* utilizes an LLM to select tools that support generating a plan to accomplish the task. We quantitatively evaluate our approach in simulation using 180 scenarios with different goals to accomplish manipulation tasks like picking and placing, handover, pouring, chopping, heating, and wiping. Additionally, we performed real-world validation experiments with a humanoid robot demonstrating a subset of these tasks.

To summarize, the main contributions of this work are: (i) a novel affordance-based scene representation combining object detection and automatic *Object Affordance Mapping* (OAM) using *ChatGPT* (ii) a task planning approach based on the established OAM and an LLM-based tool selection to generate plans, partial plans, explore and suggest alternatives in case of missing objects needed to achieve a task goal specified by the user in natural language, (iii) an extension of the *LLM+P* planning approach with automated semantic and syntactic error correction and dynamic domain generation, and (iv) real-world validation experiments with a humanoid robot demonstrating a subset of these tasks

The remainder of this work is structured as follows: First, we provide a comprehensive review of related work for both affordances and LLMs in planning tasks in Section II. Then, in Section III, we describe our proposed scene representation followed by our planning approach *AutoGPT+P* in Section IV. Subsequently, we discuss the results of our quantitative evaluation in simulation and our validation experiments on a humanoid robot.

II. Related Work

A. Affordances in Planning

The use of affordances in PDDL planning domains was initially proposed by [31] in a limited case study of using a crane-like robot to trigger switches with toy blocks. They do not distinguish between objects in their approach but only identify affordances so that specific objects cannot be selected. PDDL is a generic planning language used to define planning domains and problems within those domains. When combined with a classical planner, this PDDL goal allows generating a plan using a classical planner. The authors in [9, 10] expand the principle with a more sophisticated affordance segmentation approach to define the initial state of their PDDL problem, which is then solved to generate a plan using affordances. Their experiments in three real-world manipulation tasks demonstrate the potential of combining detected affordances to design an affordance-based PDDL domain. The work of Xu et al. [46] trained an end-to-end model that learns to generate a PDDL goal, consisting of (action, subject, object), from an input image and a natural language command, enabling the model to solve relatively simple tasks using an off-the-shelf task planner. All previously mentioned works use an affordance-based PDDL domain, which implicitly allows them to replace objects with those of the same affordance. In contrast to classical planning, the acquisition of relational affordances through probabilistic learning, used alongside a probabilistic planning algorithm that maximizes the likelihood of reaching the goal rather than minimizing plan length, was demonstrated in [32]. Relational affordances present a generalized affordance representation as a joint probability distribution over all objects, actions, and effects. The authors of [40] suggest using the (object, action, effect) relationship in task learning using reinforcement learning. Using actions only when the effect is relevant to achieving the goal reduces the search space and improves task learning in real-world navigation tasks. The work of [3, 4, 5] focuses mainly on affordance-based object replacement in planning tasks. Using a modified Hierarchical Task Network planning algorithm, they achieve flexible explicit object substitution based on functional affordances extracted by crawling dictionary definitions.

To the best of our knowledge, our work is the first to use affordance-based planning in everyday long-horizon tasks while employing implicit and explicit substitutions in planning. An overview of the related work can be found in Table I. For the comparison, we define long-horizon tasks as tasks that need more than seven actions to be fulfilled, similar to [1].

B. Large Language Models in Task Planning

Recently, LLMs have demonstrated remarkable new capabilities and outperformed humans in various domains. However, their coherent reasoning skills remain somewhat lacking [38]. Nevertheless, countless recent examples of using LLMs in robotic task planning exist. According to the authors of [35], three general operation modes exist: subtask evaluation

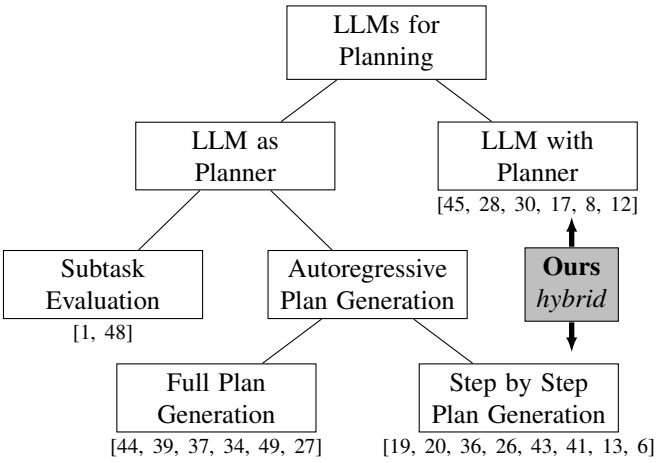

Fig. 2: A taxonomy of LLMs in planning tasks with the related work from this section referenced.

Approach	Planning Method	# Affordances	Substitutions	Long-Horizon	Novel Classes
Lörken and Hertzberg [31]	PDDL + Planner	2 (liftablibility & pushability)	implicit	no	no
Wang et al. [40]	Reinforcement Learning	1 (movability)	none	no	yes
Awaad et al. [3, 4, 5]	Hierarchical Task Network	(not directly listed)	explicit	yes	no
Moldovan et al. [32]	Own Algorithm similar to Monte Carlo Search	2 (tap and push)	none	no	no
Chu et al. [9, 10]	PDDL + Planner	7	implicit	no	yes
Xu et al. [46]	PDDL + Planner	4 (grasp, cut, contain, support)	implicit	no	yes
Ours	PDDL + Planner / Hybrid with LLM	16	explicit & implicit	yes	no

TABLE I: Comparison to the state of the art in affordance-based planning.

mode, full autoregressive plan generation, and step-by-step autoregressive plan generation. However, this classification is only sufficient when using the LLM as the planner. Recent studies, exemplified by [28] and [8], introduced a different paradigm where LLMs are used as symbolic goal generators in conjunction with a planner, which we call *LLM with Planner*. This section aims to distinguish between the different approaches and their respective categories, for which an overview is provided in Fig. 2.

1) Subtask Evaluation: In each step of plan generation, all possible actions are scored partly based on a probability provided by the LLM. The action with the best score is selected. One of the most well-known works on LLM-based task planning *SayCan* [1] utilizes a combination of a Reinforcement Learning-based affordance function and an LLM to predict the likelihood of an action. This affordance function describes the feasibility of an action in the given environment and differs from the previously described affordances. The plan is generated incrementally by selecting the action with the highest score resulting from multiplying the two functions. *SayCan* is expanded on in [48] by assessing the next-best action greedily and performing a tree search. They propose improving Monte Carlo Tree Search planning by incorporating an LLM that utilizes the context of the world state in the form of a common-sense heuristic policy.

2) Full Autoregressive Plan Generation: In this mode, the entire plan is generated based on the user-specified task. The works of Wu et al. [44] and Wake et al. [39] provide a straightforward grounding method for the LLM by identifying objects in the scene, eliminating duplicates, and simply appending the object list to the prompt. The prompt instructs the LLM on what actions to generate and only permits using the provided objects when generating the plan. Song et al. [37] improve on that general idea by utilizing dynamic in-context example retrieval to enhance performance and enable the LLM to replan in the event of an error. The authors of [34] reduce problem complexity by not giving the LLM all objects but filtering out irrelevant objects by traversing a 3D scene graph via collapsing and expanding notes before letting the LLM output a plan. Zhou et al. [49] first translate the problem in natural language into a PDDL domain and problem using the LLM. Then, based on this PDDL domain, they let the LLM generate a plan, which the LLM itself can validate and a more precise external off-the-shelf PDDL validator. The LLM can then use this feedback again to correct itself. A hybrid approach between subtask evaluation and full plan generation is introduced in [27], combined with a semantic checking of whether the goal condition is fulfilled. Their approach iteratively attempts to generate the entire plan. If the goal is not satisfied, a greedy step is taken to select the best next step according to a mixture of the LLM score and the skill feasibility.

3) Step-By-Step Autoregressive Plan Generation: In contrast to the previous mode, the plan is generated one action at a time, which allows for feedback from the execution of the action to improve the planning success rate. Huang et al. [19] addresses the issue of the LLM not being grounded in the actual scene and robot capabilities by introducing a two-step process: Firstly, a planning-LLM generates an ungrounded plan, which is then translated to the robot's abilities by a Translation-LLM. In a follow-up work [20], they first proposed generating the plan step-by-step with the LLM and continuously injecting feedback after each step to enhance the performance. The works of [36, 26, 43, 6] handle plan generation similarly but take advantage of the code generation capabilities of LLMs by letting it generate the plan as Python code. In Progprompt [36], feedback is injected with Python error messages to correct code that does not work due to syntactic or semantic errors. Tidybot [43] additionally enables customization of preferences with a two-step method where the LLM initially identifies patterns from prior Python code and then generates new Python code with those patterns and supplementary instructions. [6] proposes a system that additionally detects corrections of the plan in natural language by the human and incrementally learns to adapt plan generation in future tasks. Wang et al. [41] propose a 4-step approach: describe, explain, plan, and select, where the LLM is responsible for the describe and explain steps. In a feedback loop, the LLM iteratively generates plans that are executed until a failure occurs. Each time the plan fails, the descriptor describes the current state of the goal and the failed action in natural language. The explainer reasons why the plan failed, which is then fed to the planner, which returns a corrected plan. The selector prioritizes the actions in the plan to optimize its execution time. An embodied version of the LLM PaLM that can output plans in natural language based on multi-modal sentences is proposed in [13]. They embed visual

data and the robot state as context for the user-specified task in natural language. Their approach PaLM-E allows embodied long-horizon task planning that can even handle adversarial disturbance.

