HiCRISP: A Hierarchical Closed-Loop Robotic Intelligent Self-Correction Planner

Chenlin Ming[1], Jiacheng Lin[2], Pangkit Fong[1], Han Wang[3], Xiaoming Duan[1]and Jianping He[1] .

***Abstract*— The integration of Large Language Models (LLMs) into robotics has revolutionized human-robot interactions and autonomous task planning. However, these systems are often unable to self-correct during the task execution, which hinders their adaptability in dynamic real-world environments. To address this issue, we present a Hierarchical Closed-loop Robotic Intelligent Self-correction Planner (HiCRISP), an innovative framework that enables robots to correct errors within individual steps during the task execution. HiCRISP actively monitors and adapts the task execution process, addressing both high-level planning and low-level action errors. Extensive benchmark experiments, encompassing virtual and real-world scenarios, showcase HiCRISP's exceptional performance, positioning it as a promising solution for robotic task planning with LLMs.**

I. Introduction

The advent of Large Language Models (LLMs) leads to a notable trend across various research fields owing to their remarkable abilities in representation [1]–[3], comprehension [4], and logical reasoning [5], [6]. These proficiencies have substantially enhanced the interaction between the virtual and the real worlds. However, artificial intelligence technologies, which are represented by LLMs, usually need a mediator to complete the interaction with the real world which is called an embodiment [7]. In fact, robots with actuators, sensors, and computational cores are the optimal carriers for the four elements of artificial general intelligence (AGI): emergence, agency, affordance, and embodiment. Therefore, robotics, including quadrotors, autonomous ground vehicles (AGVs), and manipulators, is one of the best choices to be the objections for embodiment [8].

Recently, the integration of LLMs into robotics has witnessed remarkable progress, empowering robots with natural language understanding and the ability to generate task plans directly from textual instructions [9]–[14]. These LLMs, such as GPT-4 [15], have demonstrated exceptional prowess in natural language understanding and generation, making them promising tools for enhancing human-robot interactions and autonomous task planning. Moreover, in contrast to conventional robotic systems, the integration of LLMs furnishes robots with the capability to perform a wide array of everyday tasks akin to those undertaken by humans

[1]The Department of Automation, Shanghai Jiao Tong University, and Key Laboratory of System Control and Information Processing, Ministry of Education of China, Shanghai, China. Email: {mcl2019011457, fpjgaoge, xduan, jphe}@sjtu.edu.cn.

[2] The Department of Computer Science, University of Illinois Urbana-Champaign, IL, USA. Email: jl254@illinois.edu.

[3]The Department of Engineering Science, University of Oxford, Oxford, UK. E-mails: han.wang@eng.ox.ac.uk

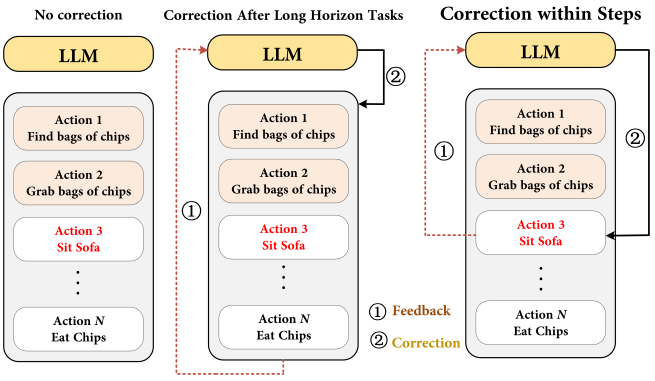

Fig. 1. Illustration of LLM-based robotic systems: (1) without correction, (2) correction after task completion, and (3) correction within steps. An error occurs during the execution of Action 3. Our proposed HiCRISP belongs to the system that performs corrections within individual steps.

[16]. This enhancement has the potential to propel smart robot systems into novel frontiers.

While the integration of LLMs into robotics holds great promise, it is still challenging. One limitation that persists in the current landscape of LLM-based robotics systems is their inability to self-correct during the execution of tasks. Specifically, once an LLM generates a task plan with multiple sequential steps based on textual instruction, it lacks the capability to address deviations from the plan caused by factors like environmental variations or unexpected obstacles. These systems typically proceed with the predetermined plan, often resulting in task failures. This limitation hampers their adaptability in real-world environments where uncertainties and unforeseen obstacles are commonplace.

