Autonomous Search with DeepSeek Planner in ROS

Chanutda Manosorn[1] and Jehyun Kim[1]

[1] Griffith University, Southport QLD, Australia

Abstract. The paper presents an autonomous robotic system integrating a large language model (LLM) planner, DeepSeek, within a ROS Noetic simulated Gazebo house for search and detect operations. Using Turtlebot3 Waffle Pi, the robot navigates autonomously along waypoints generated by DeepSeek and attempts to capture an image upon successful triggering. While the navigation follows planned trajectories using odometry, the robot failed to correctly detect the target object (a bowl), as the DWA local planner treated the bowl as an obstacle and altered the path before detection could be confirmed. As a result, the planner receives no feedback indicating that the bowl was found, and no photo command was issued; consequently, the take_photo_node was not triggered. The overall system highlights the potential of integrating DeepSeek-based planning in robotic systems, while also indicating key limitations in perception-planning feedback. Future work could include better object perception modelling and adaptive prompt design for improving effectively synchronized communication in real world applications.

Keywords: ROS Noetic, DeepSeek, TurtleBot3, Autonomous Navigation

1 Introduction

Recent catastrophic events—including a powerful earthquake in Myanmar and the collapse of a building under construction in Thailand—have resulted in numerous casualties and left many survivors trapped beneath rubble [1]. These events highlighted the need for autonomous robotic systems capable of supporting search-and-rescue operations in hazardous or inaccessible areas. A conceptual design is proposed to simulate such an initial system in a controlled environment, focusing on autonomous navigation, object detection, and visual reporting.

To address this goal, a robotic platform is developed to operate within a simulated household environment in Gazebo. The system is designed to explore indoor spaces, detect specific objects (represented by bowls), and capture images upon discovery. Additional constraints, such as limited battery power and an unknown number of targets, are incorporated to emulate real-world challenges in search-and-rescue scenarios.

Two publicly available robotics projects are selected as the primary foundations for this work. The DeepSeek-ROS2 project [2] demonstrates how large language models (LLMs) can convert natural language into ROS 2 terminal commands. Meanwhile, the TurtleBot repository [3] also provides practical examples of mobile

robot behavior under ROS. Building upon these works, the current system adapts their ideas to a ROS Noetic environment using the TurtleBot3 Waffle Pi model with an extended objective of integrating autonomous visual reporting.

In this implementation, DeepSeek is employed as a high-level planner that receives environmental context and generates waypoint-based navigation plans. A custom `take_photo_node` is also implemented, enabling the robot to save images when an object is detected. The robot operates with the following key capabilities:

1) Follows navigation plans generated by DeepSeek autonomously
2) Takes a photo upon detecting a bowl and returns to the base
3) Monitors its battery level and returns to the charging dock if energy is low
4) Continues to operate under uncertainty about target quantity and location

This work aims to explore how DeepSeek can be integrated into ROS to support autonomous object search and returning a photo under energy constraints, particularly in environments where human access is limited or risky.

2 Related Work

Several prior studies and open-source projects have explored autonomous robotics in both structured and unstructured environments.

2.1 LLM-Based Planning and Command Integration

Two open-source projects provide practical foundations for integrating large language models (LLMs) into robotic systems. The DeepSeek-ROS2 project [2] showcases a system where LLMs convert user instructions into terminal-level ROS 2 commands. These commands are formatted as single-line instructions for direct robot actuation. Although effective in translating natural language into robot actions, this architecture focuses primarily on stateless command execution and lacks intermediate decision-making or feedback loops. Complementing this, the TurtleBot repository by Silliman [3] includes beginner-friendly scripts demonstrating image capture, basic movement, and topic publishing in ROS 1. In particular, the `take_photo.py` node offers a simple yet practical approach for capturing images from the robot's onboard camera—an idea that is directly adapted into the `take_photo_node.py` used in the present work.

Together, these references influence the system architecture developed in this project, which separates planning, execution, and sensing into distinct nodes. In contrast to fixed command execution, the proposed design leverages DeepSeek as a high-level planner capable of generating waypoint-based movement plans, while executor nodes interpret and carry out those plans dynamically during runtime.

2.2 **Autonomous Planning in Rescue Robotics Systems**

Autonomous robots designed for search-and-rescue (SAR) tasks have been explored extensively in both academic literature and field deployments. Delmerico et al. [4] provide a comprehensive review of robotic systems used in SAR contexts, covering ground, aerial, and underwater platforms. The review highlights key challenges in sensor fusion, navigation, mission planning, and communication under disaster conditions. While it does not focus specifically on LLM-based planning, it offers valuable insights into the requirements of deployable SAR systems.

