RoboGo: Real-Time Object Navigation via Multimodal LLMs

Andres Aranguren [*1] **Fernando Fejzulla** [*1 2]

Abstract

We present **RoboGo**, a mobile robot system, that leverages multimodal Large Language Model (LLMs) for autonomous object seeking behaviour in real-world environments . Built on a Raspberry Pi 4 with an onboard camera and speaker, the robot actively explores its surroundings and navigates towards a user specified object using visual and spatial understanding powered by Google's *Gemini* API. The system performs two distinct types of prompt-based interactions, one to interpret the visual input and identify the relative objects position and proximiy and second prompt to obtain a defined navigation instructions to move towards the goal object. RoboGo dynamically adjust its trajectory based on the instructions and visual inspection, thus avoiding obstacles and collision throught navigation. Additionally the robot use text-to-speech (gTTS), to provide human-like audio description of the environment and proposed action. Finally we conduct a prompt analysis evaluating latency, token usage and descriptive accuracy across several prompt complexities. The proposed work demonstrates the potential of LLM's perception and decision making capacities, to power real time autonomous robots in unexplored and dynamic environments.

A video demo is available at[1] and code is on GitHub[2].

1. Introduction

Integrating advanced AI models into robotic systems promises to enhance autonomy and flexibility in unstructured environments. Recent breakthroughs in Large Language Models (LLMs) and multimodal AI have enabled models that can interpret both language and vision, opening new possibilities for robotic perception and planning. However, deploying such models on real robots poses challenges in real-time processing and grounding abstract AI outputs in concrete motor actions. This project, **RoboGo**, explores an LLM-driven approach to robotic object navigation. The goal is for a robot to find and move toward a user-chosen object (e.g., *"green ball"*) using only its onboard camera and guidance from a cloud-based multimodal LLM.

In our approach, a RaspberryPi4-based robot uses the *Google Gemini* API to analyze images from its camera. Gemini (specifically the Gemini 1.5 model) is a cutting-edge multimodal LLM capable of understanding and describing the environment that the robot is placed too. By taking advantage of Gemini's powerful vision-language understanding, the robot obtains a semantic description of its environment and the relative position of the target object. The user can select a goal object through a simple interface or voice command, after which the robot operates autonomously to hunt for and approach the object. Throughout the process, the robot uses Text-to-Speech (gTTS) to verbally announce what it observes from the picture taken in every action and what actions it is taking, providing an interactive element to the system.

This work is motivated by the desire to simplify robotic vision and control by outsourcing complex visual perception to a general AI service. Instead of relying on specialized object detectors or hard-coded image processing, RoboGo taps into a general-purpose AI model trained on vast visual and textual data. The key contributions of this project include: (1) a novel integration of a cloud-based multimodal LLM (Gemini) with a low-cost rover robot for real-time navigation, (2) a prompting strategy that yields structured navigation advice (including object direction and distance estimations) from the LLM, and (3) a demonstration of LLM-based scene understanding enabling a robot to handle arbitrary objects without additional training.

2. Background

Recent advances in robotics have begun leveraging the generalization capabilities of large pre-trained models for perception and decision-making. (Ahn et al., 2022) introduced *PaLM-SayCan*, combining a language model with affordance functions to ground high-level instructions in feasible robot actions. Similarly, (Huang et al., 2022) demonstrated that large language models (LLMs) can zero-shot plan tasks

[1] https://drive.google.com/file/d/1b58ytr--bE1kafTmhJ0xLBg3tJTmZdOz/view?usp=sharing

[2] https://github.com/ArangurenAndres/robotics_project

in virtual environments, while (Liang et al., 2023) proposed *Code-as-Policies*, using LLMs to generate executable robot code from natural language.

More recent work has explored multimodal LLMs that accept visual inputs. (Driess et al., 2023) and (Brohan et al., 2023) integrate vision and language to enable robots to reason about scenes and execute commands from visual context. These models represent a shift from task-specific perception modules to flexible, general-purpose vision-language reasoning systems.

