

Are We Close to Realizing Self-Programming Robots That Overcome the Unexpected?

Jair A. BOTTEGA¹, Takashi TSUBOUCHI¹, Xinyue RUAN², and Akihisa OHYA¹

Abstract—Are we on the verge of developing robots that can reprogram themselves to overcome unexpected situations? As robotic systems and artificial intelligence continue to evolve, the concept of self-programming robots capable of adaptive reasoning seems to be becoming a tangible reality. This paper demonstrates how Large Language Models (LLMs), specifically the OpenAI *o1-preview* model, can empower mobile robots to autonomously analyze failures and modify their operational code in real time. Traditional robot programming, bound by the need to anticipate all possible scenarios, often leads to rigid behaviors when faced with unforeseen obstacles. We present novel results where a robot, tasked with navigating a predefined path, encounters an unanticipated obstacle and autonomously generates modified code to address the challenge, including obstacle avoidance and dynamic path planning strategies. By utilizing a structured prompt and advanced reasoning capabilities, the robot moves beyond pre-programmed limitations, embodying a new level of dynamic, self-adaptive autonomy. This work highlights the transformative potential of LLMs in robotics, offering a glimpse into a future where robots not only perform tasks but also learn, adapt, and evolve autonomously in complex, real-world environments.

I. INTRODUCTION

In recent years, the integration of Large Language Models (LLMs) into robotics has opened new horizons for autonomous reasoning and decision-making capabilities in robotic systems [1], [2]. Traditional robot programming often requires exhaustive anticipation of all possible scenarios a robot might encounter, leading to rigid and inflexible behaviors when faced with unforeseen obstacles or situations. This limitation poses significant challenges in dynamic environments where unpredictability is the norm.

The problem of enabling robots to autonomously adapt their behavior and modify their code when encountering unexpected situations is crucial for the advancement of truly autonomous systems. In our previous work [3], we posed the open question of whether robots themselves could generate programs for their actions using AI when unexpected events occur—events that were not anticipated by the original programmer. While humans naturally rethink and devise new plans when their initial approach fails, empowering robots with a similar level of adaptive reasoning remains a significant challenge.

Recent advancements in LLMs have significantly enhanced their reasoning capabilities. For instance, Kojima et al. [4] demonstrated that LLMs can function as effective

zero-shot reasoners by incorporating simple prompts such as “Let’s think step by step,” which significantly improves performance on complex reasoning tasks. Building upon these developments, the newly released OpenAI *o1-preview* model introduces native chain-of-thought (CoT) reasoning through training with reinforcement learning, enabling more sophisticated and robust reasoning processes. The *o1* family models are designed to spend more time contemplating before responding, which allows them to solve complex problems in science, coding, and mathematics with high accuracy [5]. These advancements in LLM reasoning capabilities are crucial for their integration into robotic systems, enabling robots to autonomously analyze and modify their code in response to unexpected challenges.

In this paper, we present novel results demonstrating how LLMs can empower a mobile robot to autonomously reason and reprogram its code to overcome unexpected challenges encountered during task execution. By structuring prompts effectively, we enabled the robot to analyze its failure to achieve the goal and generate modified code that includes new functions necessary for addressing the encountered problem.

An illustrative example is provided where a robot is tasked with moving from point X to point Y but encounters a stationary obstacle not anticipated in its original programming. Using an LLM, the robot reasons about the situation, identifies the cause of failure, and autonomously generates modified code incorporating obstacle avoidance strategies and dynamic path planning. Such a level of reasoning and autonomous code generation was, until recently, not expected by the majority of roboticists within their lifetimes. The advent of generative AI has made this significant advancement possible, bringing forth capabilities previously considered far-fetched.

Our approach differs from previous works as it empowers the robot to not only make decisions or ask for help but to autonomously reason and modify its operational code in real-time to achieve its objectives without human intervention. This capability represents a substantial step towards truly autonomous robotic systems capable of handling a wide range of unexpected challenges in real-world environments.