A more concrete implementation is presented by Wang et al. [5], who developed a two-robot SAR system capable of autonomous 3D mapping, object recognition, and physical manipulation in an underground building. Their fully furnished system includes a tracked exploration robot and a secondary robot with a 6-DOF robotic arm for door opening and object retrieval. Additionally, the system supports remote teleoperation and multi-node wireless communication. Although their design is based on model-driven planning rather than LLMs, the architectural goals share similarities with the language-model-based coordination proposed in this work.

Although the study operates within a simulated Gazebo environment and does not include concrete robotic manipulation, it draws inspiration from these works by addressing autonomy, perception, and goal-directed behavior. The key contribution lies in replacing model-based planning with an LLM.

2.3 **Low-Cost Rescue Robot Prototypes**

Pathak et al. [6] introduced a low-cost mobile rescue robot equipped with basic sensors such as infrared, fire detection, and ultrasonic modules, along with Zigbee-based wireless communication. The system also integrates an Android application to provide visual feedback. While it presents potential for human detection in disaster zones, the architecture lacks autonomy, planning capabilities, or integration with ROS. This highlights a gap in lightweight, flexible systems that can operate semi-independently—an area this project seeks to explore through LLM-driven planning within a scalable ROS framework.

3 Proposed Approach

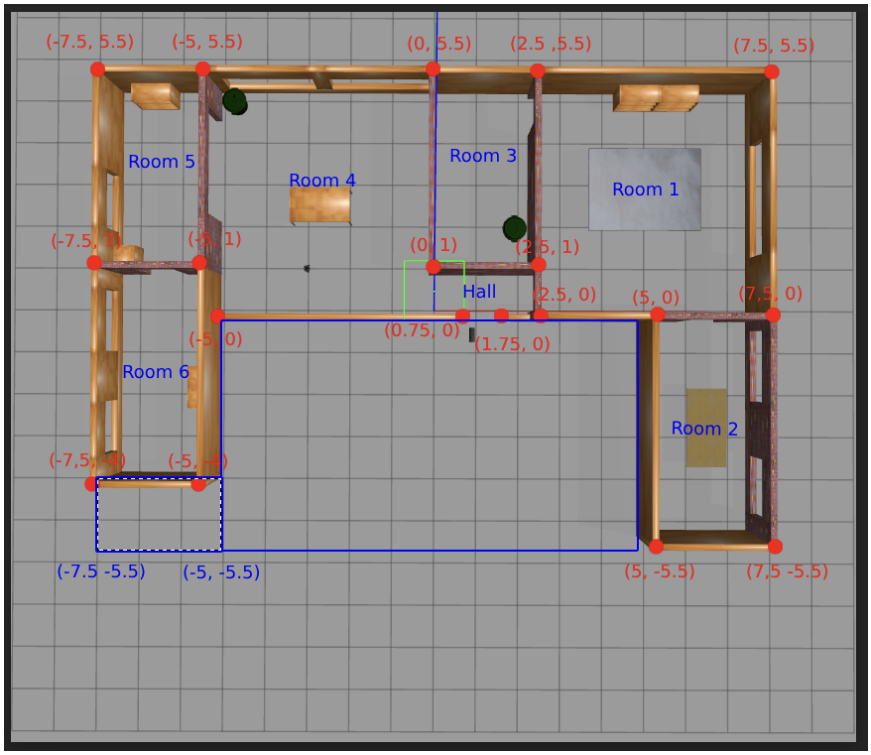

Fig. 1. Top-down view of the simulated house with room labels and restricted area.

The proposed approach enables a TurtleBot3 robot to autonomously navigate a simulated indoor environment, locate randomly placed objects (bowls), and take photos of them as instructed by a cloud-based language model (DeepSeek). The system integrates conventional ROS-based control and localization with high-level planning powered by DeepSeek, achieving a hybrid architecture that allows both reactive navigation and task-based decision-making.

The robot operates within a structured house environment composed of six labeled rooms and a central hall. Each room contains randomly placed target objects, and a restricted zone is defined to ensure safe navigation. The robot's behavior alternates between exploration and execution of instructions generated by DeepSeek based on its current pose.