Our system, RoboGo, builds on this paradigm by using the Gemini model—a proprietary multimodal LLM—for real-time visual interpretation and high-level control of a physical robot. At each step, Gemini acts as an external "brain," enabling visual reasoning and navigation via prompt-based interactions. This approach is inspired by recent works such as (Vemprala et al., 2024) and (Shah et al., 2023), and demonstrates that even simple robotic platforms can benefit from cloud-based LLMs for complex tasks through effective prompt design.

3. Relevant Work in LLM-Based Robotic Navigation

Traditional robotic navigation relies on dedicated vision algorithms (e.g., blob tracking, object detectors) and control policies to pursue targets. In contrast, our work replaces the vision module with an LLM-based perception system. A related effort is LM-Nav by Shah *et al.* (**?**), which combined a vision model, a language model, and a control module to interpret navigation instructions and reach landmarks. Unlike LM-Nav, RoboGo uses a single model (Gemini) to both recognize the target object and estimate its location, based on visual input.

Other relevant work includes integrating vision-language models like CLIP or ViLD into robotics for goal-driven behaviors. These systems use text queries to identify objects but often still rely on traditional planning algorithms. RoboGo instead leverages the LLM's natural language output to drive navigation, reducing onboard complexity but requiring a reliable cloud connection. This highlights a trade-off: rich semantic understanding via the LLM versus dependency on external services.

In summary, most prior work has focused on high-level planning or instruction following with LLMs. RoboGo contributes to the emerging field of LLM-driven closed-loop control by demonstrating how a single multimodal model can handle both perception and decision-making in real time. To our knowledge, this is among the first implementations using Google's Gemini API in a real-world robotic system.

4. System Design and Implementation

The RoboGo system is composed of a hardware platform and an integrated software stack that enables autonomous, LLM-guided behavior. The hardware includes a Raspberry Pi 4, a camera module, motor drivers for locomotion, and a speaker for audio feedback. The software stack manages communication between the robot's control components and the Gemini LLM API, enabling seamless integration of perception and decision-making. In this section, we describe the individual components and explain how they interact to support real-time navigation in dynamic environments. Additionally, we detail the process of translating the LLM's textual output into actionable robot commands. A simplified overview of the system architecture is presented in Figure 1, while a more detailed version of the full control loop, including prompt handling, parsing, and movement selection, can be found in **Appendix** (see Figure 8).

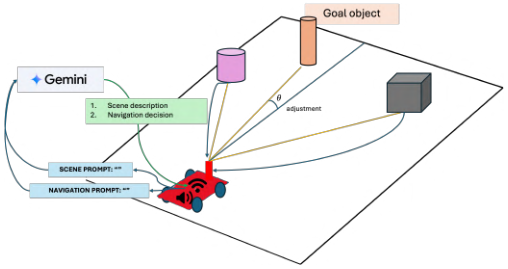

Figure 1. Diagram of RoboGo architecture. The robot captures an image of the environment and sends it to the Gemini multimodal LLM using two structured prompts: one for scene understanding and one for navigation. Gemini responds with a semantic description of visible objects and navigation instructions toward the user-defined goal object. Based on these responses, the robot adjusts its heading, moves, and gives spoken feedback using text-to-speech

4.1. Hardware

The robot is built on a *PiCar-4WD* chassis, a small four-wheel drive mobile robot kit for Raspberry Pi. A Raspberry Pi 4 serves as the main controller, running Raspberry Pi OS and the control software. The Pi is equipped with a Raspberry Pi Camera Module (v2) mounted at the front of the robot to capture the robot's forward view. For audio output, a JBL speaker is connected to the Pi (to the Pi's audio jack) to play speech responses. The PiCar-4WD kit includes motors, a motor driver board, and an optional ultrasonic sensor. In this project, we rely solely on the motors for movement and do not use any additional range sensors, as obstacle proximity and visual inspection are performed by the LLM based on the images captured by the onboard camera.