To empower robotic systems with the capability to address unexpected failures, researchers have delved into various solutions. ProgPrompt [9] employs assertion checking within the generated Python programs to ensure that actions continue to execute when confronted with errors. However, such rigid rule-based approaches usually fail to address failures in a majority of scenarios. To develop more flexible methods, REFLECT [17] and CLAIRify [18] utilize the capabilities of LLMs to detect and correct robot failures by incorporating failure feedback mechanism. Nonetheless, these methods primarily engage in error correction once long-horizon tasks have been completed, lacking the ability to rectify errors on a step-by-step basis during task execution. Consequently, their efficiency and effectiveness are limited.

To address the aforementioned issues and unlock the potential of LLM-based robotics, in this paper, we present our innovative **Hi**erarchical **C**losed-loop **R**obotic **I**ntelligent

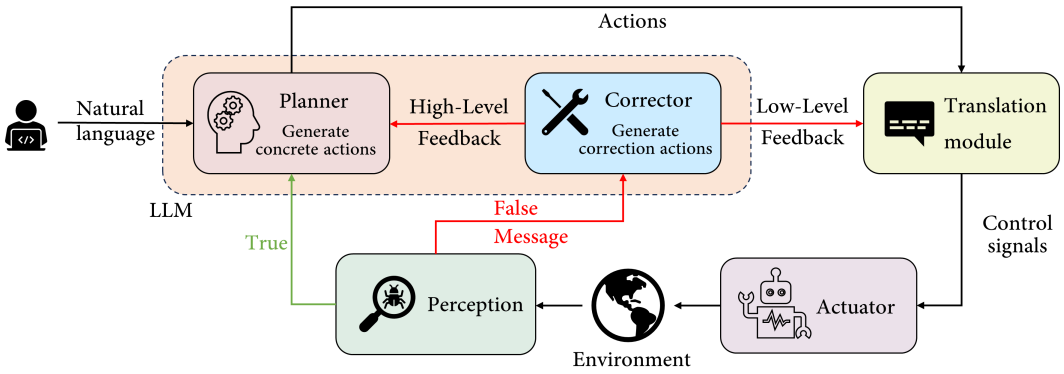

Fig. 2. Overview of our proposed HiCRISP framework. The user first sends a task in natural language, which is broken down by LLM into multiple sub-tasks. The planner, which leverages the semantic understanding capabilities of LLM, generates a sequence of reference actions. Translation module, furthermore, translates actions into the low level control sentences which the robot can executes easily. Perception detects environment information and judges whether the system state changes. If failure is detected, corrector generate correction actions in order to fix error. Corrector fix planning error and executing error through high-level feedback in III-B and low-level feedback in III-C respectively.

Self-correction **P**lanner (HiCRISP). Our approach stands out by its capacity to address errors within individual steps during task execution, rather than waiting until long-horizon tasks are completed, shown in Fig 1. By further incorporating hierarchical self-correction, our proposed HiCRISP can handle both high-level and low-level planning errors, which promotes the success rate of tasks. Our main contributions can be summarized as the following:

- We introduce a novel framework that integrates a true closed-loop self-correction for robot planning. This system enables robots to actively monitor and adjust their task execution.
- Our proposed system operates on a hierarchical structure, addressing both high-level and low-level planning errors. This hierarchical capability allows it to correct errors across different levels of task planning, thereby enhancing its overall robustness and adaptability.
- Through extensive experiments, we showcase the remarkable advantages of our proposed framework. Our results highlight the significant performance improvements achieved by integrating self-correction into the robotic planning process. We present evidence of our system's superiority in various benchmarks, underscoring its potential in the future research.

II. Related Works

Robot Task Planning with LLMs. Integrating Large Language Models (LLMs) into robotic task planning has garnered significant attention in recent years. In the field of robotics research adding natural language, experts progressed in the areas of skill learning [19]–[22], control performance [23]–[25], collaboration [26], human-robot interaction [27], navigation [28], etc. Further, researchers have explored the potential of LLMs to understand high-level instructions and generate low-level step-by-step actions that can be executed by agents directly from textual inputs [9]–[11], [29]. SayCan [11] employs LLMs to complete embodied tasks by combining pre-trained skills and learned value functions to choose contextually appropriate actions. Code as Policies (CaP) [10] employs a structured code generation approach for robot control, enabling iterative incorporation of all necessary functionalities. VoxPoser [30] utilizes 3D Value Maps for robotic manipulation, employing code modules generated by LLMs to plan and control robots. This method allows for re-planning in case of task failures, enhancing the system's robustness. Inner Monologue [31] investigates how LLMs in embodied context can use natural language feedback to form an inner monologue, improving their processing and planning in robotic control scenarios. ProgPrompt [9] introduces a programmatic LLM prompt structure that enables functional plan generation in diverse robotic environments, accommodating various robot capabilities and tasks. The methods mentioned above demonstrate the significant potential of LLMs in the field of robot task planning.