4) LLM with Planner: The authors of [45] introduce the concept of utilizing a pre-trained LLM to translate natural language commands into PDDL goals. They argue that while LLMs are not adept at reasoning, which is essential for proper planning, they excel at translation. The conversion of natural language into a PDDL goal can be seen as such a translation task. They demonstrate that LLMs are proficient in extracting goals from natural language in commonplace tasks. However, their accuracy diminishes with increasing task complexity. *LLM+P* [28] extends this idea beyond goal generation by generating the entire problem and using a classical planner to solve the task. They also improve the success rate by providing minimal examples for a similar goal state within the given domain. An extension to *LLM+P* with scene graphs to generate the problem's initial state is presented in [30]. To reduce planning time, the authors decompose the overall goal into subgoals that can be solved more efficiently. Guan et al. [17] do not only use the LLM to generate the problem but also the domain itself using syntactic feedback to correct errors. Additionally, they propose a hybrid planning approach that uses plans generated by the LLM as a starting "heuristic" to speed up planning with a local search planner. The creators of AutoTAMP [8] explore a similar direction, but instead of PDDL, they use Signal Temporal Logic Syntax to define the goal state. Furthermore, they use an automatic syntactic and semantic checking loop that verbalizes the error to the LLM and tells it to correct it. In their case, a semantic error is defined as the resulting plan not being sufficient to solve the goal, which the LLM evaluates. In the work of [12], the LLM is used to expand the domain to be able to handle open worlds as PDDL problems are specified under the closed-world assumption. This allows the system to handle situations not explicitly intended by the domain designer. Using a variety of prompts, they let the LLM generate augmentations of the PDDL domain by defining new actions or changing allowed parameter types of actions like replacing a cup with a bowl containing water. This is functionally similar to affordance-based planning, except that in our case, the replacement of one object with another is implicitly given by their shared affordance.

Our work can be seen as a hybrid approach combining the *LLM with Planner* and the *Step-By-Step Autoregressive Plan Generation* paradigms. We extend *LLM+P* by generating the initial state of the problem based on visual perception and the robot's working memory rather than natural language, which allows for a more dynamic plan generation. We also introduce automated syntactic and semantic self-correction for the generated PDDL goal. Furthermore, *LLM+P* is limited to closed-world planning, which cannot handle missing objects. We overcome this limitation with *Step-By-Step Autoregressive Plan Generation*, which iteratively updates the robot's memory by suggesting alternatives or exploring the scene until all necessary objects are found.

III. Affordance-based Scene Representation

The problem we address in this section is to extract an affordance-based scene representation from RGB images of the scene. Our task planning approach uses this representation to generate plans and reason about alternatives to missing objects. Affordances are particularly helpful in the planning context as they allow for actions in the planning domain to be defined by the functionality of the objects involved and not their class, allowing for a more generic planner [31]. Additionally, they provide useful information about how objects can be replaced with objects of the same functionality [4]. To this end, we represent the scene S symbolically as a set of object-affordance-pairs p_i, where each object has one or more affordances assigned to it as in Equation 1:

$$S = \{p_1, \ldots, p_n\}, \text{ with}$$
$$p_i = (o_i, k_i, a_i, b_i) \in (\mathbb{O} \times \mathbb{N}_0 \times \mathbb{A} \times [0,1]^4), \quad (1)$$

where $\mathbb{O}$ is the set of all object classes in the domain and $\mathbb{A}$ is the set of all possible affordances. b_i represents the object's bounding box in normalized coordinates. Thus, the space of all scenes can be expressed as $\mathbb{S} = \mathcal{P}(\mathbb{O} \times \mathbb{N}_0 \times \mathbb{A} \times [0,1]^4)$. The problem of deriving this representation from the image of a scene, i. e., Object Affordance Detection (OAD), can be formalized as in Equation 2:

$$OAD : \mathbb{I} \to \mathbb{S} \quad (2)$$

Our definition of affordances aligns with the representationalist view, as discussed in [7]. We do not use Gibson [15] initial proposal, as we do not factor in the agent's capabilities or rely on visual features to identify affordances. Instead, we use a knowledge-based approach to extract affordances by detecting object classes. To this end, we detect the affordances of the object as a whole rather than identifying which specific parts of the object possess a particular affordance. We separate our approach into two distinct stages, as seen in Fig. 3. The first stage is object detection, and the second is the creation of an *Object Affordance Mapping* (OAM). In the object detection stage, the goal, as defined in Equation 3, is to find a set of object instances $\hat{o} = (o, k) \in \mathbb{O} \times \mathbb{N}_0$ with their bounding boxes $b \in [0,1]^4$ in normalized coordinates given an image $I \in \mathbb{I}$.

$$ObjectDetection : \mathbb{I} \to \mathcal{P}(\mathbb{O} \times \mathbb{N}_0 \times [0,1]^4) \quad (3)$$

The OAM associates object classes with the set of affordances assigned to the instances of those classes, leading to

$$OAM : \mathbb{O} \to \mathcal{P}(\mathbb{A}) \quad (4)$$

The OAM can be generated offline and stored in a database as presented in Section III-A. During inference, the previously generated OAM can be used. The complete approach can be expressed by

$$OAD(I) = \{(o, k, a, b) \mid$$
$$a \in OAM(o), (o, k, b) \in \text{ObjectDetection}(I)\} \quad (5)$$

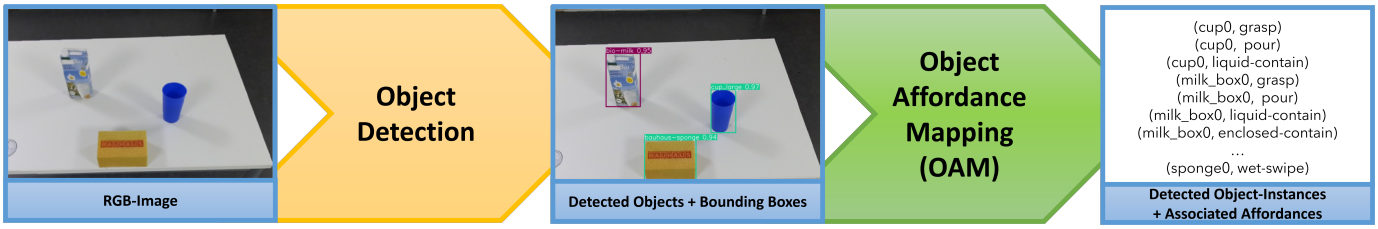

Fig. 3: Overview of Object Affordance Detection(OAD). It uses an RGB image of a scene to detect the objects present in the scene. In the second step, the *Object Affordance Mapping* (OAM) maps the objects to their corresponding affordances.

A. Object Affordance Mapping *using* ChatGPT

LLMs have the ability to reproduce real-world knowledge when prompted in natural language. This is especially true for commonsense knowledge ([22]), including interaction possibilities with everyday objects. We exploit the knowledge reproduction capabilities by querying the LLM with simple questions that do not involve complex reasoning or context understanding. We omit the formatting instructions for the LLM in the prompts for brevity.

- **List-Affordances:** In this strategy, we iterate over all objects we want to get the affordances for and ask the LLM which affordances the object has. As context, we give the LLM a list of affordances with a short description for each affordance. This has the advantage of using only a few tokens and is relatively fast. However, it lacks accuracy, possibly because the descriptions of affordances are sometimes unclear.
- **Yes/No-Questions:** In this strategy, we define a prompt formulated as a yes-no question for each affordance. We then query *ChatGPT* to answer the question with only yes or no without an explanation. In contrast to the first strategy, we can describe precisely what an affordance means. This improves accuracy as *ChatGPT* only needs to generate the answer to a binary question. However, we need significantly more tokens and time for the task.
- **Yes/No-Questions + Logical Combinations**: Early experiments showed that *ChatGPT* cannot handle logical combinations of questions very well. Therefore, in this strategy, queries contain multiple questions divided into atomic prompts, only containing one question each. This approach has the advantage of being more accurate than all other approaches. Still, it has the disadvantage of consuming even more tokens than the yes/no questions, as it often requires multiple questions per affordance.

IV. AutoGPT+P

To generate plans from a user command, *AutoGPT+P* uses a tool-based architecture. The tools are used to iteratively update the robot's memory, which contains the affordance-based scene representation until a final plan is found. A general overview of our approach can be seen in Fig. 4.

A. Problem Formulation

In the following, let $\mathbb{R}_S$ be the set of all possible object relations in the scene S and Λ be the space of natural language. The overall planning task can be specified as given a scene description $S \in \mathbb{S}$, object relations $R \in \mathbb{R}_S$, explorable locations L, and a task in natural language $\lambda \in \Lambda$, the system should return an action sequence or plan $P = (\alpha_1, \ldots, \alpha_n)$ that fulfills the task. An action $\alpha_i \in A$ is defined as the executed capability c by the agent π with the arguments $\rho = (\rho_1, \ldots, \rho_n)$. Here, A refers to the set of all available actions, and a capability defines the symbolic parameters of an action with their logical preconditions and effects. Each agent has a set of capabilities C_π that are dynamically loaded at run-time and are derived from the available skills, which are the programs for low-level action execution on the robot. S can be updated during the process by exploring a location $l \in L$ and adding the object-affordance-pairs $\hat{p} = OAD(I)$ to S with the image I taken at l.