Motor control is handled through the provided *picar-4wd*

Python library. This library abstracts low-level motor commands, allowing us to set wheel speeds for forward, backward, and turning motions. We calibrated a base motor power (duty cycle of 70/100) that provides a moderate speed and empirically determined the durations needed for basic manoeuvrers (e.g., a 0.45 s motor run to effect a 90° turn). These parameters were tuned to ensure that the robot's movements during each control loop cycle are meaningful enough to make progress, yet small enough to avoid overshooting fine adjustments.

Figure 2 shows the assembled RoboGo platform. The robot has a compact dimension (approximately 20cm × 15cm), with the Raspberry Pi and camera mounted on top. It operates untethered on battery power: a USB power bank supplies the Raspberry Pi, while a separate battery pack powers the motors through the driver board.

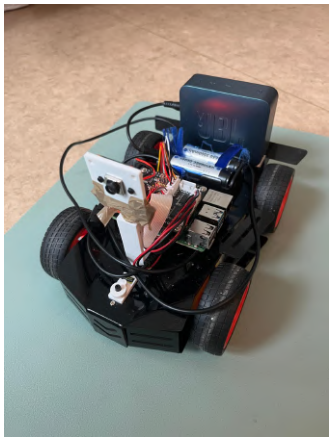

Figure 2. The RoboGo robot, based on a RaspberryPi4 with a PiCar-4WD chassis and camera. The camera (mounted at front) provides live images to the LLM, and a JBL speaker outputs speech.

4.2. Software

All RoboGo software is developed in Python using PyCharm, accessed via SSH over the robot's local network. The system consists of two main Python scripts implementing the core components described below.

- **Camera Interface:** We use the *Picamera2* library (the official Python library for the Raspberry Pi CSI camera) to capture images. The camera is configured to capture still frames at a resolution of 640×480 pixels in RGB format. For efficiency, we capture frames on every movement that the robot makes, rather than live footage; each cycle of the control loop triggers one frame capture.
- **Gemini API Client:** The robot interfaces with the Gemini LLM via Google's *google.generativeai* Python SDK, authenticated using an API key stored as an environment variable. Upon startup, the model is loaded, and image-prompt pairs are sent to receive text-based responses. The selected Gemini variant offers low-latency performance suitable for real-time interaction. See **Experiments** section for a latency study that supports this affirmation.
- **Text-to-Speech (TTS):** For voice feedback, we synthesize speech from text using *gTTS* (Google Text-to-Speech). For every action and choice the robot must make, the program creates an audio file, which is then played on a portable audio player. The robot speaks, pausing for a while.
- **Motor Control:** The *robot_controller* module provides movement control functions (forward, backward, turn_left, and turn_right) by wrapping calls to the PiCar-4WD library. Movements are executed by invoking these routines for predetermined durations, with a *stop()* function used to stop the motors when necessary.

4.3. Project pipeline for implementation

The following steps are executed in each iteration of the control loop (up to a maximum number of steps to avoid infinite running), see Figure 1 for a simplified visuasl representation:

1. **Image Capture:** Capture the latest frame from the camera as an RGB image. The image is then JPEG-encoded and subsequently base64-encoded into a string, as required by the Gemini API format.
2. **LLM Prompting:** Send a prompt to Gemini requesting environment description and navigation guidance to user defined object. We designed the following baseline structured prompt template:
   - **GEMINI_SCENE_PROMPT:**

```
GEMINI_SCENE_PROMPT = """
You are the vision system of a
↪  robot. Describe the scene in
↪  front of you.
Say exactly: I see the following
↪  elements: [list up to 5
↪  distinct objects].
The closest object appears to be
↪  [closest object]. The furthest
↪  object appears to be [furthest
↪  object].
Example: I see the following
↪  elements: a red ball, a blue
↪  box, a yellow chair.
The closest object appears to be a
↪  red ball. The furthest object
↪  appears to be a yellow chair."
```

```
"""
```

