Final Project

DS685 AI for Robotics

Professor Pantelis Monogioudis

Yash Patel

31702347

yp359

Agents for Robotics

ROSA

A Natural Language Control Interface for ROS2 Using NATS

Abstract

Modern robotic systems built on ROS2 are powerful, but interacting with them still requires familiarity with command-line tools, topic names, and system internals. This project introduces **ROSA**, a lightweight natural language interface that allows a user to interact with a ROS2 system using plain English, while keeping execution behavior predictable and safe.

ROSA is designed with a strict separation of responsibilities. The agent interprets user input and translates it into executable commands but never executes them directly. All commands are sent through a NATS message broker, where a dedicated ROS bridge performs the actual execution inside the ROS2 environment and returns the result. This separation makes the system easier to debug, safer to operate, and more flexible to extend.

The current implementation supports end-to-end execution of ROS2 command-line queries and system introspection commands, with reliable request–reply to communication. The infrastructure layer is complete and stable. Ongoing work focuses on improving intent classification within the agent to ensure that informational requests never trigger robot motion, reinforcing the system's safety and predictability goals.

1. Introduction

ROS2 provides powerful tools for building and running robotic systems, but interacting with those systems is still largely command-line driven. Even simple tasks often require remembering exact commands, topic names, or parameters. This creates a gap between how people naturally think about actions and how they are forced to communicate with the robot.

Using natural language seems like an obvious improvement. Asking for system information or telling a robot to move in plain English is far more intuitive than typing long commands. However, robotics adds an important constraint: a misunderstood instruction can cause real movement, not just a wrong answer on a screen. That makes unrestricted language interpretation risky.

ROSA was created to address this problem in a controlled way. Instead of acting like a conversational assistant, it functions as a strict translation layer between human language and ROS2 commands. The system is intentionally designed to be predictable and debuggable, with clear boundaries between interpretation, communication, and execution.

The goal of this project is not to make the robot autonomous or “intelligent,” but to make interaction with ROS2 simpler while preserving safety and control.

2. Project Objectives

The primary objective of this project is to design a safe and predictable way to interact with a ROS2 system using natural language. Rather than building a conversational assistant, the focus is on creating a controlled interface that translates user intent into clearly defined system actions.

A key goal is to maintain a strict separation between interpretation and execution. The agent is responsible only for understanding input and selecting the appropriate action, while all command execution is handled externally within the ROS2 environment. This design choice ensures better debuggability and reduces the risk of unintended behavior.

Another important objective is to prevent accidental robot motion during informational or diagnostic requests. System inspection commands, such as listing nodes or topics, must never trigger navigation behavior. This constraint drives much of the system's design and ongoing refinement.

Finally, the project aims to build an extensible foundation. By using a message broker and modular components, ROSA can be expanded in the future to support additional tools or behaviors without changing the core architecture.

3. System Architecture

3.1 Overview

The system is designed with a clear separation of responsibilities, where each component performs a specific and well-defined role.

The **ROSA Agent** receives natural language input from the user, interprets the intent, and translates it into an executable command. It does not execute any ROS commands directly.

The **Messaging Layer (NATS)** acts as the communication bridge between system components. It decouples the agent from the ROS runtime, supports request–reply messaging, and allows components to operate independently.

The **ROS Bridge** runs within the ROS2 workspace and is responsible for receiving commands from NATS, executing them in the ROS2 environment, and returning the execution output.

The **ROS2 Environment** hosts the robot, simulation, and navigation stack. It remains isolated from direct user or agent access and executes only explicitly received commands.

This architecture ensures that natural language interpretation never directly controls the robot, making system behavior predictable and traceable.

3.2 Pipeline Flowchart (Block-Based)

Start

User Input

Natural language command entered by the user

ROSA Agent

Receives input and translates it into a concrete system command

NATS Messaging Layer

Carries the command from the agent to the execution environment

ROS Bridge

Receives the command and prepares it for execution

ROS2 Environment

Executes the command within the robot or simulation runtime

Execution Output

Result or system response generated after execution

NATS Messaging Layer

Returns the output to the requesting client

ROSA Client

Displays the response to the user

End

3.3 ROSA Agent

3.3.1 Role of the ROSA Agent

The ROSA Agent serves as the primary interface between the user and the robotic system. It is responsible for receiving natural language input and translating it into a concrete, executable command. The agent is implemented using **pydantic.ai** and leverages a **large language model (LLM)** based on **LLaMA** to interpret user intent in a structured and constrained manner.

The ROSA Agent does not execute any ROS2 commands itself and has no direct access to the robot or simulation environment. All execution is delegated to downstream components through the messaging layer.

This design choice is intentional. By separating language interpretation from execution, the system reduces the risk of unintended robot behavior and improves overall safety and traceability.