Task Failure Correction with LLMs. Despite the potential of integrating LLMs in robotics, there are occasions when robots face challenges in executing LLM-generated actions. These challenges encompass cases where actions generated by LLMs cannot be performed due to their impracticality or unsuitability for the given context. Additionally, unexpected environmental changes can further hinder task execution by making the upcoming actions infeasible. In response to these failure situations, researchers have explored methods to rectify such infeasible actions and address the disruptive impact of unforeseen factors on robotic task execution. REFLECT [17] is a framework that employs LLMs to automatically detect and correct robot failures using past experiences. However, failure correction cannot commence until all processes have concluded, thus impeding the correction of errors occurring at a specific step of task execution. Similarly, DoReMi [32] presents a framework that employs a set of error action constraints to resolve discrepancies between planning and execution. Nonetheless, it predominantly targets navigation tasks. CLAIRify [18] leverages iterative prompting, integrates error feedback and employs program verification to produce syntactically valid robot task plans from high-level natural language instructions. While it excels in error correction,

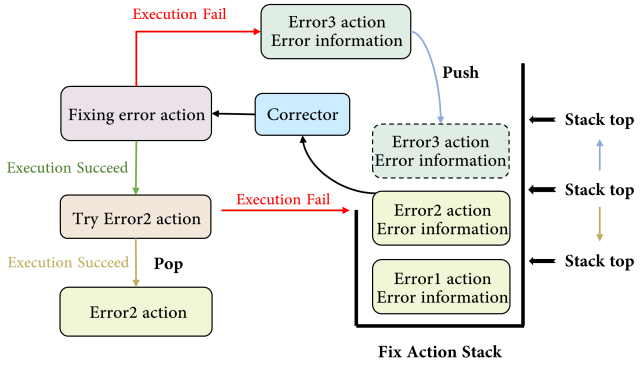

Fig. 3. Illustration of feedback stack structure. The corrector learns the error action and error information from the top of the stack, and then determines the appropriate action to fix the error. If the fixing error action cannot be successfully executed, it is added as a new item to the Fix Action stack. Otherwise, the error action at the top of the stack is attempted, with the goal of removing it from the stack.

CLAIRify specializes in generating long-horizon task plans and may not correct tasks on a step-by-step basis without completing all the programs. Similarly, corrective re-prompting proposed in [33] enhances task plans by injecting pre-condition error information into LLMs but faces challenges in correcting single-step task plans. In contrast, our proposed framework offers broader applicability by hierarchical self-correction in both long and short-horizon task planning scenarios.

Beyond the field of robotics, there is research on failure correction with LLMs in various other domains, including agent-based systems and code generation. Reflexion [34] is a framework that enhances language agents' learning in goal-driven tasks by utilizing linguistic feedback. In the code generation area, Self-Debugging [35] teaches LLMs to debug the predicted program by few-shot demonstrations, even without any feedback on the code error messages. Compared with these methods, HiCRISP focuses on robot controlling and task planning by integrating closed-loop and hierarchical self-correction.

III. Methodology

In this section, we explain how we model the system which is composed mainly of LLM and robot. LLM acts as robot's controller, planning task sequence and determining executive action. Robot, as an anchor for interacting with the environment, executes commands from LLM. Combined with perceptive information, we propose a framework named HiCRISP which can fix itself by automatic feedback(Fig 2).

A. Problem Description

1) Task Planning Formulation: Describing the behavior of our LLM-controlled system entails using a conceptual model akin to a transition system. Here, we model task planning with a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{S}_0, \mathcal{E}, \psi >$. Specifically, $\mathcal{S}$ represents a set of environment states, with $\mathcal{S}_0$ designating the initial state. $\mathcal{A}$ is a set of actions. Moreover, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ denotes the transition relationship between states. $\mathcal{E}$ denotes the prompts, i.e., the input provided to LLMs, such as all the objects available in the environment and error messages from the environment (if any). ψ represents the perception detector.