- **GEMINI_NAVIGATION_PROMPT_TEMPLATE:**

```
GEMINI_NAVIGATION_PROMPT_TEMPLATE
↪  = """
You are the navigation AI for a
↪  robot that is 25cm wide. Your
↪  task is to guide it to the GOAL
↪  OBJECT.
Analyze the current view and
↪  provide the following
↪  information, each on a new
↪  line:
1. GOAL VISIBLE: [Yes/No]
2. GOAL DIRECTION: [Not
↪  Visible/Center/Slightly
↪  Left/Slightly Right/Far
↪  Left/Far Right]
3. GOAL PROXIMITY: [Not
↪  Visible/Reachable/Very
↪  Close/Near/Medium/Far]
4. PATH STATUS: [Clear/Minor
↪  Obstacle/Major
↪  Obstacle/Blocked]
5. OBSTACLE INFO: [Describe briefly
↪  if Minor or Major Obstacle,
↪  otherwise "None"]

Prioritize reaching the GOAL OBJECT
↪  safely. Be very concise and
↪  follow the format precisely.
Only consider obstacles directly
↪  in the robot's 25cm path.
GOAL OBJECT: {goal}
"""
```

3. **Text to speech:** Using Gemini prompts output, we convert text to speech, to describe the current scene, objects in proximity and visual inspection of the goal object. Based on visual inspection, we additionally output the required trajectory adjustment and movement selected movement to proceed with.

4. **Trajectory Adjustment:** Based on the LLM's response, the robot continues navigating toward the goal object if it is detected in the visual field; otherwise, it resumes exploration until the object is found. Depending on the goal's visibility, estimated proximity, and the presence of obstacles, the robot selects the most appropriate action while moving toward the defined target. **Subsection 4.4** describes more in detail how to transform LLM output into actionable movements for the robot.

   The complete sequence of movements using the baseline prompt until reaching the goal object are report in **Annex Figure 9**.

4.4. Tanslating LLM output into actional robot commands

To translate Gemini's output into actionable robot commands we follow a three-step pipeline see Figure 8. First, image is captured by robot's camera encoded and sent to the Gemini API via `ask_gemini` function, together with a the structured prompt to request navigation advice. Gemini returns natural language response describing the scene. This language output is then parsed by `parse_navigation` function which extracts key fields into a structured dictionary representing the current scene state such as These fields include `goal_visible` (e.g., `True` or `False`), `goal_direction` (e.g., `"slightly left"`), `goal_proximity` (e.g., `"near"`). Finally the `pursue_object` function interprets this structured data to select and execute a movement command (e.g., `forward`, `left`, or `right`), adjusting the trajectory based on proximity, obstacle presence, and visibility of the goal object. This loop continues until the robot either reaches the goal or terminates due to a timeout or failure condition.

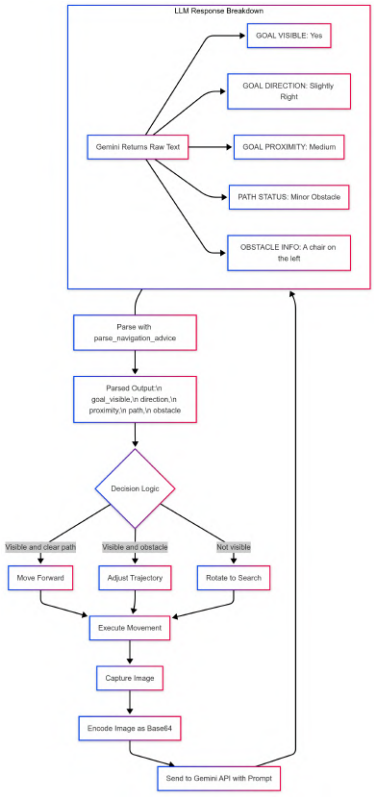

Figure 3. Pipeline to tranform LLM output into action robot movement commands. The process can be split into three main steps. 1) LLM request using structured prompt, 2) LLM output parsing into defined dictionary structure, 3) Based on output parsing proceed with deicions making and movement

Following we present an example of a typical audio se-

quence for a user-defined goal object (e.g., “green ball”):

1. ”I see the following elements: a green ball, a water bottle, and a chair.”
2. ”The goal object is: green ball.”
3. ”I will now attempt to reach the green ball.”
4. ”The green ball is visible. There is a water bottle on the right. Adjusting trajectory to the left.”
5. ”Path is clear.”
6. ”Moving toward the green ball.”
7. ”Green ball reached. Mission complete.”