Two relevant sub-tasks during planning are the *closed-world planning* problem and the *alternative suggestion* problem. The closed-world planning based on tasks in natural language can be described as follows: Given the fixed scene representation $S \in \mathbb{S}$, object relations $R \in \mathbb{R}_S$, and the user-specified task $\lambda \in \Lambda$, we need to generate a plan $P = (\alpha_1, \ldots, \alpha_n)$ that fulfills the given task. This can be written as

$$ClosedWorldPlanning : (\Lambda \times \mathbb{S} \times \mathcal{P}(\mathbb{R}_S)) \rightarrow A^{\mathbb{N}} \quad (6)$$

An alternative suggestion is the problem of suggesting an alternative object $alt \in O$, where O is the set of object classes present in the scene, given a user-specified task λ in natural language and a missing object class $o \in \mathbb{O}$ needed to fulfill that task. This can be written as

$$AlternativeSuggestion : \Lambda \times \mathbb{O} \rightarrow O \quad (7)$$

B. AutoGPT+P Feedback Loop

AutoGPT+P is a hybrid planning approach that combines two planning paradigms introduced in Section II-B: *Step-By-Step Autoregressive Plan Generation* for tool selection and an *LLM with Planner* in the *Plan Tool* (Section IV-D). The motivation behind this design is to fill the robot's memory with information using the main feedback loop as specified in Algorithm 1 until the *closed-world planning problem* is solvable with the *Plan Tool*, thus making the planning process more versatile.

The tool selection is the central part of the main feedback loop of *AutoGPT+P*. It can be specified as

$$ToolSelection : \Lambda \times \mathbb{M} \rightarrow T, \quad (8)$$

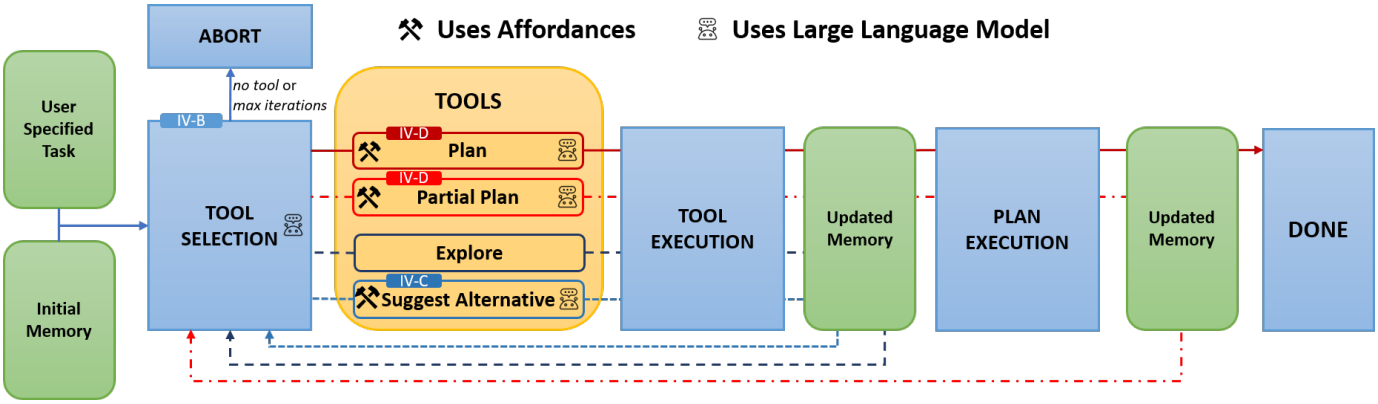

Fig. 4: Overview of the *AutoGPT+P* feedback loop presented in Section IV-B. Green boxes symbolize inputs and outputs, while blue boxes symbolize discrete steps of the process. The tool selection process chooses one of the tools in the yellow *Tools* box. The numbers on top of the boxes show in which section the aspect of the work is explained.

where $\mathbb{M}$ is the space of memory configurations and $T = \{t_1, \dots t_n\}$ is the set of tools. So based on the user prompt $\lambda \in \Lambda$ and the current memory state $M \in \mathbb{M}$, the tool selection returns a tool $t \in T$. The memory $M = (S, R, L, l_\Pi, \hat{\lambda}, Alt, P)$ consists of an affordance-based scene representation S, a set of object relations R, locations L, current agent locations l_Π, instruction history $\hat{\lambda}$, known alternatives $Alt \in (\mathbb{O} \times O)$ and most recent plan P. With M expressed in natural language, the LLM chooses from the following tools:

- **Plan**: solves the problem expressed in Equation 6
- **Partial Plan**: solves the problem expressed in Equation 6 in the best way possible with the restrictions of S.
- **Suggest Alternative**: solves the problem expressed in Equation 7
- **Explore**: move the robot to an unexplored location $l \in L$, extracts the object-affodance-pairs from the camera image and updates the scene representation: $S \leftarrow S \cup OAD(I)$.

After t is selected, it is executed to update the memory M. The generated plan is executed if either the *Plan* or *Partial Plan Tool* is selected. The plan execution assumes all actions are executed successfully and does not replan on failure. These steps are repeated until a final plan is generated via the *Plan Tool*, or no tool is selected, which is interpreted as a failure.

C. Affordance-Based Alternative Suggestion

If an explicitly requested object cannot be found within the scene, our system should reason whether another object in the scene can replace it. One of the reasons for choosing an affordance-based scene representation was that affordances allow us to reason about the functionality of an object. We leverage this reasoning for the alternative suggestion task defined in Equation 7.

Our method uses a handcrafted Chain-of-Thought process [42] consisting of two main steps, detailed in Algorithm 2. First, we query the LLM which of the affordances of the missing object class aff_m are relevant to the task λ specified by the user. We can now filter out all objects in the scene that do not have all these affordances and get the most relevant,

Algorithm 1: AutoGPT+P Feedback Loop

Input: Memory $M = (S, R, L, l_\Pi, \hat{\lambda}, Alt, P)$,
User-specified task λ
while $\neg final$ **do**
  $t \leftarrow ToolSelection(\lambda, M)$
  if t **then**
    $M, final \leftarrow ExecuteTool(t, M)$
  end
  else
    return
  end
  if $M.p$ **then**
    $executePlan(M.P)$
  end
end

which we heuristically assume is the rarest affordance in the scene $a*$. Now, we query the LLM to find out which of the objects is the most similar to the missing object concerning this affordance. If no objects have all the affordances, or the LLM returns an object not present in the scene, the fallback strategy is to query the LLM to suggest the best replacement for the missing object without explicit affordance reasoning. Our evaluation in Section V-C demonstrates that the use of affordances substantially enhances the accuracy of the substitution.

D. Affordance-Based Planning using an LLM with Planner

This component of our approach maps user-specified tasks in natural language to a sequence of parameterized actions α_i under a closed-world assumption as defined in Equation 6. It is an extension of *LLM+P* [28], with the key differences being that our *Plan Tool* (i) generates the initial state of the PDDL problem from our affordance-based scene representation and not from natural language, (ii) dynamically generates the PDDL domain based on the capabilities of the agents and the OAM, and (iii) checks for the semantic and syntactic

Algorithm 2: Alternative Suggestion

Output: Suggested Alternative $alt \in O$
Data: *Object Affordance Mapping* OAM, Large Language Model LLM
$aff_m \leftarrow OAM(m)$
$aff_{rel} \leftarrow LLM.askForAffordances(m, aff_m, \lambda)$
$O_{filtered} \leftarrow \{o* \in O \mid OAM(o*) \subseteq aff_m\}$
if $O_{filtered} = \emptyset$ **then**
  return $LLM.askForObjectDirect(m, O, i)$
end
$a* \leftarrow rarestAffordance(aff_{rel}, O)$
$alt \leftarrow LLM.askForObject(m, a*, O_{filtered})$
if $!(alt \in O_{filtered})$ **then**
  return $LLM.askForObjectDirect(m, O, \lambda)$
end
return alt

correctness of the generated goal and lets the LLM correct its own errors.

Similar to *LLM+P*, we let the LLM generate the desired goal state Γ in PDDL syntax from the user-specified task λ. Therefore, we need to generate a PDDL domain $\Delta = (\Theta, \Phi, A)$ and problem without the desired goal state $\hat{\Xi} = (\omega, \iota)$ as a reference for the LLM as explained in Section IV-D1. The generated goal Γ is then checked for semantic and syntactic correctness as seen in Section IV-D2. If there is an error within the goal, the LLM is fed an error message $error \in \Lambda$ and queried to correct the goal state. If the goal is correct, a classical planner is invoked with the generated domain and problem. This process is repeated until a plan is found or the maximum number of iterations has been reached. It is more formally described in Algorithm 3 and visualized in Fig. 5.

We use the method for both the *Plan* and *Partial Plan Tool*. The only difference between the tools is the prompt used to query the goal state from the LLM, which explicitly allows for an incomplete goal state. A significant advantage of this method compared to *LLM as Planner* is that if the symbolic goal representation accurately represents the given user-specified task, the generated plan will be optimal regarding the number of actions.