The key contributions of this paper are:

- **Demonstrating LLM-driven Autonomous Code Modification:** We show how LLMs can be leveraged to enable robots to autonomously analyze failures and generate new code to overcome unforeseen challenges.
- **Advancing Autonomous Reasoning in Robotics:** Our results highlight the potential for LLMs to enhance the

¹ Faculty of System and Information Engineering, University of Tsukuba, 1-1-1 Tennoudai, Tsukuba, Ibaraki 305-8573 Japan
tsubo@roboken.iit.tsukuba.ac.jp

² Persol Cross Technology Co. Ltd, Japan

reasoning capabilities of robots beyond decision-making to include dynamic reasoning and autonomous reprogramming in response to various types of problems.

- **Bridging the Gap in Current Research:** While existing studies have explored LLMs in code generation and decision-making, our work specifically addresses the autonomous modification of robot code in response to unanticipated challenges during task execution, a capability not extensively explored in prior research.

The remainder of this paper is organized as follows. Section II reviews related work in LLM applications for robotics. Section III details the methodology and experimental setup. Section IV presents the results and discusses the implications of autonomous code generation in robotics. Finally, Section V concludes the paper and outlines future research directions.

II. RELATED WORKS

Recent years have witnessed a surge in research exploring the intersection of LLMs and robotics, particularly in code generation and decision-making applications. Projects like *Voyager* [6] utilize an iterative prompting mechanism to generate executable code for autonomous exploration and skill acquisition in the game Minecraft. While effective in a simulated environment, *Voyager* does not fully address the complexities of real-world robotic systems.

Similarly, *Code as Policies* [2] investigates using code-generated policies to control the behavior of autonomous agents, demonstrating experiments with real robots. This work successfully leverages LLMs to generate code that calls preprogrammed functions, enabling robots to execute tasks specified in natural language. However, it does not specifically address the robot’s ability to autonomously overcome unexpected challenges by modifying its own code beyond predefined behaviors.

In the realm of robotic learning and task execution, projects like *Auto-RT* [7] and *RT-Hierarchy* [8] leverage LLMs to enhance robotic capabilities. *Auto-RT* focuses on real-time automated reasoning to improve robotic decision-making, while *RT-Hierarchy* studies hierarchical decision-making to enhance real-time processing. These studies contribute to robotic learning and policy improvement but do not specifically tackle the issue of autonomous code modification in response to unforeseen situations during task execution.

Human-robot interaction studies such as *Say-Can* [9] and *Inner Monologue* [10] have advanced the integration of natural language understanding with robotic actions, allowing robots to execute complex verbal commands and explain their decisions. *Robot Help* [11] enables robots to ask for assistance when facing unsolvable problems, relying on human intervention rather than autonomous problem-solving. While these works enhance the interaction between humans and robots, they do not empower robots to autonomously modify their code to overcome challenges.

Research efforts have also been directed toward programming robots using natural language instead of traditional coding. *Alchemist* [12] introduces an LLM-aided end-user development platform for robot applications, enabling users



Fig. 1. “Yamabico BeeGo” Mobile Robot

to program robots using natural language commands. Similarly, Sikorski et al. [13] utilize NLP and LLM techniques to control mobile robots at the edge, demonstrating the feasibility of using language models for robotic control. While these approaches showcase innovative uses of LLMs in robotics, their primary focus is on simplifying robot programming for humans rather than enhancing the robot’s autonomous decision-making and self-programming capabilities.

In the context of robots responding to unforeseen events, Kuniyoshi et al. [14] propose leveraging intrinsic dynamics and perturbations to enhance a humanoid robot’s ability to recover from falls—an unusual situation not covered by specific program codes. This approach improves the robot’s capacity to respond to certain situations without explicit program descriptions but does not involve the use of Large Language Models (LLMs) or autonomous code modification. Instead, it relies on pre-existing dynamics and control strategies, assuming that such events are possible within the robot’s designed capabilities.