5. Experiments & discussion

To evaluate the effectiveness of our model, we designed a set of controlled experiments involving different environment configurations. In each case, we placed a variety of objects at varying distances and orientations relative to the robot, and then one of the elements was selected to be the target goal. The robot was asked to navigate towards the goal using visual input and language-based decision-making. We used the same baseline prompt defined in **Section 4.3**, ensuring consistency across all trials.

Evaluation was based on the following criteria:

- **Total number of movement iterations** (N): the total number of decision-execution cycles taken by the robot.
- **Visibility persistence** (V): the number of iterations in which the goal object remained visible to the robot.
- **Final distance to the goal object** (D): the Euclidean distance between the robot and the goal object at the end of the episode (This distance was).

These metrics were used to assess both the efficiency and robustness of the system’s performance. To aggregate these factors into a single performance score S, we define the following formula:

$$S = \alpha \cdot \left(\frac{V}{N}\right) + \beta \cdot \left(1 - \frac{D}{D_{\text{max}}}\right)$$

where α and β are weighting coefficients such that $\alpha + \beta = 1$, for this project we chose $\alpha = 0.5$ and $\beta = 0.5$, and D_{max} is the maximum possible distance to the goal in the environment. A higher score indicates better performance, balancing visibility tracking with navigation efficiency and proximity to the goal. Evaluation results for the different prompt variants across multiple environment scenarios are presented in the **Annex** section, Table 2, which reports mentioned metrics and corresponding performance scores for each case.

5.1. Prompt Analysis

To evaluate the impact of prompt design on system performance, we conducted a series of experiments in which the robot was placed in the same environment see Figure 4 and queried using five distinct prompts with increasing levels of complexity and structure. Since our control pipeline relies entirely on interpreting the LLM’s output to make navigation decisions, prompt formulation becomes the most critical factor influencing the robot’s behavior. A well-structured prompt can guide the model to produce actionable and precise descriptions, improving decision accuracy and reducing ambiguity, see Table 1, for a description of the prompt variants.

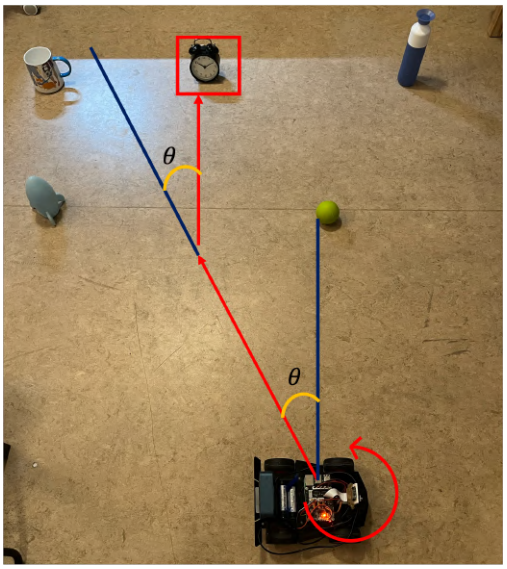

Figure 4. Environment implement during prompt analysis. Ideally based on the baseline prompt result the robot should rotate until seeing in its visual field the goal object (clock), once it detects correct is position to the left to avoid obstacle, more forward , correct again position to be centered with objective and finally move forward reaching the goal

To quantify the computational and interpretive effects of different prompts, we saved metrics such as token count and inference latency across the different prompt variants. The token count reflects the verbosity and structural overhead introduced by each prompt, which can influence both model cost and interpretability (Brown et al., 2020). Latency provides a practical measure of the responsiveness of the system, critical for real-time robotics applications (Driess et al., 2023). Figure 5 and Figure 6 present the token count

Variant	Scene Description Prompt	Navigation Prompt
V0	Simple object list	Basic navigation question
V1	Adds robot persona	Structured Q&A format
V2	Explicit output structure (e.g., "I see the following elements...")	Formal key-value pairs
V3	Adds navigational context	Includes physical constraints (e.g., robot width)
V4/V5	Requires more detail per object	Emphasizes action orientation with safety considerations

Table 1. Prompt variants used in the experiments, increasing in structure and specificity to guide LLM-based robot control.

and latency for the different prompt variants, these results allow to compare prompt efficiency, required for the trade-off analysis between expressiveness and runtime performance.