1) Dynamic Generation of the Affordance-based Domain and Problem: For the LLM to generate the desired goal state based on the user-specified task, it needs a PDDL domain Δ and problem Ξ without the goal state as context.

In PDDL, types are defined by listing all subtypes of a given type. Our domain has three top-level types: object, agent, and location. As it should be possible to navigate to another agent too, for example, to hand over an object, the agent is a subtype of location. Let $sub(\theta)$ define the set of subtypes of a given type. To build the type hierarchy, we first need to declare all affordances as object subtypes, so $sub(object) = \mathbb{A}$. Afterwards, we need to map all object classes that have a given affordance to the subtype of that affordance. So for all

Algorithm 3: Planning with Self-Correction with external feedback

Input: User-specified task λ, Scene S, Object Relations R, Agent Locations l_Π Capability Sets of Agents C_Π, maximum loops n
Output: Plan P
Data: Large Language Model LLM, Predefined Predicates Φ, Semantic Conditions Σ, Planner $planner$
$\Delta \leftarrow createDomain(S, C_\Pi, \Phi)$
$\hat{\Xi} \leftarrow createProblemInitialState(\Delta, S, R, l_\Pi)$
$loops \leftarrow 0$
while $!P$ & $loops < n$ **do**
  $loops \leftarrow loops + 1$
  if *error* **then**
    $\Gamma \leftarrow LLM.correctGoal(error)$
  end
  else
    $\Gamma \leftarrow LLM.askForGoal(\Phi, \hat{\Xi})$
  end
  $\Xi \leftarrow (\omega, \iota, \Gamma)$
  $error \leftarrow checkSyntax(\Delta, \Xi)$
  $error \leftarrow checkSemantics(\Gamma, \Sigma)$
  if *!error* **then**
    $P = planner.plan(\Delta, \Xi)$
  end
end
return P

affordances $a \in \mathbb{A}$ Equation 9 holds.

$$sub(a) = \{o \mid a \in OAM(o), o \in O\} \tag{9}$$

To model the different capabilities of agents, we need to define a type for each capability available, so $sub(agent) = C_\Pi$. Additionally, we make all agents that have a capability c subtypes of that capability type, so for all capabilities $c \in C_\Pi$ Equation 10 holds.

$$sub(c) = \{\pi \mid c \in C_\pi\} \tag{10}$$

To allow for human-robot collaboration, we need to define the different agent types, robot, and human, and dynamically assign costs to them based on user preference.

The actions can be derived directly from the capabilities by adding an agent of the corresponding capability type to the parameters and adding an action effect to increase the total costs based on the agent type. By associating specific costs for agents in the initial state specification of the PDDL problem, we can influence the participation of each agent. For example, by setting the cost of a human to 1000 and of the robot to 1, the robot will execute all actions that it can perform with its capabilities and will only ask the human for help if there is no other possibility.

To define the problem's initial state, we add each object instance with its type to the problem's object definition. The initial state can be directly derived from R; only the current

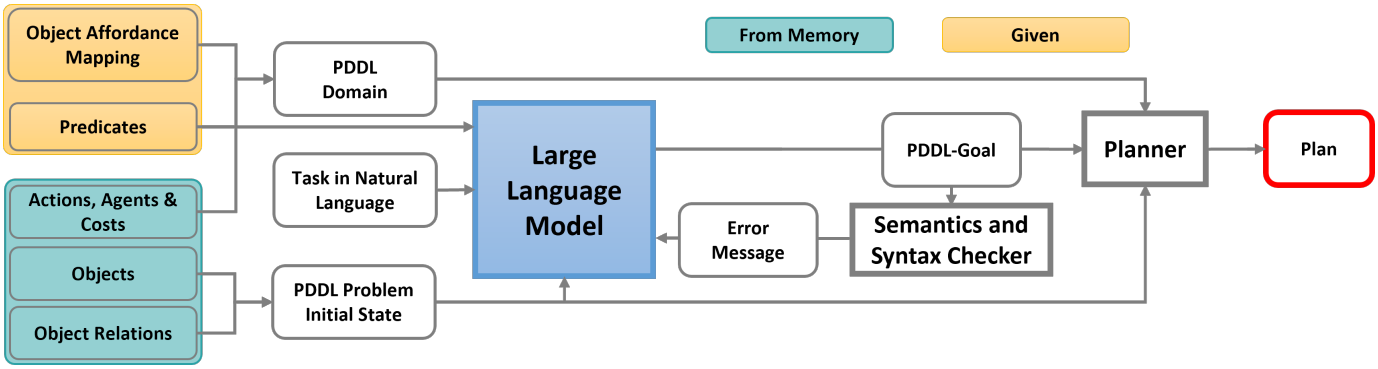

Fig. 5: Overview of the Planning Tool. Rounded boxes represent the input and the output of the components that are represented as rectangles.

agent locations are given by l_π. Finally, the goal state is queried from the LLM to complete the problem.

The advantage of using affordances in our domain is that we only need to define one logical action for all objects with which the action can be performed. Without affordances, we would need a *place* action for all combinations of objects on which other objects can be placed. This would make the domain far more complex and thus increase the search time of the planner.

2) Self-Correction with External Feedback: The work of [16] demonstrated that conversational agents using LLMs can correct themselves when an external program gives an expressive error message. We leverage this capability by detecting syntactic and semantic errors within the goal state and thereby help ground the LLM. A syntactic error can be a wrong use of parenthesis, non-existent predicates, non-existing objects, or predicates with objects of the wrong type or quantity. When parsing the goal state, those errors can be easily checked by matching the predicate names and object names and types with those of the domain and initial state.

We define semantic errors as the occurrence of multiple predicates that cannot be true at the same time in a real scene. This contrasts to [8], where a semantic error is defined as an action sequence that does not fulfill the user-specified task according to the LLM. For example, the object apple cannot be on the table and the counter at the same time, so the goal state `and (on apple table) (on apple counter)` is semantically incorrect. We can express those semantic conditions Σ_Δ using predicate logic and check whether a goal matches a semantic condition in the logic programming language *Prolog*[11], which is an implementation of first-order predicate logic. In the following definition, let the disjunctive normal form (DNF) of Γ be $DNF(\Gamma) = OR(\gamma_1, \ldots, \gamma_n)$ with $\gamma_i = AND(\varphi_{i,1}, \ldots, \varphi_{i,m_i})$. We define a goal state Γ to be sufficient to the set of semantic conditions of the domain Σ_Δ if there exists γ_i where γ_i is sufficient to all conditions in Σ_Δ.

Therefore, in the semantic error check specified in Algorithm 4, we transform the goal state Γ into its DNF and map all sub-states γ_i to Prolog predicates. We then evaluate all semantic conditions for these predicates. If no sub-state matches all semantic conditions, we return the manually specified error message of the condition of the sub-state with the fewest errors to the LLM to correct itself.

Algorithm 4: Semantic Error Check

Input: Generated Goal State Γ, Set of semantic conditions Σ
Output: Unfulfilled Condition σ

```
dnf ← transformToDNF(Γ)
best ← ∅
for γ ∈ dnf do
    failed ← ∅
    for σ ∈ Σ do
        if !checkCondition(γ, σ) then
            failed ← failed ∪ σ
        end
    end
    if |best| > |failed| then
        best ← failed
    end
end
return getMostGeneral(best)
```

V. Evaluation and Validation

We first evaluate the performance of the automated OAM on our proposed affordances. Then, we assess the success rate of our *Suggest Alternative Tool* against a naive alternative suggestion. Furthermore, we compare the *Plan Tool* on its own against *SayCan* on the *SayCan* instruction set and our own evaluation set before evaluating the whole AutoGPT+P system with scenarios focused on tool selection. In this evaluation, *GPT-3* refers to the `GPT-3.5-turbo-0613` model, and *GPT-4* refers to the `GPT-4-0613` model accessed by the OpenAI API.

The quantitative evaluation is conducted via simulation wherein a scene is represented through symbolic object relations, and actions are executed by applying their respective action effects to the scene. For evaluating the *Plan Tool*, all

objects in the scene are known from the beginning, unlike *AutoGPT+P*, where this is not always the case. Exploration was simulated by changing the robot's location and adding all objects designated to the explored location to the robot's memory. We use *Fast Downward* [18] as the planner with a time limit of 300 seconds.

We designed a collection of evaluation scenarios, each consisting of the user's task, the formal goal state to be achieved, and the specifications of the scene, including the objects, relations, and locations. The primary evaluation criterion was whether the generated plan achieves the objective that meets the desired goal state. This can be verified by simulating the plan's actions using Prolog, transforming the goal into its DNF, and assessing whether the set of literals that comprise a substate of the DNF is a subset of the literals that describe the scene state after executing the plan.

A. Object Affordance Mapping using ChatGPT

To evaluate the OAM, the relevant metrics[1] are precision (prec), recall (rec), and F1-score (F1) with

$$\text{prec} = \frac{TP}{TP+FP}\ ,\ \text{rec} = \frac{TP}{TP+FN}\ ,\ \text{F1} = 2 \times \frac{\text{prec} \times \text{rec}}{\text{prec} + \text{rec}}$$

where in our case

- TP is the number of true positives, so object-affordance-pairs (OAP) that are both in the ground truth (GT) and were detected
- FP is the number of false positives, so OAPs that were detected but not the GT
- FN is the number of false negatives, which is the number of OAPs that are in GT but were not detected

An independent "training set" of 30 object classes was used to optimize the prompts. The evaluation involved a test set comprising 70 object classes, each labeled with their respective affordances. These were also used to evaluate the *Plan* and *Suggest Alternative Tool*. We examined the metrics for different affordance extraction methods listed in Section III-A with our 40 proposed affordances that can be seen in the appendix.