Our work distinguishes itself by empowering robots to autonomously analyze failures and modify their operational code in real-time to overcome unforeseen challenges without human intervention. By leveraging LLMs, we enable robots to reason dynamically across different types of problems, a capability not extensively explored in prior research.

III. METHODOLOGY

A. Experimental Setup

1) *Robot Platform:* The experiments were conducted using the *Yamabico BeeGo*, a member of the Yamabico family of mobile robots [16]. The BeeGo is equipped with a 360-degree LiDAR sensor and two motor actuators controlling its wheels, enabling it to navigate within a defined environment.

2) *Software Environment:* The robot is programmed using ROS 1, which provides the necessary middleware for handling sensor data, motor control, and inter-node communication. The initial behavior of the robot is illustrated in Algorithm 1, which outlines the state transitions and obstacle handling mechanisms using a structured pseudocode approach.

B. Task Description

1) *Objective:* The primary objective of the experiment is to evaluate the robot’s ability to autonomously modify its operational code to overcome unexpected obstacles that were

Algorithm 1 Robot Navigation: Basic Obstacle Handling

```
1: Initialize:  
   Position  $(x, y) \leftarrow (0.0, 0.0)$   
   Orientation  $yaw \leftarrow 0.0$   
   State  $\leftarrow$  MOVING  
2: while robot is operational do  
3:   DetectObstacle  
4:   CheckGoal  
   state MOVING  
5:   if ObstacleDetected then  
6:     SendCommand(Stop)  
7:     State  $\leftarrow$  STOP  
8:      $stop\_time \leftarrow$  current_time  
9:     Log("Obstacle detected. Stopping.")  
10:  else if AtGoal then  
11:    SendCommand(Stop)  
12:    State  $\leftarrow$  ATGOAL  
13:    Log("Arrived at goal.")  
14:  else  
15:    SendCommand(FollowLine)  
16:  end if  
   state STOP  
17:  if not ObstacleDetected then  
18:    SendCommand(MoveStraight)  
19:    State  $\leftarrow$  MOVING  
20:    Log("Obstacle cleared. Resuming movement.")  
21:  end if  
22:  if  $stopped\_time > 20$  sec then  
23:    Log("Stop timeout at position  $(x, y)$ ."  
24:  end if  
   state ATGOAL  
25:  Terminate  
26: end while
```

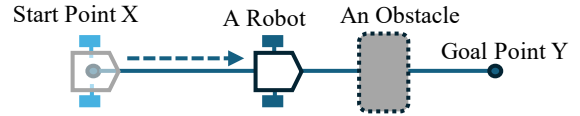
not anticipated by the original programming. Specifically, the robot is tasked to move from a starting point X to a goal point Y within a specified time duration.

2) *Scenario Setup:* The experimental environment is a straight-line path from point X to point Y , with an obstacle deliberately placed between them to obstruct the robot's passage, as depicted in Figure 2. The initial code does not incorporate obstacle avoidance strategies; instead, it is designed to halt the robot when an obstacle is detected and resume movement once the obstacle is removed.

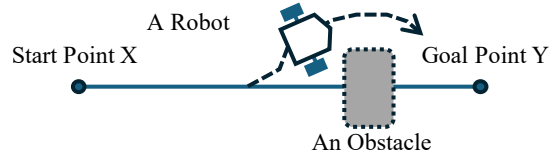
C. Failure Induction and Data Collection

1) *Inducing Failure:* To simulate an unexpected event, an obstacle was introduced along the robot's predefined path. The initial program was executed, causing the robot to stop upon detecting the obstacle. Since the code lacks obstacle avoidance mechanisms and the obstacle is not removed, the robot remained stationary, eventually leading to a timeout after 20 seconds.

2) *Data Logging:* During the failed execution, the following data were collected:



(i) After starting and moving straight for a while, an obstacle stays in front of the robot and the robot stops. The obstacle continues staying. The given time duration expires, which is unexpected situation.



(ii) After the unexpected situation were recognized, a new program were regenerated for the robot to reach the goal and it moved.