According to the previously definition, LLMs serve as task planners, responsible for generating actions for the entire task or correcting actions for the current step, as guided by the prompts. For the former, the process can be formulated as:

$$\text{LLM}(\mathcal{S}_0, \mathcal{E}, task) = \{a_i | i = 0, 1, 2, \cdots, n\} \tag{1}$$

where $\text{LLM}(,,)$ denotes the LLM task planner and $a_i \in \mathcal{A}$ represents the generated actions.n represents the length of action sequence. $\mathcal{E}$ contains feasible functions in state $\mathcal{S}_0$, object items in the environment and few-shot examples of sample tasks and plans. $task$ is the input from user. In this way, a chain transition system is determined:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_{n+1} \tag{2}$$

where $s_i \in \mathcal{S}$ denotes the states. s_0 and s_{n+1} represent the initial state and the final success state, respectively. However, we may encounter situations where actions cannot be executed. In such circumstances, error correction mechanisms come into play to address and resolve the issues, allowing for the successful execution of the actions. Specifically, when state s_i executes action a_i but fails to transfer to s_{i+1} as intended, we employ a closed-loop approach that combines perceptual information ψ with the current state information s_{error} (which may be identical to s_i). This process aids in automatically generating corrective actions through the LLM, as depicted by the equation:

$$\text{LLM}(s_{\text{error}}, \mathcal{E}, \psi) = a_{\text{correction}} \tag{3}$$

where $a_{\text{correction}}$ represents the rectified action. This equation encapsulates the mechanism by which the LLM leverages prior information, perceptual input, and current state data to execute corrective actions and rectify errors in each step.

2) Detecting Failures at High and Low Levels: Within the framework of our transition system, the transition from one state $s \in \mathcal{S}$ to the next state $s' \in \mathcal{S}$, induced by the execution of action $a \in \mathcal{A}$, is denoted as $s \xrightarrow{a} s'$. Throughout this transition, our perception module, denoted as ψ, plays a pivotal role in the detection of two distinct types of failures: high-level state failures and low-level action failures.

1) **High-Level State Failure**: In this scenario, the perception module ψ may indicate that actions generated by LLMs are executable, yet the system fails to transition correctly to the intended subsequent state s'. In such cases, we categorize this type of transition failure as a high-level state failure, thereby triggering our high-level feedback process.
2) **Low-Level Action Failure**: Alternatively, when LLM generated action a cannot be performed successfully, the perception module ψ provides feedback indicating a failure in executing action a. Consequently, this disrupts the sequence of transitions from the current stage to the subsequent state. We classify this particular type of transition failure as a low-level action failure, leveraging our low-level feedback mechanism.

Note that both types of failures can occur simultaneously, especially in complex environments. In such cases, we first examine and deduce the low-level error report. Following confirmation of the accuracy of the low-level status, we proceed to infer the high-level error report as necessary.

B. High-level Feedback

Given the constraints of all available information and the inherent uncertainty of future environmental conditions, high-level state failure occurs sometimes, leading to task plans change in order for system to operate continuously. In this situation, we propose high-level feedback. For actions $a_i, i \in \{0, \cdots, n\}$, when the perception module ψ detects that the direct successor of s_i does not match s_{i+1}, our system initiates a closed-loop feedback loop. We consolidate language prompts by combining error messages and current status information, which is then fed back into the LLM. The LLM generates a rectified action $a_{\text{correction}}$. By executing $a_{\text{correction}}$, the current state is directed towards transitioning into $s_{\text{correction}}$, satisfying the condition: $s_{\text{correction}} \xrightarrow{a_i} s'$ where the state s' obtained by performing the same action a_i is in the original transition chain. Mathematically, the entire process can be described as follows: when a transition system undergoes $s \xrightarrow{a_i} s_{\text{error}}$, our framework adjusts it back to s' in the original transition chain, represented as:

$$s \xrightarrow{a_i} s_{\text{error}} \xrightarrow{a_{\text{correction}}} s_{\text{correction}} \xrightarrow{a_i} s' \tag{4}$$

It should be noted that the $\xrightarrow{a_{\text{correction}}} s_{\text{correction}}$ segment in Eq. (4) probably consist of multiple fragments in order to bridge state s_{error} and s'. Moreover, state s_{error} and $s_{\text{correction}}$ can be identical to s and s', respectively.

C. Low-level Feedback

When dealing with real robots or robot simulations within a physical environment, actions generated by LLMs tend to operate at such a high level that they often surpass the understanding of robotic systems. For instance, LLM-generated actions, such as object manipulation, encompass multiple intricate control tasks like positioning or trajectory planning. Moreover, the real-world environment introduces numerous unforeseen deviations and noise interference, potentially perturbing the execution of sub-steps within these high-level actions and leading to task failures. To address the above issues, we propose a low-level feedback mechanism.