GPT	List-Affordances			Yes/No			Logical		
	prec	rec	F1	prec	rec	F1	prec	rec	F1
3	0.31	0.49	0.38	0.70	0.78	0.74	0.78	0.85	0.81
4	0.59	0.67	0.62	0.78	**0.95**	0.86	**0.87**	0.91	**0.89**

TABLE II: Comparison of *ChatGPT* OAM methods on our proposed set of affordances for affordance-based planning with the best values for precision, recall, and F1-score in bold

As the results in Table II indicate, *GPT-4* outperforms *GPT-3* in most cases. The data suggests that the most effective method is the combination of yes/no questions and logic. Despite achieving a high level of accuracy in affordance detection, the uncertainty in affordance estimation is a factor that should be considered in future work.

[1]All these metrics fall within the range of 0 to 1, with higher values indicating better performance.

	GPT-3		*GPT-4*	
	Naive	Ours	Naive	Ours
simple	0.73	0.87	**0.90**	**0.90**
medium	0.63	**0.90**	0.70	0.83
complex	0.33	0.80	0.67	**0.80**

TABLE III: Comparison of the success rate of our *Suggest Alternative Tool* with a naive approach. The best values for the success rate are in bold.

B. Suggest Alternative Tool

For the *Suggest Alternative Tool*, we compare our approach with a naive alternative suggestion approach. This approach asks the LLM to determine which object from the scene can best replace the missing object without any further reasoning. We evaluate the performance of both methods using 30 predefined scenarios. Each scenario includes the missing object, the user-specified task, the objects in the scene, and a list of allowed alternative objects. The task is considered accomplished if the method provides one of the permitted alternatives.

We use 30 scenarios with three difficulty levels, each based on the number of objects present in the scene. The first level is simple and involves five objects. The medium level has twenty objects, whereas the complex level has 70 objects, with one missing. Our rationale behind this setup is that as the number of objects in a scene increases, it becomes more challenging to identify the missing object accurately. Our approach and the naive approach were assessed using *GPT-3* and *GPT-4*, and the results are available in Table III.

We found that as the number of objects in the scene increases, all approaches experience decreased accuracy. However, compared to the naive approach, which experiences a significant drop in accuracy from 0.73 to 0.33 for *GPT-3* and from 0.9 to 0.67 for *GPT-4*, our approach has only a slight drop in accuracy from 0.9 to 0.8 and from 0.87 to 0.8, respectively. In addition, unlike the naive approach, there is only a slight difference in accuracy between *GPT-3* and *GPT-4* when using our approach. This could be because the LLM is guided through the replacement process by a directed Chain-of-Thought process, eliminating incorrect answers.

C. Plan Tool

Our *Plan Tool* was assessed using two sets of scenarios. The first set comprised scenarios from *SayCan* [1], which was utilized to draw comparisons between our method and *SayCan*, a state-of-the-art planning approach using LLMs. We created the second set of scenarios to find the limitations of the LLM's reasoning capabilities for understanding the user's intentions. Therefore, we created five subsets of scenarios, each of them containing 30 prompts with a wide variety of goal tasks from cutting, heating, cleaning, pouring, opening, or moving objects. The *Simple Task* and *Simple Goal* subsets contain simple user requests using either a verb to express the goal or the goal in the form of a state. *Complex Scene* contains task similar to *Simple Goal* but in scenes with 100

instead of around 30 objects like the scenes in all other subsets. *Complex Goal* increases complexity compared to *Simple Goal* by logically connecting the subgoals with phrases like *"and"*, *"or"*, *"if"*, etc. The other two sets *Knowledge* and *Implicit* contain more difficult-to-understand tasks. *Knowledge* requires commonsense knowledge to derive the goal state, while the *Implicit* set does not directly contain a task but more implicitly phrased user intentions like "I am thirsty".

As an additional baseline, we compare our approach to a naive *LLM as Planner* implementation that generates a plan based on a textual representation of the initial scene state, a description of the available actions, and the user-specified task in natural language. To assess whether the affordance information alone improves the planning capabilities of the LLM, we compare it against a second *LLM as Planner* implementation that additionally has information about the objects' affordances in the prompt.

As can be seen in Table IV, our method outperforms the naive *LLM as Planner* implementation for both *GPT-3* and *GPT-4*, confirming the findings of [28] that the *LLM with Planner* paradigm is superior. Contrary to our expectations, the addition of affordance information decreased the performance of the *LLM as Planner* version. This could be explained by the LLM being overwhelmed by the additional affordance information, which was responsible for almost half of the prompt length, and thus lost track of the task at hand. This shows that directly providing affordance information to the LLM is not sufficient, but using the affordance information in a rule-based system like a classical planner – as we do – is a more effective approach.

Moreover, our method performs equally or better than *SayCan* in all categories when using *GPT-4* but performs worse when using *GPT-3*. Utilizing *GPT-4*, our approach outperforms *SayCan*, especially in the *Embodiment* and *Long-Horizon* instruction categories. The *Embodiment* category refers to scenarios where the robot's current state needs to be considered, whereas *Long-Horizon* refers to tasks that need more than 7 actions to fulfill. This is likely owed to our method solely creating a goal state from the user's statement rather than the entire plan. Thus, instructions that require knowledge of the robot's position or current state, or those that require extensive planning, are not as constrained by LLM's limited reasoning capabilities, as the planner generates the plan in a rule-based manner. It should be clarified that *SayCan* is designed to optimize the plan's execution, not just the plan itself. Furthermore, we utilize a more recent LLM as opposed to *SayCan*'s use of PaLM. Also, as can be seen by the leap in success rate from *GPT-3* to *GPT-4*, the used LLM has a significant influence on the performance of our method. Therefore, it is not possible to directly compare to the approach of *SayCan* and make absolute statements about the superiority of one method over the other as long as different LLMs are used as a backbone for planning. However, as PaLM is not publicly available, we are not able to provide an evaluation that makes a direct comparison possible.

We can see from Table V that the planner performs without failure for the *Simple Task* and *Simple Goal* subsets but has more problems with complex goals and the more vague tasks in the *Knowledge* and *Implicit* subsets. *GPT-3* seems to have even more problems interpreting vague user instructions to translate them to goals. Additionally, as the planner always finds the minimal plan for the generated goal, most of the found plans are also minimal. The reason for a generated plan not being minimal is primarily a generated goal that is too restrictive. For example, if the task is "Bring me an apple or a banana" and the generated goal is `inhand apple0 human0` instead of `or (inhand apple0 human0) (inhand banana0 human0)`, the generated plan will not be minimal if it requires more actions to bring the banana than the apple. The average planning time of our approach in the *SayCan* set of instructions was 3.3 seconds with *GPT-3* and 15,4 seconds with *GPT-4*. The scene complexity had a significant influence on the planning time of the *Fast Downward* planner and only a minor influence on the inference time of the LLM. Comparison between the *Simple Goal* and *Complex Scene* sets show that increasing the object number from 30 to 100 only marginally increases the average LLM inference time from 2.7 to 2.8 seconds for *GPT-3* and from 19.1 to 21.8 seconds for *GPT-4*. The overall planning time increases from 8.4 to 31.6 for *GPT-3* and from 28.0 to 59.4 seconds for *GPT-4*. This shows that the bottleneck for the planning time in more complex scenes is the planner and not the LLM.

The results also show the efficiency of self-correction with external feedback, which is a central improvement of our work over *LLM+P*. The *Plan Tool* without self-correction can be seen as an implementation of *LLM+P* that generates the initial state of the PDDL problem directly from the environment representation instead of natural language. It improves the success rate slightly from 0.32 to 0.49 for *GPT-3* and 0.79 to 0.81 for *GPT-4*. The improvement is rather insignificant on our dataset, however, on the *SayCan* instruction set, the difference between *GPT-4* with and without self-correction is far more significant, showing an improvement from 0.78 to 0.98, as can be seen in Table IV. It is mostly caused by the *Structured Language* instruction family and can be mostly explained by semantic error correction. For example, for the task "Pick up the apple and move it to the trash", *ChatGPT* answers with

```
and (inhand apple0 robot0) (in apple0 trash_can0)
```

which is recognized by the semantic error detection and responded with the error message "There is a logical contradiction in the goal. An object that is in the hand of an agent cannot be in another hand or at another place. Please correct your answer". *ChatGPT* then corrects its answer to

```
in apple0 trash_can0.
```

Overall, the results show that even though simple and explicit tasks can be translated well by *GPT-4*, it struggles to correctly interpret the user's intentions when the goal is more indirectly stated. Contextual cues from the environment must be considered to understand the user's goal, as most humans would be able to do. In addition, this also shows that the system as a whole should seek clarification if the user's