3.1 Environment Modelling and Sensors

The simulation environment replicates a single-floor house consisting of six labeled rooms (Room 1 to Room 6) and a central hallway. To better align with the RViz top-down view, the house layout is rotated 90 degrees clockwise. Red-labeled points on the

simulation map mark the corners of each room, and the coordinate system uses the ROS map frame convention (x-axis rightward, y-axis upward).

A restricted area, spanning from (-5.0, -5.5) to (5.0, 0.0), is defined both visually and within the planning logic to prevent the robot from entering unsafe zones. Bowls are spawned at random positions within each room according to predefined boundaries in the ROOM_LOCATION dictionary. This ensures variability across simulations while preserving spatial constraints.

A charging station labeled "charger" is placed near Room 6 at coordinates (-4.5, 1.25). The robot begins at this location, and its AMCL pose is initialized by a separate script. Obstacle avoidance and navigation rely on odometry (velocity and position) and are handled through the Dynamic Window Approach (DWA), without relying on Lidar. The robot periodically sends its pose to the DeepSeek planner to receive high-level commands.

3.2 Control System Architecture

Table 1. Function of ROS nodes in the system

Module	ROS Node	Function
Planning	Deepseek Planner	Generates waypoint plans using natural language + state
Command Execution	Command Listener/Executor	Interprets planner commands, manages battery & photo logic
Movement	move_base	Controls robot navigation based on goal commands
Perception	take_photo_node	Captures images when instructed (i.e. when a bowl is found)
Simulation	Gazebo + Environment	Simulates robot world and physical feedback

Although the system is implemented on a single robot, its architecture distributes tasks across several functionally distinct modules (nodes), each responsible for a specialised function such as planning, control, perception, or monitoring. These nodes operate collaboratively toward a shared goal, forming a multi-agent system in the context defined for this project.

3.3 Workflow Overview

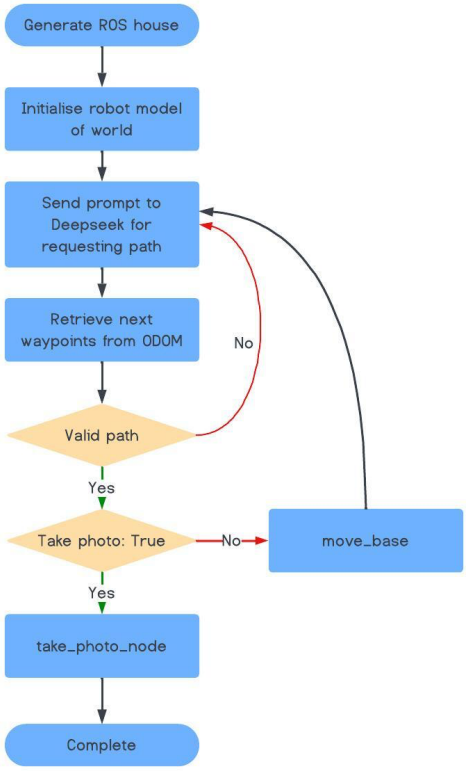

Fig. 2. Overall workflow of DeepSeek-based robot navigation and photo task execution

The system integrates high-level planning, autonomous navigation, and perception using modular ROS nodes as shown in Figure 2. DeepSeek generates movement plans which are executed by the robot via move_base, while image data is streamed from the turturbot3 Waffle pi's camera to capture using take_photo_node. The planner triggers the photo capture node, which saves an image using real-time camera input.

3.4 Integration of ROS and Deepseek

The robot's low-level controls (sensors, motors, state updates) are handled in ROS. DeepSeek takes care of high-level planning and decision-making. By keeping these separate, the system is easier to test and extend. Later on, it would be possible to swap in a smarter model for the LLM without changing the core robot logic.
Overall, this project shows how a robot can follow instructions like "explore and take pictures of bowls" using both sensors and external planning from an LLM.

4 Implementation and Execution

This system is implemented using ROS Noetic with TurtleBot3 Waffle Pi in a Gazebo-simulated house environment. The project integrates an LLM-based planner using DeepSeek with a modular ROS architecture, enabling the robot to explore rooms, detect specific objects (bowls), take photos upon detection, and return to the charging station if energy is low. The components and their interactions are summarized as below.