3.3.2 Core Responsibilities

The ROSA Agent is designed with a narrow and clearly defined scope. Its responsibilities include:

- Accepting natural language commands from the user
- Interpreting the intent of the command using a constrained LLM-based approach
- Translating the input into a system-level executable command
- Dispatching the command through the messaging layer
- Receiving and relaying execution responses back to the user

The agent does not ask follow-up questions, explain its reasoning, or perform execution directly.

3.3.3 Key Files Driving the Agent

The behavior and functionality of the ROSA Agent are primarily driven by three files: client.py, rosa_agent.py, and react_prompt.txt.

client.py

This file manages communication with the NATS messaging system. It establishes a connection to the NATS server, publishes commands generated by the agent, and waits for a response using a request–reply mechanism. It does not contain any logic related to language interpretation or decision-making.

rosa_agent.py

This file contains the core logic of the ROSA Agent and integrates **pydantic.ai** with a **LLaMA-based language model**. The LLM is used to interpret natural language input and generate structured, executable commands under strict constraints. The implementation is intentionally designed to produce actionable outputs only, rather than conversational responses.

react_prompt.txt

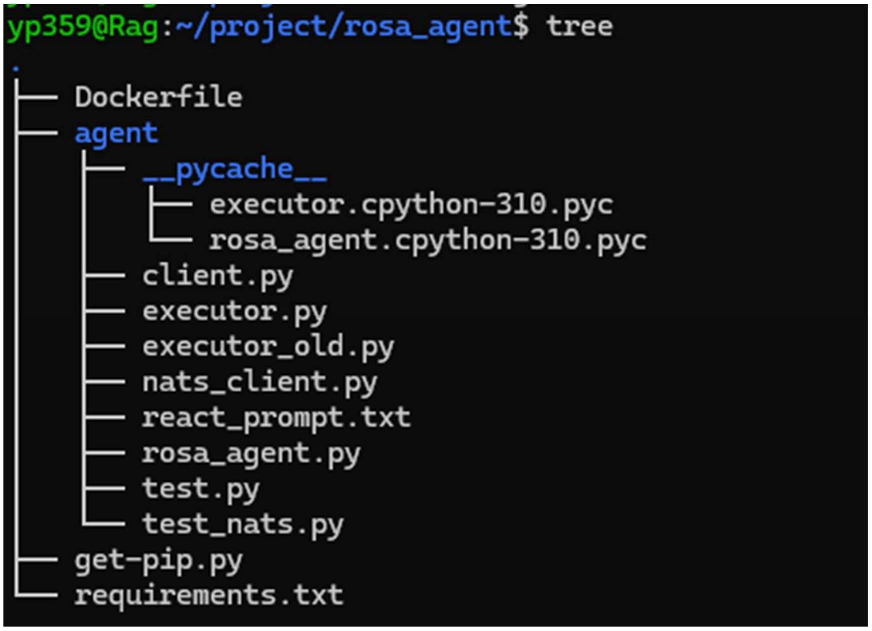
This file defines behavioral constraints for the language model. It specifies what the agent is allowed to do and what it must avoid, such as asking clarifying questions or providing explanations. Keeping these constraints separate from the code allows behavior changes without modifying the core logic.

3.3.4 Validation and Current Behavior

The ROSA Agent has been validated through standalone tests that confirm successful connection to the NATS server, correct message publication, and proper handling of responses. Test executions demonstrate that the agent can reliably send commands and receive corresponding execution outputs.

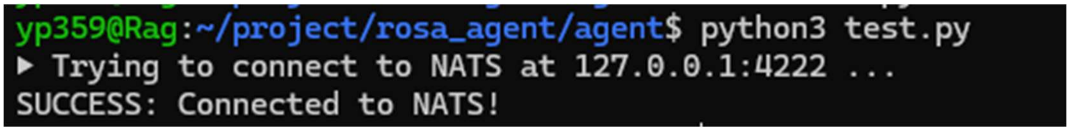
At this stage, the agent's communication and dispatch mechanisms are stable. Ongoing work is focused on refining intent classification to ensure strict separation between informational commands and navigation actions.

Here executor.py, executor_old.py were earlier used to establish connection in earlier trials.