Instruction Family	*SayCan* (plan)	*GPT-3* As Planner	*GPT-3* As Planner+A	*GPT-4* As Planner	*GPT-4* As Planner+A	*GPT-3* Ours	*GPT-3* Ours (Auto)	*GPT-4* Ours	*GPT-4* Ours (Auto)
NL Primitive	0.93	0.47	0.53	0.93	0.93	0.73	0.73	**1.00**	**1.00**
NL Verb	0.60	0.00	0.00	0.67	0.87	0.27	0.33	0.93	**1.00**
NL Noun	0.93	0.13	0.07	0.26	0.20	0.27	0.40	0.93	**1.00**
Structured	**0.93**	0.20	0.13	0.87	0.60	0.00	0.13	0.20	**0.93**
Embodiment	0.64	0.09	0.00	0.55	0.55	0.64	0.64	0.82	**1.00**
Crowd-Sourced	0.73	0.13	0.07	**0.93**	0.73	0.27	0.33	0.73	**0.93**
Long-Horizon	0.73	0.00	0.00	0.40	0.33	0.20	0.33	0.80	**1.00**
Drawer	**1.00**	0.00	0.00	0.00	0.00	0.66	0.33	**1.00**	**1.00**
Average	0.81	0.14	0.12	0.66	0.59	0.34	0.40	0.78	**0.98**

TABLE IV: Ablation results of the planning success rate with our *Plan Tool* with different versions of GPT and with(Auto) or without automatic self-corrections on the *SayCan* instruction set. *GPT-X as Planner* refers to the naive baseline of using the LLM directly as the planner, *GPT-X as Planner+A* refers to the same planner with additional context information about affordances added to the prompt.

Subset	*GPT-3*		*GPT-3* Auto		*GPT-4*		*GPT-4* Auto	
	success	min	success	min	success	min	success	min
Simple Task	0.70	0.63	0.70	0.63	0.97	0.97	**1.00**	**1.00**
Simple Goal	0.63	0.60	0.90	0.83	**1.00**	**0.97**	**1.00**	0.93
Complex Scene	0.17	0.13	0.77	0.53	0.93	0.87	**0.97**	**0.93**
Complex Goal	0.23	0.17	0.33	0.23	**0.87**	0.70	**0.87**	**0.73**
Knowledge	0.10	0.10	0.10	0.10	0.53	0.53	**0.57**	**0.57**
Implicit	0.10	0.10	0.13	0.13	0.43	0.40	**0.47**	**0.43**
Average	0.32	0.29	0.49	0.42	0.79	0.74	**0.81**	**0.77**

TABLE V: Ablation results of planning with our *Plan Tool* with different versions of GPT and with or without automatic self-corrections (Auto) on our instruction set. Success refers to the success rate, whereas min refers to the rate of plans that had the minimal length possible for the given goal.

intentions are unclear.

D. AutoGPT+P

As the planning process for *AutoGPT+P* involves several steps beyond just planning, using previous scenarios and metrics alone is inadequate. For this reason, we do not only evaluate *AutoGPT+P* against *SayCan* as the scenarios from the *SayCan* set of instructions do not include exploration or explicit object substitution. A crucial aspect of *AutoGPT+P* is selecting the appropriate tool for each situation and only calling tools other than the *Plan Tool* if they are not in the scene. Therefore, we have incorporated an evaluation metric to determine the optimal number of tools and evaluated the rate of successful plans that use the optimal number of tools. This metric is referred to as *minimal tools* in Table VI.

We designed five scenario sets to assess performance, each containing 30 scenarios. Four of these sets concentrate on individual tools, while the final set requires combining all tools to accomplish complicated tasks. We randomly picked scenarios from the prior segment for the *Plan* subset. Meanwhile, we crafted entirely new scenarios for the *Explore* and *Partial Plan* subsets. For the *Explore* set, hints were partially provided regarding the location of objects, such as "Bring me the cucumber from the fridge". The *Suggest Alternative* and *Combined* sets feature the same scenarios with the exception that for the *Combined* set, only the initial location of the robot is explored. The results can be viewed in Table VI.

Subset	*GPT-3*			*GPT-4*		
	success	minimal	minimal tools	success	minimal	minimal tools
Plan	0.53	0.50	0.30	**0.87**	**0.80**	**0.80**
Partial Plan	0.37	0.20	**0.23**	**0.83**	**0.67**	0.13
Explore	0.10	0.03	0.00	**0.77**	**0.23**	**0.63**
Suggest Alternative	0.13	0.13	0.03	**0.77**	**0.53**	**0.73**
Combined	0.13	0.07	0.10	**0.70**	**0.53**	**0.47**
Average	0.25	0.19	0.13	**0.79**	**0.55**	**0.55**

TABLE VI: Evaluation of AutoGPT+P in the metrics success rate, minimal plan length, and minimal tool usage rate comparing *GPT-3* to *GPT-4*. Best values are written in bold.

As can be seen from the *Plan* and *Partial Plan* set, introducing a prior tool selection process does not make the performance worse compared to the scenarios from the planning evaluation. With exploration involved, the success rate gets slightly worse, with the most common mistake being planning before having explored all relevant locations. The *Suggest Alternative* set also has a similar lowered success rate, caused mainly by invalid alternative suggestions. This is expected as the success rate of the medium-sized scenes was 0.83 for the *Suggest Alternative Tool*. The success rate for the *Combined* set is 0.07 lower than the *Suggest Alternative* set, which shows that the addition of exploration does lead to a lower success rate than just using the *Suggest Alternative Tool* alone. Reviewing the data, the most common reason for failure is planning before all necessary objects or replacements are determined.

What can also be seen is that the tool usage is often minimal when only one tool needs to be used, with the exception of the *Partial Plan* set where the *Suggest Alternative Tool* is often called or the tool selection gets stuck in a loop of selecting *Partial Plan*. From the *Explore* scenarios, we can observe that when given a hint of the location of an item, the tool selection never fails to explore the correct location. However, when not given any clues, the tool selection seems to randomly explore locations even if, from the name of the location, it can be inferred that the relevant objects are unlikely there. For example, the system tries to explore the window location to search for vegetables.

In contrast to *GPT-4*, *GPT-3* mostly fails at the tool selection task. For *Plan* and *Partial Plan* sets, it performs in the same success range as in the *Plan Tool* scenarios. However, the tool selection is not optimal even for those, as seen from the *minimal tools rate*. If it needs to use the *Explore* and *Suggest Alternative* tools, it seemingly chooses the tools randomly and thus has a low success rate.

Overall, the results show the viability of the tool selection process to solve tasks with missing objects and partially unexplored scenes. However, issues like preemptive calling of the *Plan Tool* or inadequate calls of the *Suggest Alternative Tool* remain. Furthermore, as the evaluation set is limited in scope, the results should be approached with caution.

E. Validation Experiments on ARMAR-6 and ARMAR-DE

To validate the feasibility of our system, we performed several experiments on the humanoid robots ARMAR-6 [2] and ARMAR-DE. As our approach primarily focuses on object affordance detection and planning, we made several assumptions to ease the integration on the robot. We relied on predefined object models for manipulation tasks like grasping, placing, and pouring. Additionally, the locations the robot could navigate to and the environment model are entirely known. However, we dynamically detect the locations of all objects that can be manipulated. Furthermore, to detect liquids inside containers, we assume a predefined liquid is in every liquid container. The object relations are estimated based on the related objects' affordances, and the spatial relations of the object poses are estimated by a fine-tuned MegaPose model [25]. For object detection, we used the *yolov5* object detector [21] that was fine-tuned on a predefined object set from [47]. For grasping and placing, we used an affordance-based, memory-centered manipulation framework[33], and for pouring, we used an affordance keypoint detection method [14] to detect the opening of the source container and assume that the target container is symmetric, so moving the object's keypoint above the center of the target object is sufficient. We avoided using real liquids during the experiments to avoid damage to the robot. We performed four kinds of tasks with five different formulations of our user requests each. These included picking and placing, handover, pouring, and wiping tasks. All of these require different levels of human-robot collaboration. Whereas the robot needs no help executing the pick and place and wiping tasks, it needs to ask for help for the pouring tasks to open the liquid container. In handover tasks, the human and the robot are equally part of the task. As shown in the supplementary material, our proposed system, *AutoGPT+P*, exhibits proficiency in generating executable plans on our robot. Out of the 20 real-life scenarios, 15 were planned successfully. Subsequent investigation of failure cases revealed that most of these cases could be attributed to false positive detection of objects or the robot's inability to accurately grasp the target object. This highlights the limited resilience of our approach to failures, which needs to be addressed in future work. A more in-depth evaluation of the experiments can be found in the supplementary material file.

F. Discussion

The results of our evaluation show the potential of *AutoGPT+P* in real-world scenarios. We show that our system can handle difficult planning situations, such as missing objects. Furthermore, our system achieves a high success rate in transforming user-specified tasks into valid plans. For explicitly stated user goals, it has a planning success rate of nearly 100%. However, when the user-specified task is more vague, it is mostly limited by its inability to assess the uncertainty in the user's intent and ask for clarification. Another limitation of *AutoGPT+P*'s application in the real world is that everything is modeled in a deterministic way, so uncertainties caused by object recognition, the LLM-generated *Object Affordance Mapping*, or the unclear user-specified tasks are not taken into account. Furthermore, we do not incorporate any feedback (e. g., about skill failures) from execution into our approach, which makes it prone to errors when executed in real-world scenarios. In our evaluation, we also did not consider plans longer than 20 steps, but [30] provides evidence that *LLM+P*, and thus our approach, cannot effectively solve problems that require longer plans.