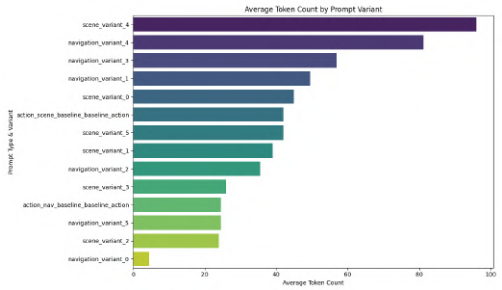

Figure 5.

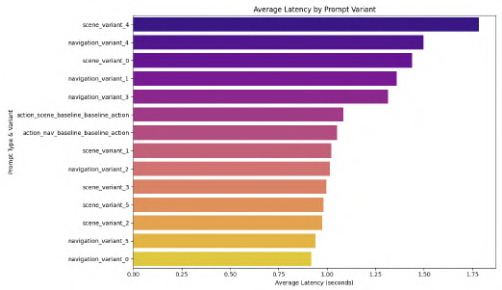

Figure 6.

The results in Figure 5 and Figure 6reveal a trade-off between prompt complexity and system efficiency. Prompt variants such as `scene_v_4` and `navigation_v_4`, which are highly structured and context-rich, exhibit the highest average token counts and longest inference latencies. This indicates that while these prompts provide more elaborate guidance to the language model potentially improving downstream decision quality they also increase computational load and delay, which could hinder real-time responsiveness. In contrast, simpler prompts like `navigation_variant_0` achieve minimal latency and token usage, but may lack the specificity needed for reliable action planning. The proposed baseline variants described in **section 4** with predefined templates (`action_scene_baseline_action`) are in the middle ground, balancing interpretability and efficiency. These findings highlight the importance of carefully designing prompts to match the task requirements and hardware constraints, especially in robotics systems where both precision and reaction time are critical.

6. Conclusion

This project introduced a low-cost robotic system powered by a multimodal LLM (Gemini) for visual navigation and object-seeking tasks. By implementing structured prompting and semantic parsing, the robot is able to perceive, reason, and act in real time using only an onboard vision and LLM-guided decisions. Our experiments demonstrate the significance of prompt design and how it affects both latency and task performance, highlighting the importance of prompt engineering in LLM-driven robotics. RoboGo showcases the potential of integrating general-purpose language models into real-world embodied agents, enabling flexible behaviour without retraining on domain-specific environments.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

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

Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang,

W., et al. Palm-e: An embodied multimodal language model. 2023.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022.

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

Shah, D., Osiński, B., Levine, S., et al. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*, pp. 492–504. PMLR, 2023.

Vemprala, S. H., Bonatti, R., Bucker, A., and Kapoor, A. Chatgpt for robotics: Design principles and model abilities. *Ieee Access*, 2024.