4.1 Robot Operating System (ROS)

ROS serves as the main framework to control the robot's behaviour. Three core ROS nodes are implemented:

1) Planner Node (`deepseek_planner_node.py`): Communicates with the DeepSeek API to obtain high-level movement plans or task commands based on context and robot status (i.e. battery level, object detection).
2) Command Executor (`ros_command_listener.py`): Executes movement goals using `move_base`, monitors battery depletion over time, detects objects via OpenCV, and handles photo capture requests.
3) Photo Node (`take_photo_node.py`): Listens to image messages from the camera and saves an image to disk when called.
4) The robot is spawned and initially set at a fixed location near the charging dock position(pos_x:-4.5, pos_y:1.25). A separate script sets the robot's initial AMCL pose. Obstacle avoidance and navigation rely on odometry (velocity and pose) from Dynamic Window Approach and do not depend on Lidar. The robot follows waypoints from DeepSeek based on odom feedback.

4.2 DeepSeek-based planner

DeepSeek serves as a high-level planner by converting structured prompts into lists of waypoints. A custom prompt template includes movement parameters, prohibited zones, and action rules (i.e. send `take_photo` upon bowl detection). The planner and executor communicate using messages over ROS topics, allowing structured status updates (i.e. battery, bowl detection) and commands (i.e. movement, photo trigger).

4.3 Gazebo Simulation and Object Spawning

The simulation takes place in a predefined house environment using Gazebo. Objects such as bowls are randomly spawned in different rooms using `spawn_bowls.py`, while a power charger is placed close to the initial position (-4.5, 1.25). The YAML map file `house_map3.yaml` defines the world for navigation and SLAM.

4.4 System Execution Flow

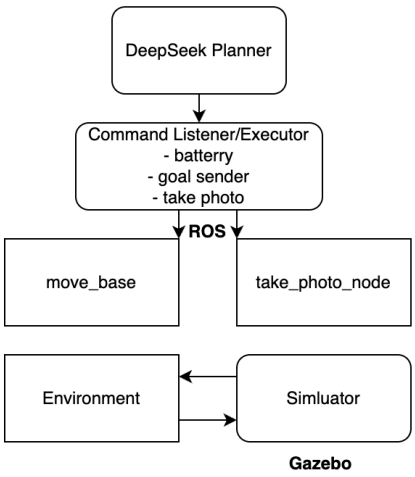

Fig. 3. System Diagram: LLM Planner with ROS-Based Execution

Figure 3 shows the system consisted of ROS nodes for SLAM, environment perception, planning, and execution. The DeepSeek Planner generates high-level commands, while the Command Listener/Executor node handles battery monitoring, movement, and triggering photo capture. The robot operates autonomously, following a typical execution sequence outlined below:

1) Startup: Launch Gazebo environment and set the robot's initial pose.
2) Spawning: Randomly place bowls and a charger using a model spawn service.
3) Planning: The planner requests DeepSeek for navigation waypoints.
4) Execution: Navigation goals are executed via ‘move_base’, which interprets waypoints and drives the robot accordingly using odometry feedback.
5) Detection: Upon detecting a bowl, a photo is captured.
6) Return: If the battery is low or a bowl is found, the robot returns to the origin. After taking a photo, the executor publishes a 'Done’ as ‘True' signal, which triggers the planner to generate a return-to-origin command.

4.5 Execution

To run the full simulation, the system must be compiled and launched in multiple terminals:

1) Terminal 1: Launch the Gazebo environment and set the robot’s initial position using roslaunch turtlebot3_gazebo turtlebot3_house.launch x_pos:= -4.5 y_pos:=1.25.
2) Terminal 2: Launch the simulation framework and system nodes with roslaunch deepseek_world_search custom_navigation.launch.

3) Terminal 3: Spawn random bowls and place the charger using rosrun deepseek_world_search spawn_bowls.py.
4) Terminal 4: Start the command listener node that executes movement, handles battery status, and triggers photo capture using rosrun deepseek_world_search ros_command_listener.py.
5) Terminal 5: Launch the DeepSeek planner node with rosrun deepseek_world_search deepseek_planner.py.

5 Experiments and Testing

This section presents the results of step-by-step testing on 3 versions of the system tuning, each reflecting a design refinement based on the observed limitations. The experiments are conducted in the same simulated environment using Gazebo and TurtleBot3, and evaluated in terms of navigation behavior, goal completion, and system response.