Fig. 2. A Schematic Illustration of the Problem

- **Code Parameters:** Values of key parameters such as starting point X , destination point Y , and timeout duration.
- **Execution Logs:** Information about the line where code execution halted and the final state of the robot.
- **Robot States:** Initial and final positions.

This data was essential for providing the AI model with comprehensive information to analyze the failure and suggest appropriate code modifications.

D. Prompt Engineering for AI Assistance

1) *Structured Prompt Design:* A structured prompt was crafted to facilitate the AI model's understanding and reasoning. The prompt included detailed sections to ensure that the AI had all necessary information to analyze the failure and generate modified code. The prompt structure is as follows:

```
You are a mobile robot equipped with a  
360-degree LiDAR sensor and an odometry  
system. Your objective is: {{Objective  
Description}}. You have the capability  
to autonomously reprogram your code when  
something unexpect occurs and you cannot  
achieve a goal.
```

```
## Command Description  
**Objective:** {{Objective Description}}
```

```
## Executed Code  
``plaintext  
{{Executed Code}}  
```
```

```
Goal Completion Status
Status: {{Status}}
```

```
Execution Log
- **Logged Messages:** {{Execution Log
Messages}}
```

```

- **Code Execution Terminated At:**
 {{Termination Point}}
- **Robot Current Position:** {{(Z_x, Z_y)}}

Code Parameters
- **Starting Point:** {{(X_x, X_y)}}
- **Destination Point:** {{(Y_x, Y_y)}}
- **Timeout Duration:** {{t}} seconds