VI. Conclusion and Future Work

In this work, we propose representing objects in the scene as a set of object-affordance-pairs. The scene representation is generated through combined object detection and *Object Affordance Mapping* (OAM), where object classes are associated with their affordances. Our work demonstrates the utility of *ChatGPT* in automatically deriving an OAM for novel classes based on a fixed set of predefined affordances. On our newly proposed set of affordances for planning, we achieved an F1-score of 89%.

We utilized the scene representation in *AutoGPT+P*, our proposed planning system, which uses the concept of affordances for planning and alternative suggestions. It consists of an LLM-based tool selection loop that chooses from one of four tools to solve the user-specified task: *Plan*, *Partial Plan*, *Explore*, and *Suggest Alternative*. The *Suggest Alternative Tool* uses the affordances of a missing object to steer the LLM during the alternative suggestion process. Additionally, the *Plan* and *Partial Plan Tool* utilize the LLM to produce goal states in an affordance-based planning domain and generate a plan fulfilling the (partial) goal with a classical planner. The

experiments demonstrate that the *Plan Tool* vastly outperforms the naive baseline of a naive *LLM as Planner* implementation. The self-correction of semantic and syntactic errors has a significant influence, raising the success rate from 78% to 98% when compared to the method without self-correction.

Furthermore, our affordance-guided *Suggest Alternative Tool* outperforms a naive approach in scenes with 20 and 70 objects by 13%. When evaluating the system's overall performance, we reach an average success rate of 79% on our dataset containing 150 tasks. Difficulties persist mainly due to the LLM selecting incorrect tools. Therefore, a reevaluation of the tool selection process is necessary to address this issue.

Our validation experiments show that the generated plans can be successfully executed on the robot and that the symbolic representation of objects from the planning domain can be transferred to the subsymbolic object representations needed for skill execution, however, our experiments also showed the low resilience to failures during execution.

In future work, probabilistic aspects should be integrated for improved accuracy in real-world deployment. This involves representing the OAM as a probabilistic function that can be updated incrementally based on user feedback or execution. Alternatively, direct verbal corrections from the human like "You cannot use a fork for cutting." can also be taken into account to update the probability of a given affordance. The system should also decide whether to retry a failed action or generate a different plan instead based on the updated confidence of the affordance. This could reduce the error rate during execution. Additionally, a probabilistic representation of the object-affordance-pairs, which includes the confidence level from the object detection, can be combined with this. The resulting probabilistic scene representation can then be used in conjunction with a planner that optimizes the probability of a plan's success rather than just plan length.

Furthermore, a more versatile human-robot interaction would be beneficial. This involves equipping the system with the capability to seek clarification if the user's instruction is unclear and granting the user the ability to modify or terminate the plan during execution.

Acknowledgments

The research leading to these results has received funding from the European Union's Horizon Europe programme under grant agreement No. 101070292 (HARIA) and No. 101070596 (euROBIN) and from the Carl Zeiss Foundation through the JuBot project.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, K. Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, A. Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, N. Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, S. Levine, Yao Lu, Linda Luu, Carolina Parada, P. Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, D. Reyes, P. Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, F. Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[2] Tamim Asfour, Mirko Wächter, Lukas Kaul, Samuel Rader, Pascal Weiner, Simon Ottenhaus, Raphael Grimm, You Zhou, Markus Grotz, and Fabian Paus. Armar-6: A high-performance humanoid for human-robot collaboration in real world scenarios. *IEEE Robotics & Automation Magazine*, 26(4):108–121, 2019.

[3] Iman Awaad, Gerhard K Kraetzschmar, and Joachim Hertzberg. Affordance-based reasoning in robot task planning. In *Planning and Robotics (PlanRob) Workshop ICAPS-2013*, 2013.

[4] Iman Awaad, Gerhard Kraetzschmar, and Joachim Hertzberg. Finding Ways to Get the Job Done: An Affordance-Based Approach. *International Conference on Automated Planning and Scheduling*, 24(1):499–503, 2014.

[5] Iman Awaad, Gerhard K Kraetzschmar, and Joachim Hertzberg. The role of functional affordances in socializing robots. *International Journal of Social Robotics*, 7: 421–438, 2015.

[6] Leonard Bärmann, Rainer Kartmann, Fabian Peller-Konrad, Jan Niehues, Alex Waibel, and Tamim Asfour. Incremental learning of humanoid robot behavior from natural interaction and large language models. *arXiv preprint arXiv:2309.04316*, 2024.

[7] Anthony Chemero and Michael T. Turvey. Gibsonian affordances for roboticists. *Adaptive Behavior*, 15(4): 473–480, 2007.

[8] Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers. *arXiv preprint arXiv:2306.06531*, 2023.

[9] Fu-Jen Chu, Ruinian Xu, Landan Seguin, and Patricio A Vela. Toward affordance detection and ranking on novel objects for real-world robotic manipulation. *IEEE Robotics & Automation Letters*, 4(4):4070–4077, 2019.

[10] Fu-Jen Chu, Ruinian Xu, Chao Tang, and Patricio A Vela. Recognizing object affordances to support scene reasoning for manipulation tasks. *arXiv preprint arXiv:1909.05770*, 2019.

[11] Pierre Deransart, Laurent Cervoni, and AbdelAli Ed-Dbali. *Prolog: the standard: reference manual*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 3540593047.

[12] Yan Ding, Xiaohan Zhang, Saeid Amiri, Nieqing Cao, Hao Yang, Andy Kaminski, Chad Esselink, and Shiqi Zhang. Integrating Action Knowledge and LLMs for Task Planning and Situation Handling in Open Worlds. *arXiv preprint arXiv:2305.17590*, 2023.

[13] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu,

Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An Embodied Multimodal Language Model. *arXiv preprint arXiv:2303.03378*, 2023.

[14] Jianfeng Gao, Zhi Tao, Noémie Jaquier, and Tamim Asfour. K-vil: Keypoints-based visual imitation learning. *IEEE Transactions on Robotics*, 39(5):3888–3908, 2023.

[15] James J Gibson. The theory of affordances. *Hilldale, USA*, 1(2):67–82, 1977.

[16] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[17] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. *arXiv preprint arXiv:2305.14909*, 2023.

[18] M. Helmert. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[19] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv preprint arXiv:2201.07207*, 2022.

[20] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models. *arXiv preprint arXiv:2207.05608*, 2022.

[21] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, Yonghye Kwon, Kalen Michael, Jiacong Fang, Zeng Yifu, Colin Wong, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation, 2022. URL https://doi.org/10.5281/zenodo.7347926.

[22] Nikhil Kandpal, H. Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large Language Models Struggle to Learn Long-Tail Knowledge. In *International Conference on Machine Learning*, 2022.

[23] Rainer Kartmann and Tamim Asfour. Interactive and incremental learning of spatial object relations from human demonstrations. *Frontiers in Robotics & AI*, 10: 1–14, 2023.

[24] Rainer Kartmann, Danqing Liu, and Tamim Asfour. Semantic scene manipulation based on 3d spatial object relations and language instructions. In *IEEE-RAS International Conference on Humanoid Robots*, pages 306–313, 2021.

[25] Yann Labbé, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. Megapose: 6d pose estimation of novel objects via render & compare. *arXiv preprint arXiv:2212.06870*, 2022.

[26] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language Model Programs for Embodied Control. *arXiv preprint arXiv:2209.07753*, 2023.

[27] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. Text2Motion: From Natural Language Instructions to Feasible Plans. *arXiv preprint arXiv:2303.12153*, 2023.

[28] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[29] Rui Liu and Xiaoli Zhang. A review of methodologies for natural-language-facilitated human–robot cooperation. *International Journal of Advanced Robotic Systems*, 16(3):1729881419851402, 2019.

[30] Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models. *arXiv preprint arXiv:2404.03275*, 2024.

[31] Christopher Lörken and Joachim Hertzberg. Grounding planning operators by affordances. In *International Conference on Cognitive Systems (CogSys)*, pages 79–84, 2008.

[32] Bogdan Moldovan, Plinio Moreno, Davide Nitti, José Santos-Victor, and Luc De Raedt. Relational affordances for multiple-object manipulation. *Autonomous Robots*, 42:19–44, 2018.

[33] Christoph Pohl, Fabian Reister, Fabian Peller-Konrad, and Tamim Asfour. MAkEable: Memory-centered and Affordance-based Task Execution Framework for Transferable Mobile Manipulation Skills. *arXiv preprint arXiv:2401.16899*, 2024.

[34] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Task Planning. *arXiv preprint arXiv:2307.06135*, 2023.

[35] Christina Sarkisyan, Alexandr Korchemnyi, Alexey K. Kovalev, and Aleksandr I. Panov. Evaluation of Pretrained Large Language Models in Embodied Planning Tasks. In *Artificial General Intelligence*, pages 222–232, 2023.

[36] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *IEEE International Conference on Robotics and Automation*, 2023.

[37] Chan Hee Song, Jiaman Wu, Clayton Washington,

Brian M. Sadler, Wei-Lun Chao, and Yu Su. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. *arXiv preprint arXiv:2212.04088*, 2023.

[38] Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large Language Models Still Can't Plan (A Benchmark for LLMs on Planning and Reasoning about Change). *arXiv preprint arXiv:2206.10498*, 2023.