A. Appendix

A.1. Project diagram

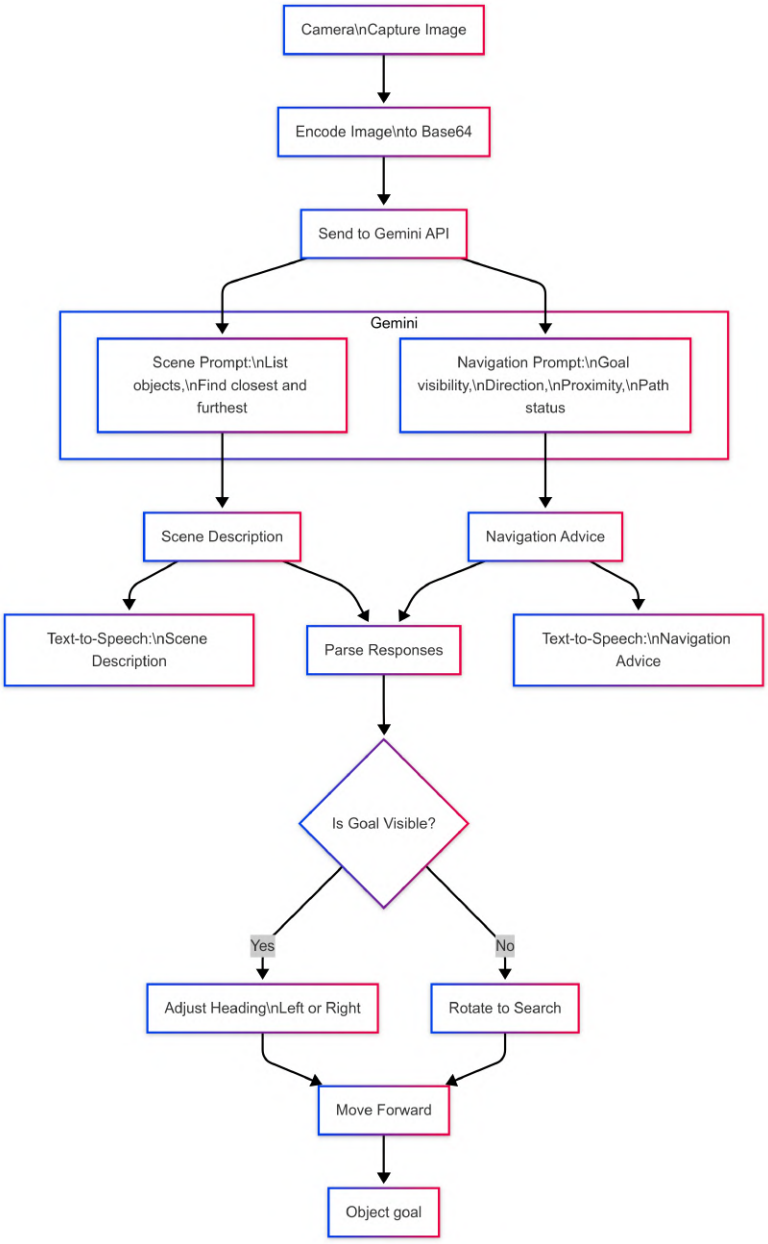

Figure 7. System-level flowchart of the RoboGo control loop. The robot captures an image, sends it to the Gemini LLM with two distinct prompts (scene and navigation), and receives high-level semantic responses. These are parsed and used to determine navigation decisions such as adjusting heading or rotating to search. The robot then executes a selected movement and repeats the cycle until the goal object is reached.

A.2. Prompt metrics per case

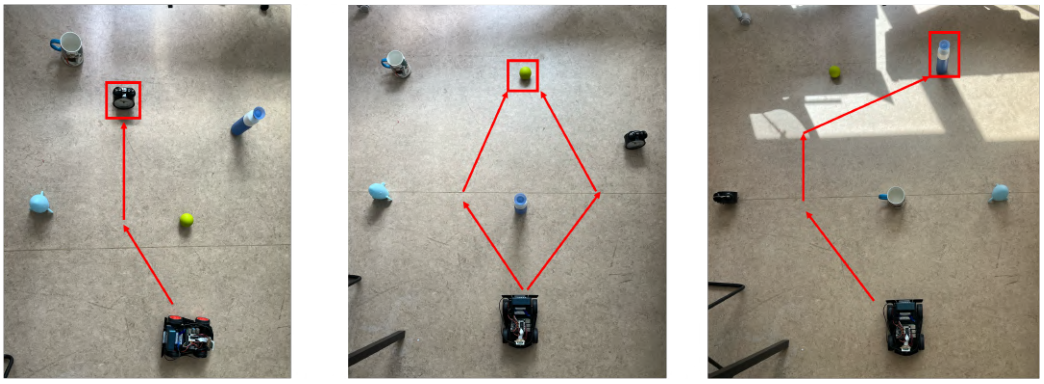

Figure 8. Environments used to evaluate prompts variants performance. From left to right Env (A), Env (B), Env (C). Red paths correspond to the optimal trajectory to reach the goal object, nonenthelss given that the model has no previous knowledge about its environment the stochastic nature of the system ends up in more iterations without a guarantee to reach the object, since the robot moves in an open environment. To limit the exploration variables, each experiment should be run in a closed environment without any other objects than the ones presented in the figure