Available Functions
- {{Function List}}

Instruction: Analyze why you did not
 achieve your objective. Subsequently,
 propose modifications to your code to
 ensure successful completion of the
 specified task. You may define new
 functions as necessary. Please rewrite
 your code and include the implementation
 of any new functions you introduce,
 using the plaintext tag for the code
 section.

```

This prompt was designed to be autonomous and generalized, allowing it to be adaptable for various unexpected scenarios beyond the current experiment. The use of the ‘plaintext’ tag is particularly important as it clearly separates the generated code from other outputs, making it easier to identify and extract the code directly from the AI model’s response for implementation.

#### E. AI Model Utilization

1) *Model Selection:* The experiment utilized the newly released OpenAI *o1-preview* model, which is designed to exhibit enhanced reasoning capabilities by spending more time processing and refining its responses. In the *o1* models, the temperature parameter is fixed at 1 and cannot be adjusted by users. This design choice aims to enhance the models’ reasoning capabilities by allowing them to explore a broader range of possible responses, which is particularly beneficial for complex problem-solving tasks. According to OpenAI’s technical research, *o1-preview* demonstrates superior performance in complex reasoning tasks, coding, and problem-solving compared to its predecessors [5].

2) *Rationale for Choice:* The *o1-preview* model was chosen for its advanced reasoning and coding abilities, making it well-suited for the task of autonomously analyzing failures and generating effective code modifications. Its capacity to handle intricate tasks aligns with the requirements of enabling a robot to adapt its behavior in real-time. In our tests, no previous OpenAI models were capable of solving this problem effectively with the general approach of the structured prompt given, reinforcing the need for the enhanced capabilities of the *o1-preview* model.

#### F. Autonomous Code Modification Process

Upon encountering a failure, where the robot stopped due to an unexpected obstacle, the collected data—including execution logs, code parameters, and the robot’s final state—was structured into the prompt and submitted to the *o1-preview* model. After reasoning and analyzing the failure, the AI

provided an output containing a modified piece of code. This code was then introduced into the robot as a new ROS node and executed immediately to address the encountered obstacle.

Due to the unavailability of the *o1-preview* model via the API, the generated code was manually incorporated into the robot’s system as a new ROS node. This allowed the robot to resume its task and navigate around the obstacle autonomously. Future plans include automating this integration once the *o1-preview* model becomes available via the API.

The developed code, results, and demonstration videos are available at the following repository: [https://github.com/RXY-iit/robot\\_overcome](https://github.com/RXY-iit/robot_overcome).

## IV. RESULTS

This section presents the outcomes of our experiments, demonstrating the robot’s ability to autonomously reason about failures and modify its operational code using the *o1-preview* Large Language Model (LLM). The chosen use case, explained in the methodology, served as a practical example to assess the LLM’s capabilities. The results highlight how structured prompting enabled the *o1-preview* model to generate successful, context-specific code modifications tailored to this scenario, showcasing the model’s advanced reasoning capabilities and the effectiveness of AI-driven autonomous programming in real-world applications.

### A. Autonomous Reasoning Capabilities

Upon encountering an unexpected stationary obstacle during its mission to navigate from point  $X$  to point  $Y$ , the robot utilized the structured prompt to autonomously analyze the failure without any human intervention. The structured prompt provided the *o1-preview* model with comprehensive data, including execution logs, robot states, and executed code parameters. Despite receiving only this information, the robot effectively diagnosed the issue, showcasing its capability to understand and reason about failures independently.

### B. Advanced Reasoning with the *o1-preview* Model

The *o1-preview* model demonstrated superior reasoning capabilities, as evidenced by its detailed chain of thought generated within 11 seconds. The model systematically broke down the problem by:

- Analyzing the executed code to identify shortcomings in obstacle handling.
- Evaluating the robot’s state transitions and pinpointing the lack of obstacle avoidance logic.
- Proposing strategic modifications, including the introduction of new functions for dynamic path planning and obstacle avoidance.

This structured reasoning process, which operates independently of the model’s final output, culminated in a comprehensive analysis and a set of actionable code modifications. This approach highlights the model’s ability to autonomously engage in complex problem-solving tasks, demonstrating its capacity to break down challenges systematically, propose

effective solutions, and generate code that addresses specific obstacles encountered during task execution.

### C. Code Generation and Integration

After the chain of thought process, the *o1-preview* model's final output was provided, during around 4 seconds. The final output is shown below:

#### Analysis:

The robot did not achieve its objective of moving from point X to point Y because it encountered an obstacle at position (1.93, 0.00) and stopped. The current code only stops the robot when an obstacle is detected but does not have any obstacle avoidance behavior to navigate around the obstacle and continue towards the goal.

#### Modifications:

To ensure successful completion of the task, we need to implement obstacle avoidance behavior. When an obstacle is detected, the robot should adjust its path to navigate around the obstacle while still moving towards the goal. We can achieve this by:

- **Implementing an Obstacle Avoidance Function:**
  - Introduce a new function `avoid_obstacle()` that calculates a new direction to steer the robot away from the obstacle.
  - Use the laser scan data to determine the safest direction to proceed.
- **Modifying the Main Loop:**
  - Replace the stop behavior with the obstacle avoidance function when an obstacle is detected.
  - Ensure the robot resumes normal navigation towards the goal once the obstacle is avoided.
- **Adjusting Control Parameters:**
  - Fine-tune the gains and limits in the `follow_line()` function to improve navigation performance.

#### Corrected Code:

[[ROS Node Script: 309 Lines of C++ Code]]

#### Explanation of Modifications:

[[Detailed Explanation of the Code Changes]]

The generated code was extracted from the output and integrated into the robot's system by creating a new ROS node. This node incorporated the AI-generated obstacle avoidance and dynamic path planning logic, enabling the robot to autonomously navigate around unforeseen obstacles, as described in Algorithm 2. Observe the new created function to avoid obstacles.

### Algorithm 2 Generated Enhanced Code