Within the low-level feedback system, we first introduce an Action Translation Module, a critical component designed to bridge the gap between LLM-generated high-level actions and the comprehensible low-level control signals required by robotic systems. For example, the Action Translation Module can translate the LLM-generated action `Move to Area 1` into a series of sub-steps encompassing intricate trajectory planning. By integrating low-level feedback, we establish a dynamic feedback loop that harnesses error information and deviations, detected by the perception detector, and feeds this invaluable data back to the controller. This iterative process not only drives continuous improvements but also significantly elevates the precision and accuracy of task execution.

Algorithm 1: HiSCRIP

Input: User input commands: $task$
Output: Observation data from perception module;

$\{a_i\}_{i=0}^{n} = LLM(\mathcal{S}_0, \mathcal{E}, task)$;
for a_i *in* $\{a_i\}_{i=0}^{n}$ **do**
  Execute a_i; Perception returns $flag$, $info$;
  if *not* $flag$ **then**
    Initialize $stack$; $stack.push([a_i, info])$;
    for $i = 1, \cdots, threshold$ **do**
      ψ = stack.top.info;
      $a_{correction} = LLM(\mathcal{S}_{error}, \mathcal{E}, \psi)$;
      Execute $a_{correction}$;
      Perception returns $flag$, $info$;
      if $flag$ **then**
        for $stack.depth > 0$ **do**
          a_{try} =stack.top.action;
          Execute a_{try};
          Perception returns $flag$, $info$;
          if $flag$ **then**
            $stack.pop$;
          else
            $stack.update([a_{try}, info])$;
            break;
        end
      else
        $stack.push([a_{correction}, info])$;
    end
end

D. Feedback Structure

When the feedback mechanism is activated, the originally sequenced primary program is halted. Continuing its execution is highly likely to result in the failure of subsequent tasks due to incorrect timing continuation. Therefore, the immediate task at hand must be resolved first to establish a sound foundation for long-term tasks. The feedback structure employs a stack-based approach, as shown in Fig. 3, adhering to the "first in, last out" principle. Stack elements encompass executed actions and the associated error causes. Whenever an action generated by the LLMs is executed erroneously, both the action itself and the error information are bundled and stored in the stack. Subsequently, when successful execution occurs, the stack is employed to revisit and reattempt the action which is at the top of the stack, initiating recursive operations. In conjunction with Section III-B, the entire process can be summarized in Algorithm 1.

It should be noted that, at the bottom of the stack resides a_i, the action responsible for the initial failure, along with the corresponding error messages. The action a_i can lead to both $s \xrightarrow{a_i} s_{\text{error}}$ and $s_{\text{correction}} \xrightarrow{a_i} s'$. Therefore, when the stack becomes empty, it signifies that the state has transitioned to s', indicating that the previously encountered error has been successfully addressed. Meanwhile, there exists a limitation on the stack's depth. Once the stack capacity is exceeded or the number of accumulated elements surpasses a predefined

Fig. 4. HiCRISP demonstrates superior performance in a VH simulator when compared to ProgPrompt [9]. In situations where the agent loses its position on the sofa and is unable to execute the command "`Sit sofa`," HiCRISP exhibits the capability to learn from the error message and take corrective action by executing the command "`Find sofa`." This corrective action not only addresses the error but also guides the agent to reach the desired final state.

TABLE I

PROGPROMPT AND HICRISP PERFORMANCE ON THE VH TEST-TIME TASKS.

Task Description	$\|A\|$	PSR(ProgPrompt) [9]	Precision(ProgPrompt) [9]	Exec(ProgPrompt) [9]	PSR(HiCRISP)	Precision(HiCRISP)	Exec(HiCRISP)
Eat chips on the sofa	6	0.12 ± 0.05	1.00 ± 0.00	0.71 ± 0.00	0.33 ± 0.17	1.00 ± 0.00	0.71 ± 0.00
Put salmon in the fridge	7	0.00 ± 0.00	0.99 ± 0.00	0.80 ± 0.00	0.00 ± 0.00	0.99 ± 0.00	0.90 ± 0.00
Watch TV	6	0.63 ± 0.00	1.00 ± 0.00	0.89 ± 0.00	0.73 ± 0.10	1.00 ± 0.00	1.00 ± 0.00
Wash the plate	10.5	0.00 ± 0.00	1.00 ± 0.00	0.96 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Bring coffeepot and cup-cake to the coffee table	8	0.00 ± 0.00	0.99 ± 0.00	0.71 ± 0.00	0.00 ± 0.00	0.99 ± 0.00	0.71 ± 0.00
Microwave salmon	11.5	0.00 ± 0.00	1.00 ± 0.00	0.80 ± 0.00	0.00 ± 0.00	0.99 ± 0.01	0.87 ± 0.00
Turn off light	3	0.00 ± 0.00	1.00 ± 0.00	0.67 ± 0.00	0.00 ± 0.00	0.99 ± 0.00	1.00 ± 0.00
Brush teeth	11	0.17 ± 0.00	1.00 ± 0.00	0.77 ± 0.00	0.18 ± 0.00	1.00 ± 0.00	0.82 ± 0.00
Throw away apple	5	0.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Make toast	10	0.16 ± 0.00	0.94 ± 0.00	0.75 ± 0.00	0.15 ± 0.00	0.94 ± 0.00	0.75 ± 0.00