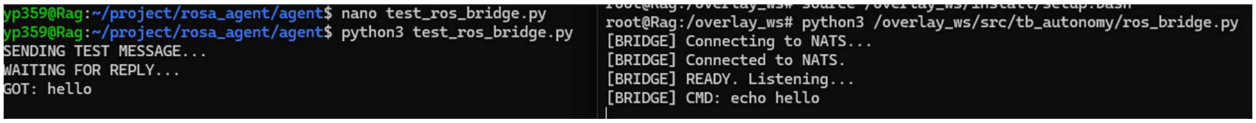
```
yp359@Rag:~/project/rosa_agent$ tree
.
├── Dockerfile
├── agent
│   ├── __pycache__
│   │   ├── executor.cpython-310.pyc
│   │   └── rosa_agent.cpython-310.pyc
│   ├── client.py
│   ├── executor.py
│   ├── executor_old.py
│   ├── nats_client.py
│   ├── react_prompt.txt
│   ├── rosa_agent.py
│   ├── test.py
│   └── test_nats.py
├── get-pip.py
└── requirements.txt
```

Fig. Rosa_Agent Tree



```
yp359@Rag:~/project/rosa_agent/agent$ python3 test.py
▶ Trying to connect to NATS at 127.0.0.1:4222 ...
SUCCESS: Connected to NATS!
```

Fig. Connection Between Rosa_agent and NATS server



```
yp359@Rag:~/project/rosa_agent/agent$ nano test_ros_bridge.py
yp359@Rag:~/project/rosa_agent/agent$ python3 test_ros_bridge.py
SENDING TEST MESSAGE...
WAITING FOR REPLY...
GOT: hello
```

```
root@Rag:/overlay_ws# source /overlay_ws/install/setup.bash
root@Rag:/overlay_ws# python3 /overlay_ws/src/tb_autonomy/ros_bridge.py
[BRIDGE] Connecting to NATS...
[BRIDGE] Connected to NATS.
[BRIDGE] READY. Listening...
[BRIDGE] CMD: echo hello
```

Fig. Connection between Rosa_agent and Ros_bridge

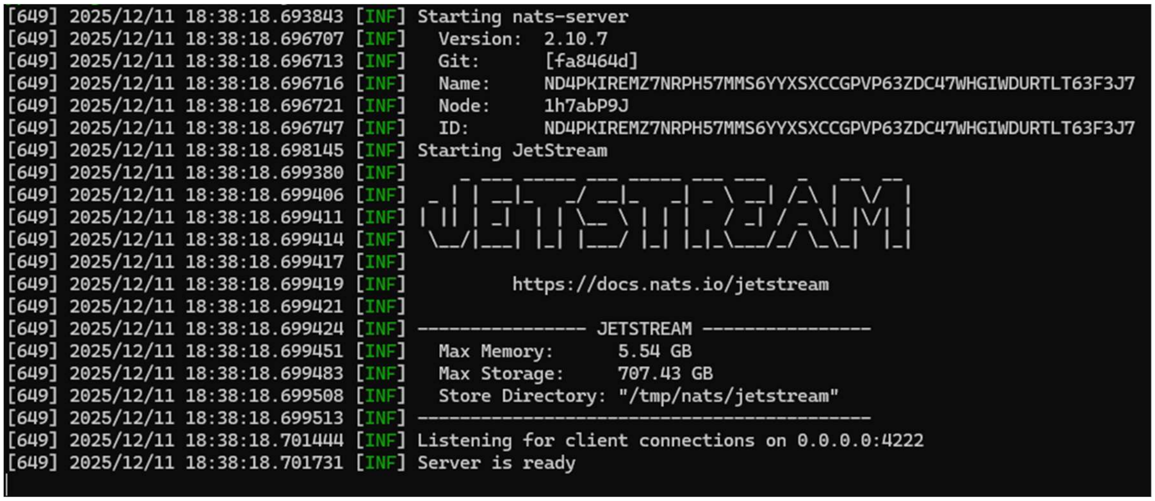
3.4 Messaging Layer (NATS)

The messaging layer is implemented using **NATS** and acts as the communication backbone of the system. It enables message exchange between the ROSA Agent and the ROS2 execution environment while keeping both components loosely coupled.

In the current setup, the NATS server runs on the host machine and listens for client connections on **port 4222**. JetStream is enabled to support reliable request–reply communication. The server initializes successfully and confirms readiness before any system components begin communication.

Commands generated by the ROSA Agent are published to NATS and received by the ROS Bridge for execution. Execution results are sent back through the same messaging layer. NATS does not interpret or modify messages and functions purely as a transport mechanism.