[39] Naoki Wake, Atsushi Kanehira, Kazuhiro Sasabuchi, Jun Takamatsu, and Katsushi Ikeuchi. ChatGPT Empowered Long-Step Robot Control in Various Environments: A Case Application. *IEEE Access*, 11:95060–95078, 2023.

[40] Chang Wang, Koen V. Hindriks, and Robert Babuska. Robot learning and use of affordances in goal-directed tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2288–2294, 2013.

[41] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. *arXiv preprint arXiv:2302.01560*, 2023.

[42] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*, 2023.

[43] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. TidyBot: Personalized Robot Assistance with Large Language Models. *arXiv preprint arXiv:2305.05658*, 2023.

[44] Zhenyu Wu, Ziwei Wang, Xiuwei Xu, Jiwen Lu, and Haibin Yan. Embodied Task Planning with Large Language Models. *arXiv preprint arXiv:2307.01848*, 2023.

[45] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating Natural Language to Planning Goals with Large-Language Models. *arXiv preprint arXiv:2302.05128*, 2023.

[46] Ruinian Xu, Hongyi Chen, Yunzhi Lin, and Patricio A. Vela. SGL: Symbolic Goal Learning in a Hybrid, Modular Framework for Human Instruction Following. *IEEE Robotics & Automation Letters*, 7(4):10375–10382, 2022.

[47] Abdelrahman Younes and Tamim Asfour. KITchen: A Real-World Benchmark and Dataset for 6D Object Pose Estimation in Kitchen Environments. *arXiv preprint arXiv:2403.16238*, 2024.

[48] Zirui Zhao, Wee Sun Lee, and David Hsu. Large Language Models as Commonsense Knowledge for Large-Scale Task Planning. *arXiv preprint arXiv:2305.14078*, 2023.

[49] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. ISR-LLM: Iterative Self-Refined Large Language Model for Long-Horizon Sequential Task Planning. *arXiv preprint arXiv:2308.13724*, 2023.

APPENDIX

A. Proposed Affordances for Planning

The following table lists all affordances that were detected using our automatic OAM using ChatGPT. Affordances in bold are used in the planning domain that is used for evaluation.

Affordance	Description
grasp	The object can be grasped in any way
carry	The object can be carried with two hands
assisted-carry	Two or more people can carry the object cooperatively without being in each other's way
cut	The object can be used to cut other objects
contain	The object is designed to put either objects or liquids inside of it
liquid-contain	The object is designed to put liquids inside of it
enclosed-contain	The object can be closed so the objects stored inside it do not fall or leak out when moving
pour	The object can be used to pour liquids
precise-pour	The object can be used to precisely pour liquids into small containers like glasses
drink	The object is designed to drink from
constrained-move	The object can only be moved with restrictions as it is mounted to another object like a door
rotate	The object can be turned
axis-roll	The object can be rolled around an axis
free-roll	The object can be rolled freely in any direction as it is approximately sphere-shaped
push	The object can be pushed away from oneself
pull	The object can be pulled towards oneself
open	The object can be opened
close	The object can be be closed
support	The object provides good support for other objects standing on it
stack	The object can be stacked on objects of the same type

Affordance	Description
sturdy-support	The object supports other objects on it and it allows to cut objects on top of it
vertical-support	The object can be leaned against safely
scoop	The object can be used to scoop or shove material like powder or objects
stir	The object can be used as a tool to stir
distance-connect	The object physically connects other objects without the connected objects needing to be in contact
contact-connect	The object connects objects together
pierce	The object can be used to pierce through other objects
pick	The object can be used to pierce other objects to pick them up
hit	The object can be used to hit other objects with
pound	The object can be swung to pound other objects
swing	The object can be swung to hit other objects with
dry-swipe	The object can be used to wipe dust or rubble efficiently
wet-swipe	The object can be used to swipe other objects with water
heat	The object can be used to make other objects or liquids warmer
heat-resistance	The object can be safely exposed to temperatures over 100 degrees celcius
liquid	The object is a liquid
drinkable	The object can be safely drunk by a human
consumable	The object can be safely consumed by a human

B. Description of Experiments on ARMAR-6 and ARMAR-DE

The aim of the experiments is to validate that generated plans can be executed on a real robot. The actually executed actions are subject to improvement but work as a proof of concept for mapping from symbolic to subsymbolic actions. Failure cases during the experiments were mostly caused by object detection and object localization. To make the execution process more straightforward we ignored classes that were often detected as false positives. As the main focus of this work is not object detection, this should be justifiable. We plan to integrate a more robust object detector and localizer into our approach to be less prone to errors. When there were no false negatives or positives our method reliably generated valid plans with only a few failure cases. For example, the goal state for "Bring me a coffee cup" was generated to be and `(in-hand coffee_cup0 robot0) (at robot0 human0)`. The results are described in the following notation:
`Task:` Describes the user-specified task in natural language (With other formulations that resulted in the same plan, but are not shown in the video in brackets)
`Locations:` Describes known locations that can be explored
`Initial State:` Describes the initial state unknown to the robot (except for agent locations). Object relations are discovered by object detection.
`Generated Plan:` Used Tools with parameters, for the PLAN Tool the generated goal and plan are listed below, for SUGGEST ALTERNATIVE the suggested alternative to the missing object is listed below in the notation "missing -> alternative"
`Failures:` user-specified tasks that should result in the same plan but do not (due to reasons stated in brackets). We only describe failure cases due to detection and planning as failure cases due to execution are independent of our approach. For execution failures we simply restarted the experiment.

1) **Pick and Place:**
   `Task:` "Put the sponge next to the screwbox" ("Put the sponge on table1", "Put the sponge on the other table")
   `Locations:` `table0, table1`
   `Initial State:`
   `at robot0 table1, on sponge0 table0, on tea_packaging0 table0, on tea_packaging1 table0, on milk_box0 table0, on coffee_cup0 table0, liquid_in milk0 milk_box0, closed milk_box0, on screw_box0 table1, on spraybottle0 table1, on grease0 table1, on soap0 table1`
   `Generated Plan:`
   `EXPLORE table0`
   `PLAN (for goal: on sponge0 table1)`
   `grasp robot0 sponge0 table0 left` (Robot grasps sponge from table0 with left hand)
   `move robot0 table0 table1` (Robot moves from table0 to table1)
   `place robot0 sponge0 table1 left` (Robot places sponge on table1 with the left hand)
   `Failures`:
   "Put the sponge next to the spraybottle" (due to misdetected spraybottle)
   "Put the sponge on the table in front of you" (the PLAN tool does not know the initial position before exploration, so the wrong table is set as the target)
2) **Handover:**
   `Task:` "Give me a glass" ("Fetch me a glass, "I want to have a glass", "Hand me a glass over")
   `Locations:` `table0, human0`
   `Initial State:`
   `at robot0 human0, on coffee_cup0 table0, on milk_box0 table0, liquid_in milk0 milk_box0, closed milk_box0`
   `Generated Plan:`
   `EXPLORE table0`
   `SUGGEST ALTERNATIVE glass`
   `glass -> coffee_cup`
   `PLAN (for goal: in-hand coffee_cup0 human0)`
   `grasp robot0 coffee_cup0 table0 left` (Robot grasps coffee cup from table0 with left hand)
   `move robot0 table0 human01` (Robot moves from table0 to the human)
   `handover robot0 human0 coffee_cup left` (Robot gives human the coffee cup)
   `Failures:`
   "Bring me a glass" (Results in goal state `and (in-hand coffee_cup0 robot0) (at robot0 human0)`)

3) **Pouring:**

Task: ”I want a glass of water” (”Pour me a glass of water”, ”I want to drink water”, ”I am thirsty”)
Locations: `table0`
Initial State:

`at robot0 table0, at human0 table0, on coffee_cup0 table0, on milk_box0 table0, liquid_in milk0 milk_box0, closed milk_box0`

Generated Plan:

```
SUGGEST ALTERNATIVE glass
    glass -> coffee_cup
SUGGEST ALTERNATIVE water
    water -> milk
PLAN (for goal: liquid_in milk0 coffee_cup0)
    open human0 milk_box0 left (Robot asks human to open the milk box for it)
    grasp robot0 milk_box0 table0 right (Robot grasps milk box from table0 with right hand)
    pour robot0 milk_box0 milk0 coffee_cup0 right (Robot pours milk from the milk box to the coffee cup)
```

Failures:

”I want a cup of water” (LLM keeps calling SUGGEST ALTERNATIVE coffee_cup0, this is an entirely wrong usage of this tool)

4) **Wiping:**

Task: ”I spilled milk on this table” (”Clean this table”, ”Wipe this table”, ”This table is dirty”)
Locations: `table0, table1`
Initial State:

`at robot0 human0, at human0 table0, on screw_box0 table1, on spraybottle0 table1, on grease0 table1, on soap0 table1, on sponge0 table1`

Generated Plan:

```
EXPLORE table1
PLAN (for goal: clean table0)
    grasp robot0 sponge0 table1 left (Robot grasps sponge from table1 with left hand)
    move robot0 table1 table0 (Robot moves from table1 to table0)
    wipe robot0 table0 sponge left (Robot wipes table0 with sponge in left hand)
```

Failures:

”I spilled my milk on the table” (LLM generates goal state `clean table1`, which is clearly not meant as the human is at table0.)