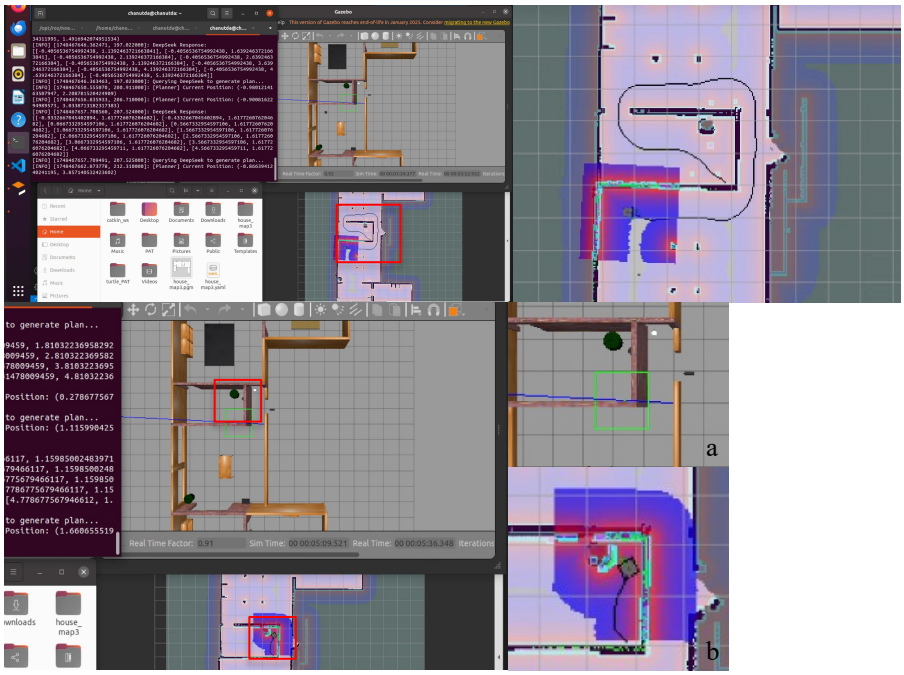

Fig. 4. Experiment of deepseek_bowl: waypoints and unable to escape from corner

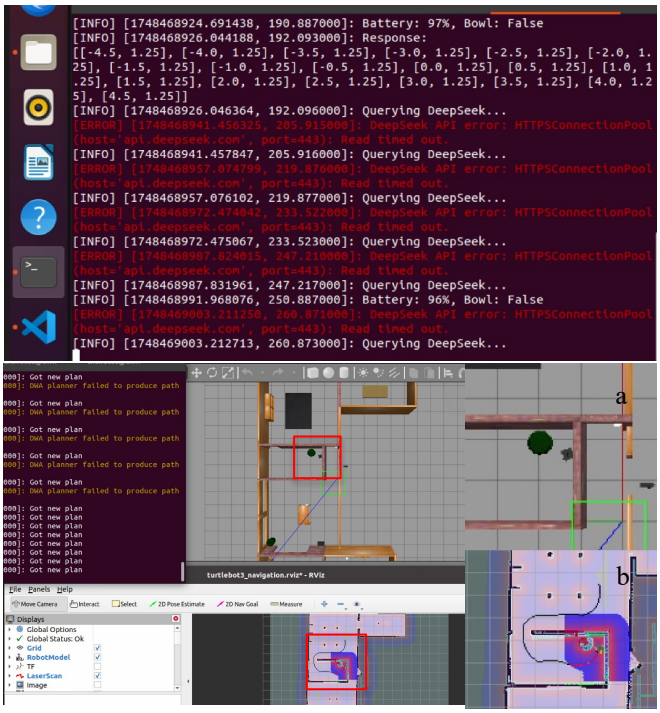

Fig. 5.Experiment of deepseek_world_search: timeout issues and improved avoidance

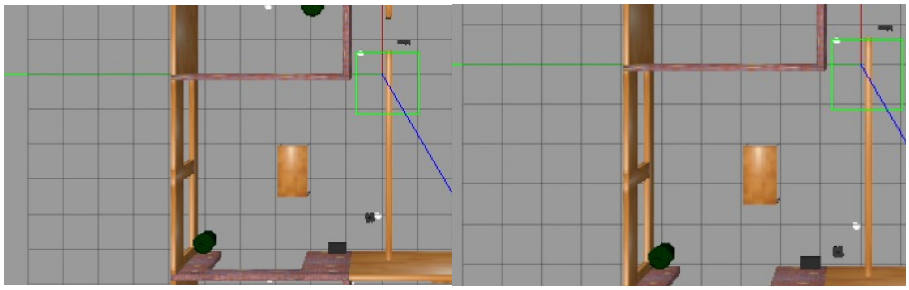

Fig. 6. Experiment of deepseek_world_search v2: ran into a bowl but not recognized as a target

5.1 Testing on deepseek_bowl

Figure 4 shows the initial test with deepseek_bowl. Top right image: The robot successfully begins movement following the planned path from Deepseek. Bottom right images (a and b): The robot becomes stuck in a corner, repeatedly requesting new plans but odometry is unable to progress.