```

1: Initialize:
 Position $(x, y) \leftarrow (0.0, 0.0)$
 Orientation $yaw \leftarrow 0.0$
 State \leftarrow MOVING
 ObstacleDirection \leftarrow None
2: function AVOID_OBSTACLE
3: Input: LaserScanData
4: Process: Determine direction of closest obstacle
5: if Obstacle is to the left then
6: Adjust $yaw \leftarrow yaw - \delta$
7: Log("Obstacle on left. Turning right.")
8: else if Obstacle is to the right then
9: Adjust $yaw \leftarrow yaw + \delta$
10: Log("Obstacle on right. Turning left.")
11: else
12: Adjust $yaw \leftarrow yaw + \delta \triangleright$ Default to turning left
13: Log("Obstacle ahead. Turning left.")
14: end if
15: SendCommand(SetYaw, yaw)
16: SendCommand(MoveForwardSlow)
17: end function
18: while robot is operational do
19: DetectObstacle
20: CheckGoal
 state MOVING
21: if ObstacleDetected then
22: Call avoid_obstacle()
23: else if AtGoal then
24: SendCommand(Stop)
25: State \leftarrow ATGOAL
26: Log("Arrived at goal.")
27: else
28: SendCommand(FollowLine)
29: end if
 state ATGOAL
30: Terminate
31: if NoObstacleDetected and State = MOVING then
32: SendCommand(FollowLine)
33: end if
34: end while

```

### D. Performance Evaluation

1) *Initial Failure with Original Code:* In the initial run, the robot executed its original code, which lacked obstacle avoidance mechanisms. As expected, upon encountering the stationary obstacle, the robot transitioned to the STOP state and remained stationary until the timeout occurred after 20 seconds. This outcome validated the limitations of the original programming in handling unexpected scenarios where obstacles are not removed.

2) *Comparison of AI Models:* To evaluate the effectiveness of autonomous code modification, we conducted experiments using both the *o1-preview* and GPT-4o models. Each model was tested in 10 trials where it generated code

intended to enable the robot to navigate around the obstacle and reach the goal. The results are summarized in Table I.

TABLE I  
PERFORMANCE COMPARISON OF AI MODELS

| Model             | Compiled | Reached Goal | Successful and Safe |
|-------------------|----------|--------------|---------------------|
| <i>o1-preview</i> | 100%     | 90%          | 30%                 |
| GPT-4o            | 60%      | 0%           | 0%                  |

3) *Analysis of Results:* For the *o1-preview* model:

- **Code Compilation:** All generated codes compiled successfully (100%).
- **Navigation Success:** The robot reached the goal in 9 out of 10 trials (90% success rate).
- **Obstacle Interaction:** In 60% of the trials, the robot slightly touched the obstacle. This issue arises from the model’s lack of awareness of the robot’s physical dimensions. The distances are measured relative to the LiDAR sensor, not the robot’s body, which might lead to slight collisions. This problem could potentially be resolved by providing the LLM with additional context about the robot’s physical structure or by adjusting the obstacle detection method.
- **Goal Handling:** In 40% of the trials, the robot passed through the goal without stopping. This error is influenced by biases in the original human-written code, where the execution loop does not terminate correctly after the robot reaches the goal. While the *o1-preview* model attempted to address this imperfection, it occasionally did not publish a stop command to the `/cmd_vel` topic before breaking the execution loop, which caused the robot to pass through the target without stopping.

For the GPT-4o model:

- **Code Compilation:** Only 6 out of 10 generated codes compiled successfully (60%).
- **Navigation Success:** The robot did not reach the goal in any of the trials (0% success rate).
- **Overall Performance:** The model was unable to produce effective code modifications to enable the robot to navigate around the obstacle. In one of the generated codes, the robot accelerated to a high velocity in a straight line after avoiding the obstacle. This behavior demonstrated unsafe operation due to the unreliability of the model’s reasoning.

These results highlight the superior reasoning and code generation capabilities of the *o1-preview* model compared to GPT-4o. While the *o1-preview* model demonstrated a high success rate, the observed unsafe behaviors emphasize the importance of testing the generated code in a controlled environment before deploying it in real-world applications.

### E. Summary of Findings

The experimental results confirm that:

- ***o1-preview* Model Effectiveness:** The *o1-preview* model successfully generated executable code in all attempts, with a 90% success rate in enabling the robot to reach the goal.

- **Limitations Observed:** Despite the high success rate, the robot occasionally touched obstacles and sometimes failed to stop at the goal, indicating the need for more precise control logic.
- **GPT-4o Model Limitations:** The GPT-4o model showed limited effectiveness, with only 60% code compilation success and no successful navigation to the goal.
- **Importance of Advanced Reasoning:** The superior performance of the *o1-preview* model underscores the significance of advanced reasoning capabilities in autonomous code modification for robotics.

These findings demonstrate the potential of using advanced LLMs for enabling robots to autonomously overcome unexpected challenges, while also highlighting areas for future improvement.

### F. DISCUSSION

**Limitations and Considerations:** The current experiment was limited to a single type of unexpected event—a stationary obstacle. While the results are promising, further testing in diverse environments and with various unforeseen challenges is necessary to fully evaluate the capabilities of autonomous code modification. Moreover, both models demonstrated the potential to generate unsafe behaviors. These risks could be mitigated by implementing an intermediary step, such as running the generated code in a simulation before deployment, or by using a more advanced model capable of reasoning more effectively and preemptively addressing potential issues the robot may encounter.

Furthermore, the lack of reliability in current state-of-the-art (SOTA) models underscores the need for careful assessment as newer and more powerful models are released. Although advancements are promising, these models are not yet reliable enough to consistently generate safe and robust code, making safety and reliability critical challenges for enabling widespread adoption of autonomous code generation in robotics.

**Ethical and Safety Concerns:** Unlike traditional robotics applications, which are extensively tested in simulations and controlled environments before deployment, our approach executes newly generated code immediately without prior testing or safety validation. This raises significant safety and trust issues, as the only validation is performed by the AI itself. In scenarios where self-programming robots are deployed publicly, potential unsafe behaviors pose ethical dilemmas: who bears responsibility if harm occurs—the AI company, the robot developer? Or even the end-user, whose actions or commands may have inadvertently led the robot to behave unsafely? This question becomes particularly complex given that well-aligned AI models might still generate unsafe code under unforeseen conditions.

**Enhancing Robot Comprehension with Multimodal AI:** While our primary approach relies on text-based reasoning, we conducted a showcase experiment to explore the potential of incorporating multimodal AI models. In this experiment, the robot was tasked with the goal: “Go to the first floor.”

During execution, the robot encountered an issue where it could not proceed. By providing the robot with a series of images captured during execution (see Figure 3) and the execution log messages, the robot analyzed the images and logs to determine the cause of the failure. It successfully identified that the elevator was under repair, as indicated by a sign on the elevator door in Japanese.

The robot then generated a voice response to the human operator:

*"The service elevator is under repair, as indicated by the sign on the door. I will now search for an alternative elevator to proceed to the first floor. Please stand by while I reassess the environment."*



Fig. 3. Images captured by the robot during execution, showing the elevator under repair.

This demonstrates that by incorporating visual data into its reasoning process, the robot can better understand complex situations and adapt its behavior accordingly. While current multimodal AI models may not yet match the reasoning capabilities of the *o1-preview* model, this experiment highlights the potential benefits of integrating multimodal inputs for enhancing robotic comprehension and autonomy. Rapid advancements suggest future models will likely integrate these functionalities, allowing robots to identify and respond to a wider range of complex, real-world scenarios.