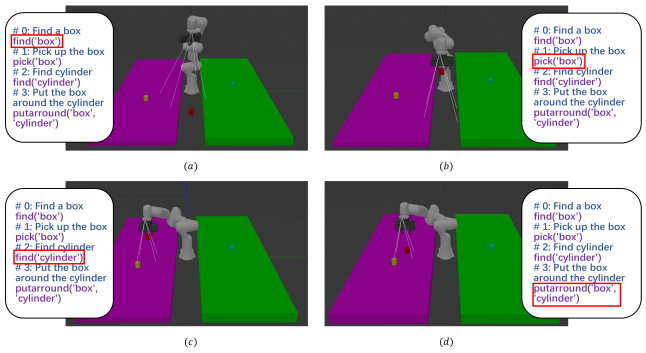

Fig. 5. HiCRISP acts in Gazebo Engine. LLM breaks down the input: "`Place a box around a cylinder`" into four steps: (1) locate a box; (2) pick up the box; (3) find a cylinder; (4) place the box around the cylinder.

threshold, the feedback correction process is terminated. In this manner, while the error may remain unrectified, it ensures that the system's state remains finite and avoids the risk of the entire system falling into a perpetual loop of ineffectual corrections.

IV. SIMULATION AND EXPERIMENTS

To verify the effectiveness of our approach across diverse scenarios and objects, we conducted a series of experiments including both virtual simulations and real-world settings. This section provides detailed insights into high-level task planning (Section IV-A), overall system operation (Section IV-B), and real-world experiments (Section IV-C). Our results demonstrate the framework's versatility, showcasing its applicability across various objects and scenarios. All experiments were conducted using GPT-3.5, with Gazebo simulations built on ROS Noetic and Ubuntu 20.04 as the underlying infrastructure.

A. Virtual Home Simulation

First, we validate our HiCRISP on the Virtual Home Environment [36] which contains a wealth of everyday scenes and objects. Here, we focused on assessing the effectiveness of our high-level planner. In the simulation environment, the agent executed actions based on the LLM-generated output, which occasionally results in task planning failure. Through a series of mission tests, compared with Progprompt [9], our proposed framework effectively addressed these challenges. As an illustrated example in Fig. 4, consider the command input "`Eat chips on the sofa`", where the planning and state check prompts are identical, ensuring planning consistency. Progprompt follows the pre-generated subtasks but faces failure while executing the "`Sit sofa`" script. The essential reason for this failure is the wrong scheduling order in the task schedule. Progprompt loses the "sofa" position after executing the "`Grab chips`". In HiCRISP, after the "`Grab chips`", the system receives the error message and combines it into the feedback branch. By taking actions called "`Find sofa`", illustrated in Fig. 4, the process of the task regains its possibility to transit into the final state. Table I illustrates the results of evaluation matrix by using Progprompt and HiCRISP respectively.

Note that, we perform two iterations for each task in both methods and take the average. The main focus is on improving the execution rate. However, the remaining two metrics are derived according to the standards provided by ProgPrompt [9], and their values are constrained by the evaluation method stated in ProgPrompt.

B. Gazebo Simulation

To bridge the divide between simulation and reality, we develop a platform for robotic arm control in the Gazebo