5.2 Testing on deepseek_world_search

Figure 5 shows the test with deepseek_world_search. The top image: In deepseek_world_search, frequent API timeouts from Deepseek result in navigation pauses and delayed decisions. Bottom right images (a and b): After refining parameters, the robot demonstrates improved obstacle avoidance and maintains smoother navigation paths through constrained areas.

5.3 Testing on deepseek_world_search after refinement

Figure 6 shows the test using deepseek_world_search v2. Left: the robot detects a bowl in front but does not treat it as a photo target. Right: Instead, the bowl is interpreted as an obstacle. As a result, the robot navigates around it without triggering the take_photo action.

5.4 Testing on take_photo_node

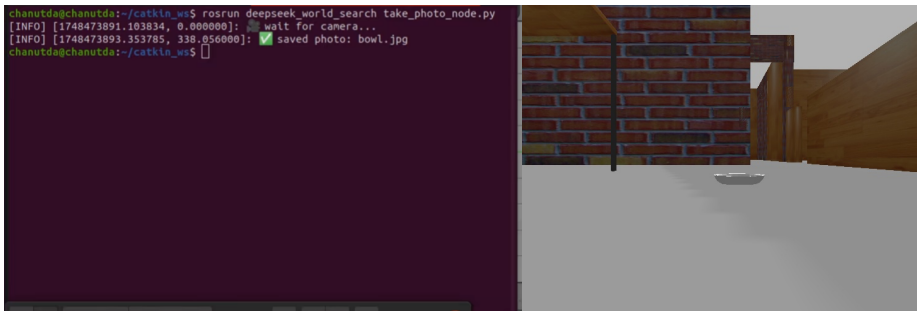

Fig. 7. Execution of take_photo_node: bowl image captured

The figure saved the successful execution of the take_photo_node. On the Figure 7 (left), the terminal shows that the image was saved as bowl.jpg. On the right, the gazebo simulation displays the robot's camera view.

6 Discussion

The above experiments aim to evaluate how DeepSeek-generated plans could be executed effectively by the robot in an indoor environment. The project aims to create a robot capable of exploring unknown spaces, detecting objects of interest (a bowl), and performing context-aware tasks such as taking a photo. The robot should be able to interpret physical interactions, respond dynamically to real-time sensor feedback, and maintain consistent communication with the DeepSeek API under varying operational conditions.

6.1 DeepSeek Planning and Prompt Refinement

In the initial version (deepseek_bowl), a simple prompt is used to instruct DeepSeek to generate waypoints based on the robot's current position. The robot follows these waypoints correctly; however, its movements are inefficient and often lead to the robot being stuck in corner or narrow regions.

In the second version (deepseek_world_search), additional robot state information is introduced. The planner now responds to battery status and object detection, with logic to return to the starting position when power is low. The DeepSeek prompt is expanded accordingly. A new problem emerges: the robot frequently becomes stuck within small indoor regions, especially near the edges of rooms. The DWA planner often reports that it could not generate a viable local trajectory. Moreover, the longer and more complex prompt increases the likelihood of API timeouts from DeepSeek, reducing the system's reliability during exploration.

To overcome the local planning issue, the third version (deepseek_world_search v2) involves tuning key DWA parameters such as velocity and simulation time. Additionally, trial structured room location data (ROOM_LOCATION) is embedded into the prompt, allowing DeepSeek to plan with greater environmental awareness. These enhancements improve trajectory diversity and enable the robot to escape from previously trapped areas more effectively. Nevertheless, the system still fails to treat the bowl as a target. Although the bowl gets hit, the robot did not initiate the photo capture process. This is likely because the odometry and costmap systems continued to treat the bowl as an obstacle, and DeepSeek cannot accurately infer physical interactions from sensor data alone.