**Future Trade-offs:** Balancing reasoning speed and performance will remain a critical trade-off as models evolve. The development of faster and more capable reasoning models could mitigate current delays, enabling more responsive and adaptable robotic systems.

## V. CONCLUSION

In this paper, we have demonstrated that Large Language Models (LLMs), specifically the OpenAI *o1-preview* model, can empower mobile robots to autonomously analyze failures and modify their operational code to overcome unexpected challenges. By leveraging structured prompts and the advanced reasoning capabilities of the *o1-preview* model, we enabled a robot to navigate around unforeseen obstacles without human intervention. This approach represents a significant advancement in robotic autonomy, moving beyond pre-programmed behaviors to dynamic, self-adaptive systems capable of handling a variety of real-world scenarios.

Our findings highlight the potential of integrating LLMs into robotic systems for autonomous code modification and

reasoning. While the results are promising, further research is encouraged to generalize this approach to more complex and diverse environments, as well as to address the ethical and safety considerations associated with autonomous self-programming robots. Exploring the incorporation of multimodal AI models could enhance environmental understanding, and developing robust frameworks for safe and trustworthy robot behavior in dynamic, unpredictable settings remains a crucial area for advancement. Future research could focus on implementing new reasoning models within APIs to test robots' capabilities for self-reprogramming in more complex environments, thereby enabling them to encounter and address a wider range of unexpected situations.

## REFERENCES

- [1] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, "Large language models for robotics: A survey," arXiv preprint arXiv:2311.07226, 2023.
- [2] 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," arXiv preprint arXiv:2209.07753, 2022.
- [3] T. Tsubouchi, S. Obayashi, and X. Ruan, "Can a Robot Regenerate New Programs by Itself with the Help of a Generative AI when Unexpected Events Occur? — an Open Issue Proposal —," in *Proc. ICRA@40*, 2024, to appear.
- [4] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22199–22213, 2022.
- [5] OpenAI, "OpenAI o1 System Card," OpenAI, Sept. 12, 2024. [Online]. Available: [https://assets.ctfassets.net/kftzwdyauwt9/67qJD51Aur3eIc96iOfeOP/71551c3d223cd97e591aa89567306912/o1\\_system\\_card.pdf](https://assets.ctfassets.net/kftzwdyauwt9/67qJD51Aur3eIc96iOfeOP/71551c3d223cd97e591aa89567306912/o1_system_card.pdf)
- [6] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. J. Fan, and A. Anandkumar, "Voyager: An Open-Ended Embodied Agent with Large Language Models," arXiv:2305.16291v2 [cs.AI], 2023.
- [7] M. Ahn, D. Dwibedi, C. Finn, et al., "AutoRT: Embodied Foundation Models for Large Scale Orchestration of Robotic Agents," arXiv:2401.12963, 2024.
- [8] S. Belkhale, T. Ding, T. Xiao, et al., "RT-Hierarchy: Action Hierarchies using Language," arXiv:2403.01823, 2024.
- [9] M. Ahn, A. Brohan, N. Brown, et al., "Do As I Can and Not As I Say: Grounding Language in Robotic Affordances," arXiv preprint arXiv:2204.01691, 2022.
- [10] W. Huang, F. Xia, T. Xiao, et al., "Inner Monologue: Embodied Reasoning through Planning with Language Models," arXiv preprint arXiv:2207.05608, 2022.
- [11] J. Liang, F. Xia, W. Yu, et al., "Learning to Learn Faster from Human Feedback with Language Model Predictive Control," arXiv:2402.11450, 2024.
- [12] U. B. Karli, J.-T. Chen, V. N. Antony, and C.-M. Huang, "Alchemist: LLM-Aided End-User Development of Robot Applications," in *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24)*, ACM, New York, NY, USA, pp. 361–370, 2024.
- [13] P. Sikorski, L. Schrader, K. Yu, L. Billadeau, J. Meenakshi, N. Mutharasan, F. Esposito, H. AliAkbarpour, and M. Babaiahi, "Deployment of NLP and LLM Techniques to Control Mobile Robots at the Edge: A Case Study Using GPT-4-Turbo and LLaMA 2," arXiv:2405.17670v2 [cs.RO], 2024.
- [14] Y. Kuniyoshi, Y. Ohmura, K. Terada, T. Yamamoto, and A. Nagakubo, "Exploiting the Global Dynamics Structure of Whole-Body Humanoid Motion—Getting the 'Knack' of Roll-and-Rise Motion," in *Robotics Research. The Eleventh International Symposium*, Springer Tracts in Advanced Robotics, vol. 15, Springer, Berlin, Heidelberg, pp. 357–364, 2005.
- [15] W. Yu, N. Gileadi, C. Fu, et al., "Language to Rewards for Robotic Skill Synthesis," arXiv preprint arXiv:2306.08647, 2023.
- [16] <https://www.roboken.iit.tsukuba.ac.jp/en/theme/>