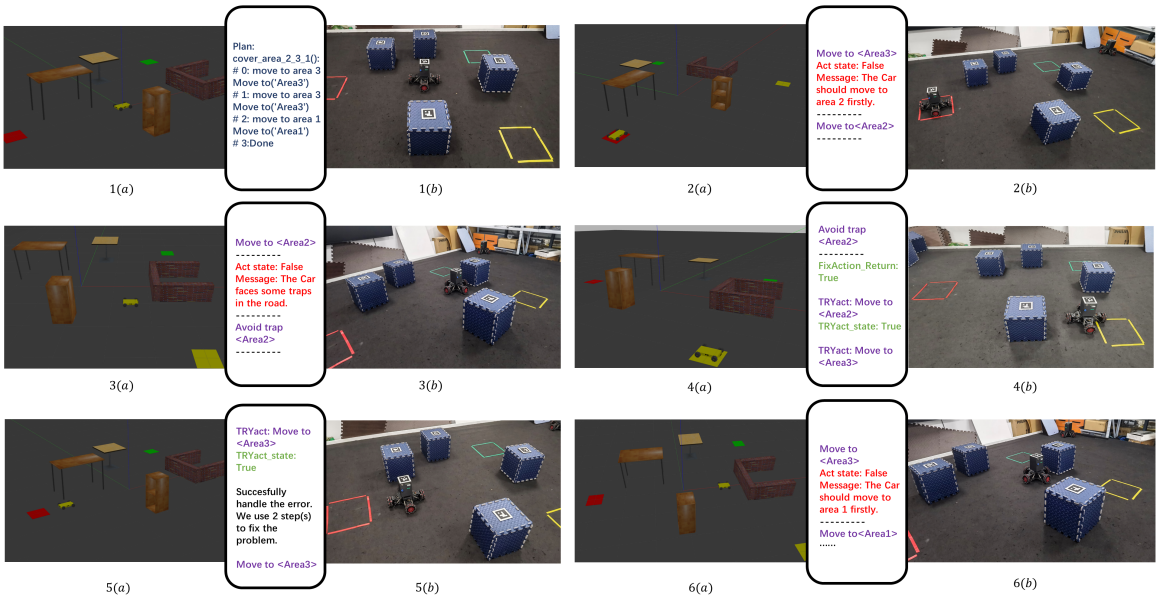

Fig. 6. HiCRISP operates on both the Gazebo engine and real-world AGV platform. We command the vehicle to sequentially approach specific landmarks according to a predetermined order. Action failure arises when obstacles obstruct the intended trajectory. Planning failure occur if the vehicle erroneously navigates towards an incorrect landmark. HiCRISP addresses these failures by providing corresponding corrective actions and rectifying the issues.

physical engine, renowned for its realism. This approach surpasses the use of natural language environments, as it enables us to closely simulate real-world scenarios.

To assess our framework's effectiveness, we perform pick-and-place tasks with the assistance of a Panda robot. As outlined in Section III-C, we utilize the Rapidly-Exploring Random Trees (RRT) algorithm to generate precise motion trajectories, facilitating the seamless translation of target names into corresponding control trajectories.

In our specific experiments, the perception module provides detailed perception information to facilitate feedback conditions. To account for potential deviations in perception and abstract order ambiguity, we introduced a controlled random offset to the Panda robot's actuator, adhering to a fixed Gaussian distribution. This offset intentionally increases the likelihood of not meeting the conditions when the Panda robot executes the "place" action. The conditions for state transitions are automatically verified by the detector. During simulated experiments, when the detector issues an error message: "The box is not in close proximity to the cylinder", LLM promptly analyzes the received error message and current status information and determines the correct course of action for self-correction. In this particular situation, LLM may prescribe the command "`Pick box`" as the box has become disengaged from the robot's actuator. Additionally, the distance between the actual box position and the cylinder position, as provided by the Detector, is fed back through the low-level feedback loop to correct the mapping relationship in the action decoder. Low-level feedback significantly enhances action execution, enabling the robot to perform more effectively in subsequent attempts. This low-level feedback mechanism has proven particularly valuable in scenarios where repeating the same action would otherwise have a high likelihood of failure, ultimately leading to improved action success rates.

C. Real world experiment

To further validate our framework, we conducted AGV experiments in both Gazebo simulations and real-world environments, utilizing a multi-robot test bed and indoor AGV systems [37]. As illustrated in Fig. 6, the AGV was tasked with reaching different areas in a specified sequence. When the AGV inadvertently approached incorrect areas, the system detected the errors and generated corrective action plans, as depicted in Fig. 6. Operating in an unfamiliar environment with limited perceptual range, unexpected obstacles along the planned path occasionally led to action failures. Through low-level feedback, the action encoder adapted to new constraints and generated revised trajectories. The results of these experiments demonstrate that the HiCRISP framework can be widely applied in robotics.

V. Conculsion

In this paper, we introduce HiCRISP, a hierarchical robot control framework that restructures task planning and addresses erroneous messages, whether at the high level or low level, during robot missions via a feedback structure, thereby enhancing overall mission performance. By synergistically merging the feedback structure with the merits of LLM, HiCRISP exhibits the prowess to adeptly manage a wide array of tasks while possessing the ability to self-correct. Prospective areas of research could encompass augmenting the prowess of the perception module or delving into comprehensive and theoretically substantiated evaluations of system properties.