6.2 Functional Validation of Photo Capture Module

To isolate the problem with object interaction, a separate experiment (take_photo_node) is conducted to validate the functionality of the robot's image capture node. This module, when executed independently, successfully captures and saves an image upon receiving the correct command. This confirms that the hardware and ROS node responsible for photo capture are working as intended. Therefore, the inability to trigger image capture in earlier experiments stem from the integration gap between object detection, odometry feedback, and planner logic—not from technical failure in the camera node itself.

6.3 **Evaluation**

A key issue is that DeepSeek-generated waypoints, though syntactically correct, often lead the robot into tight corners or near walls where the DWA planner fails to find valid trajectories. Since DeepSeek lacks access to a map, it could not assess path feasibility. Another trial, adding ROOM_LOCATION is a reasonable workaround, giving it some spatial context for avoiding problematic regions.

Another challenge is how the bowl is interpreted. Although DeepSeek is told to “take a photo if the robot hits a bowl,” the navigation system treats the bowl as an obstacle, and odometry fails to validate contact. This suggests a disconnect between symbolic goals and physical detection. If the system distinguished “obstacle” from “target object,” the robot could have prioritized goal-directed motion over avoidance.

Prompt length affects performance. Although providing details is better, longer prompts lead to more frequent DeepSeek timeouts, disrupting plan updates. It is significant to design prompts that balance clarity with responsiveness.

Finally, the planner operated as a one-way system—sending plans without receiving real-time feedback. Providing DeepSeek with status responses (i.e. "plan failed" or "bowl encountered") could enable adaptive replanning. Such a feedback loop is critical for robust LLM-driven control in dynamic settings.

7 Conclusion

The system demonstrates partial success in a simulated Gazebo house using ROS Noetic and TurtleBot3 Waffle Pi. Through multiple test runs, the robot demonstrates the ability to explore following the plan, where odometry executes the path from DeepSeek waypoints. However, it cannot effectively detect the bowl as a target, as it runs into the bowl and DWA treats the bowl like an obstacle, causing it to change direction. Therefore, no command is sent to DeepSeek indicating bowl: True, and subsequently, no command for taking a photo is issued. This behavior results in the take_photo_node not being auto-executed, despite the node functioning correctly in isolation.

Moreover, the prompt sent to DeepSeek is a key part of this implementation. A too-long prompt results in timeout, while if the prompt does not cover enough aspects, the robot’s behavior could become incomplete or inconsistent, failing to respond as expected under certain situations.

For future improvement, integrating a more robust perception system such as YOLO for object detection could replace simple OpenCV-based. Additionally, sending asynchronous feedback to DeepSeek (such as confirmation of detection, or automatic photo capture triggers) could enhance communication efficiency and robot behaviour. Optimising prompt quality, applying dynamic prompt construction, or switching to

other LLMs are also promising directions to reduce latency and improve autonomy in real-world applications.

Acknowledgements. The authors would like to thank OpenAI's ChatGPT (https://chat.openai.com) for assisting in refining the system values and experimental section of this paper. Moreover, this implementation integrates DeepSeek (https://deepseek.com) as a core planner for LLMs-based driven robotic navigation.

References

1. BBC News: Hundreds feared dead in Myanmar earthquake and ‘enormous damage’ across region, rescuer tells BBC - live updates. https://www.bbc.com/news/live/c4gex01m7n5t

2. Bitbucket: deepseek-ros2. https://bitbucket.org/theconstructcore/deepseek-ros2/src/master/

3. Silliman, M.: GitHub - markwsilliman/turtlebot: Basic (hello world style) scripts for TurtleBot (ROS / Python). https://github.com/markwsilliman/turtlebot/tree/master

4. Delmerico, J., Mintchev, S., Giusti, A., Gromov, B., Melo, K., Horvat, T., Cadena, C., Hutter, M., Ijspeert, A.J., Floreano, D., Gambardella, L.M., Siegwart, R., Scaramuzza, D.: The current state and future outlook of rescue robotics. *J. Field Robot.* **36**, 1171–1191 (2019). https://doi.org/10.1002/rob.21887

5. Wang, G., et al.: Development of a search and rescue robot system for the underground building environment. *J. Field Robot.* **40**(3), 655–683 (2023). https://doi.org/10.1002/rob.22152

6. Pathak, J., Sakore, N., Kapare, R., Kulkarni, A., Mali, Y.: Mobile Rescue Robot. *Int. J. Sci. Res. Sci. Eng. Inf. Technol.* **4**(8), 39–43 (2019). https://doi.org/10.32628/IJSRCSEIT