Prompt Variant	**Environment A**				**Environment B**				**Environment C**			
	N	V	D	S	N	V	D	S	N	V	D	S
V0 (Basic)	5	1	2.4	0.38	6	2	2.1	0.43	6	2	2.3	0.40
V1 (Persona)	6	3	1.9	0.50	5	2	1.7	0.53	6	3	1.8	0.52
V2 (Structured)	6	4	1.5	0.60	5	3	1.6	0.61	6	4	1.4	0.63
V3 (Contextual)	6	5	0.9	0.74	5	4	1.0	0.72	6	5	0.8	0.76
V4 (Detailed)	7	5	1.0	0.71	6	5	1.1	0.70	6	6	0.9	0.74
V5 (Action-rich)	6	6	0.6	0.77	6	6	0.5	0.80	5	5	0.7	0.76

Table 2. Evaluation results for each prompt variant across three environment scenarios (A, B, C). For each prompt variant, 10 episodes were executed per environment. The table reports the average values of the following metrics: total movement iterations (N), visibility persistence (V), final distance to the goal object (D cm), and the computed performance score (S).

A.3. Sequence example using baseline prompt

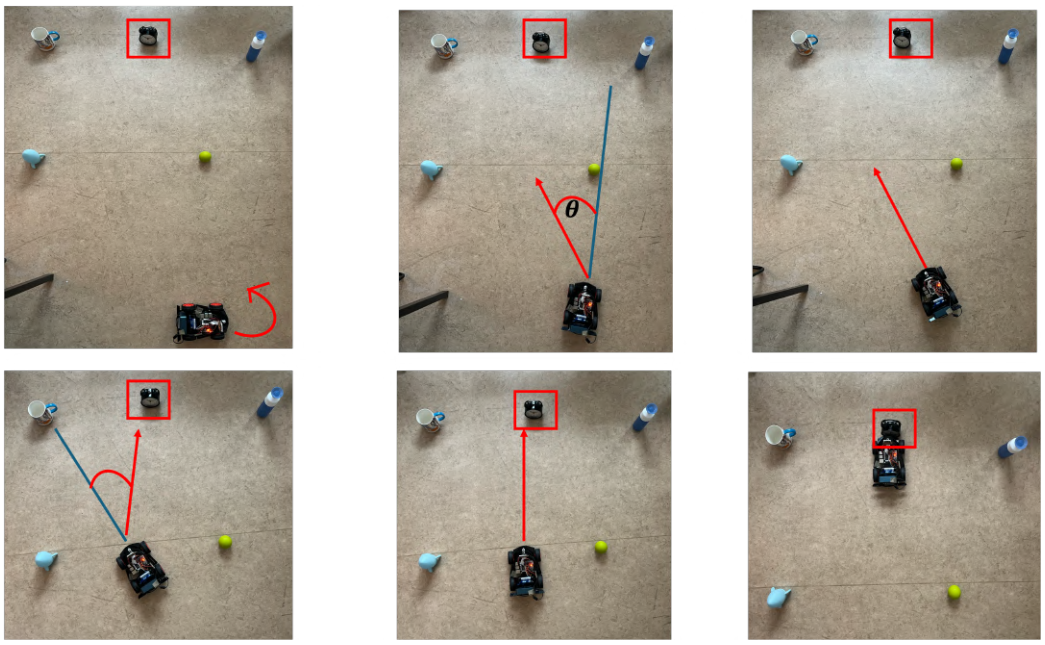

Figure 9. Decision making sequence until reaching goal object powered by LLM visual inspection using baseline prompt