REFERENCES

[1] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[3] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[4] R. Taylor, M. Kardas, G. Cucurull, T. Scialom, A. Hartshorn, E. Saravia, A. Poulton, V. Kerkez, and R. Stojnic, "Galactica: A large language model for science," *arXiv preprint arXiv:2211.09085*, 2022.

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[6] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *arXiv preprint arXiv:2305.10601*, 2023.

[7] A. Fiske, P. Henningsen, and A. Buyx, "Your robot therapist will see you now: ethical implications of embodied artificial intelligence in psychiatry, psychology, and psychotherapy," *Journal of medical Internet research*, vol. 21, no. 5, p. e13216, 2019.

[8] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, p. 20, 2023.

[9] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.

[10] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and 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.

[11] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*. PMLR, 2023, pp. 287–318.

[12] P. A. Jansen, "Visually-grounded planning without vision: Language models infer detailed plans from high-level instructions," *arXiv preprint arXiv:2009.14259*, 2020.

[13] S. Li, X. Puig, C. Paxton, Y. Du, C. Wang, L. Fan, T. Chen, D.-A. Huang, E. Akyürek, A. Anandkumar, *et al.*, "Pre-trained language models for interactive decision-making," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 199–31 212, 2022.

[14] R. Patel and E. Pavlick, "Mapping language models to grounded conceptual spaces," in *International Conference on Learning Representations*, 2021.

[15] R. OpenAI, "Gpt-4 technical report," *arXiv*, pp. 2303–08 774, 2023.

[16] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser, "Tidybot: Personalized robot assistance with large language models," *arXiv preprint arXiv:2305.05658*, 2023.

[17] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," in *7th Annual Conference on Robot Learning*, 2023.

[18] M. Skreta, N. Yoshikawa, S. Arellano-Rubach, Z. Ji, L. B. Kristensen, K. Darvish, A. Aspuru-Guzik, F. Shkurti, and A. Garg, "Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting," *arXiv preprint arXiv:2303.14100*, 2023.

[19] T. Xiao, H. Chan, P. Sermanet, A. Wahid, A. Brohan, K. Hausman, S. Levine, and J. Tompson, "Robotic skill acquisition via instruction augmentation with vision-language models," in *Workshop on Language and Robotics at CoRL 2022*, 2022.

[20] H. Ha, P. Florence, and S. Song, "Scaling up and distilling down: Language-guided robot skill acquisition," in *7th Annual Conference on Robot Learning*, 2023.

[21] L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg, "Concept2robot: Learning manipulation concepts from instructions and human demonstrations," *The International Journal of Robotics Research*, vol. 40, no. 12-14, pp. 1419–1434, 2021.

[22] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, *et al.*, "Language to rewards for robotic skill synthesis," *arXiv preprint arXiv:2306.08647*, 2023.

[23] Y. J. Ma, W. Liang, V. Som, V. Kumar, A. Zhang, O. Bastani, and D. Jayaraman, "Liv: Language-image representations and rewards for robotic control," *arXiv preprint arXiv:2306.00958*, 2023.

[24] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *7th Annual Conference on Robot Learning*, 2023.

[25] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2motion: From natural language instructions to feasible plans," in *ICRA2023 Workshop on Pretraining for Robotics (PT4R)*, 2023.

[26] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," *arXiv preprint arXiv:2307.04738*, 2023.

[27] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, "Interactive language: Talking to robots in real time," *IEEE Robotics and Automation Letters*, 2023.

[28] D. Shah, B. Osiński, S. Levine, *et al.*, "Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Conference on Robot Learning*. PMLR, 2023, pp. 492–504.

[29] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[30] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," in *7th Annual Conference on Robot Learning*, 2023.

[31] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *6th Annual Conference on Robot Learning*, 2022.

[32] Y. Guo, Y.-J. Wang, L. Zha, Z. Jiang, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," *arXiv preprint arXiv:2307.00329*, 2023.

[33] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex, "Planning with large language models via corrective re-prompting," *arXiv preprint arXiv:2211.09935*, 2022.

[34] N. Shinn, B. Labash, and A. Gopinath, "Reflexion: an autonomous agent with dynamic memory and self-reflection," *arXiv preprint arXiv:2303.11366*, 2023.

[35] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *arXiv preprint arXiv:2304.05128*, 2023.

[36] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[37] X. Ding, H. Wang, H. Li, H. Jiang, and J. He, "Robopheus: A virtual-physical interactive mobile robotic testbed," *arXiv preprint arXiv:2103.04391*, 2021.