By using NATS, the system allows independent startup and recovery of components while maintaining predictable and traceable command execution.



```
[649] 2025/12/11 18:38:18.693843 [INF] Starting nats-server
[649] 2025/12/11 18:38:18.696707 [INF]   Version:  2.10.7
[649] 2025/12/11 18:38:18.696713 [INF]   Git:      [fa8464d]
[649] 2025/12/11 18:38:18.696716 [INF]   Name:     ND4PKIREMZ7NRPH57MMS6YYXSXCCGPVP63ZDC47WHGIWDURTLT63F3J7
[649] 2025/12/11 18:38:18.696721 [INF]   Node:     1h7abP9J
[649] 2025/12/11 18:38:18.696747 [INF]   ID:       ND4PKIREMZ7NRPH57MMS6YYXSXCCGPVP63ZDC47WHGIWDURTLT63F3J7
[649] 2025/12/11 18:38:18.698145 [INF] Starting JetStream
[649] 2025/12/11 18:38:18.699380 [INF]     _ ___ _____ ___ _____ ___ ___   _   __  __
[649] 2025/12/11 18:38:18.699406 [INF]  _ | | __|_   _/ __|_   _| _ \ __| /_\ |  \/  |
[649] 2025/12/11 18:38:18.699411 [INF] | || | _|  | | \__ \ | | |   / _| / _ \| |\/| |
[649] 2025/12/11 18:38:18.699414 [INF]  \__/|___| |_| |___/ |_| |_|_\___/_/ \_\_|  |_|
[649] 2025/12/11 18:38:18.699417 [INF]
[649] 2025/12/11 18:38:18.699419 [INF]          https://docs.nats.io/jetstream
[649] 2025/12/11 18:38:18.699421 [INF]
[649] 2025/12/11 18:38:18.699424 [INF] ---------------- JETSTREAM ----------------
[649] 2025/12/11 18:38:18.699451 [INF]   Max Memory:      5.54 GB
[649] 2025/12/11 18:38:18.699483 [INF]   Max Storage:     707.43 GB
[649] 2025/12/11 18:38:18.699508 [INF]   Store Directory: "/tmp/nats/jetstream"
[649] 2025/12/11 18:38:18.699513 [INF] -------------------------------------------
[649] 2025/12/11 18:38:18.701444 [INF] Listening for client connections on 0.0.0.0:4222
[649] 2025/12/11 18:38:18.701731 [INF] Server is ready
```

Fig. NATs Server

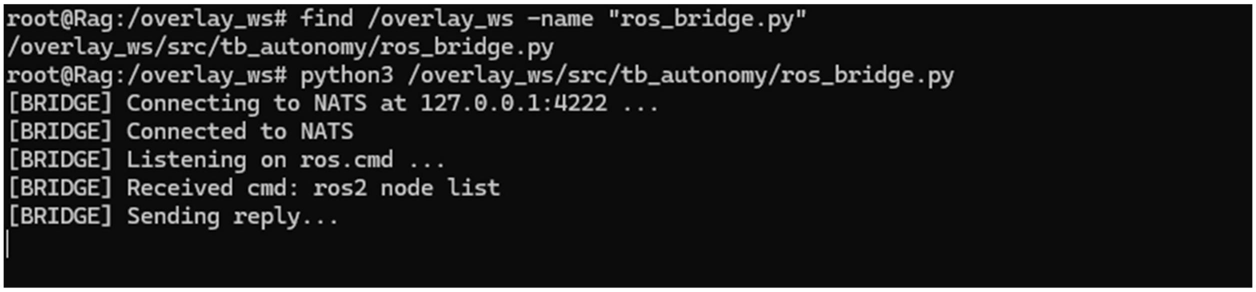
3.5 ROS Bridge

The ROS Bridge serves as the execution interface between the messaging layer and the ROS2 environment. It runs inside the ROS2 container and is responsible for receiving commands from NATS and executing them within the correct ROS2 context.

The bridge subscribes to predefined NATS subjects and waits for incoming commands published by the ROSA Agent. Upon receiving a command, it executes the corresponding ROS2 CLI or navigation-related action. The bridge does not interpret user intent or modify commands; it strictly executes what it receives.

After execution, the ROS Bridge captures the resulting output and sends it back through the NATS messaging layer using a request–reply pattern. This allows the ROSA Agent to receive confirmation and results for each issued command.

By isolating execution logic within the ROS Bridge, the system ensures that robot actions occur only within a controlled environment. This separation improves safety, simplifies debugging, and prevents direct access to the ROS runtime from the agent.



```
root@Rag:/overlay_ws# find /overlay_ws -name "ros_bridge.py"
/overlay_ws/src/tb_autonomy/ros_bridge.py
root@Rag:/overlay_ws# python3 /overlay_ws/src/tb_autonomy/ros_bridge.py
[BRIDGE] Connecting to NATS at 127.0.0.1:4222 ...
[BRIDGE] Connected to NATS
[BRIDGE] Listening on ros.cmd ...
[BRIDGE] Received cmd: ros2 node list
[BRIDGE] Sending reply...
```

Fig. Ros_bridge connected to NATs server

3.6 ROS2 Environment

The ROS2 Environment hosts the robot simulation, navigation stack, and all ROS2 nodes required for system operation. It represents the only part of the system where commands are actually executed.

This environment remains isolated from direct user or agent access. All interactions with ROS2 occur exclusively through the ROS Bridge, ensuring that execution is tightly controlled and traceable. The ROS2 runtime does not receive natural language input and does not make decisions independently.

By restricting execution to explicitly received commands, the ROS2 Environment maintains predictable behavior and reduces the risk of unintended actions. This isolation is a key design choice that supports system safety, reliability, and ease of debugging.

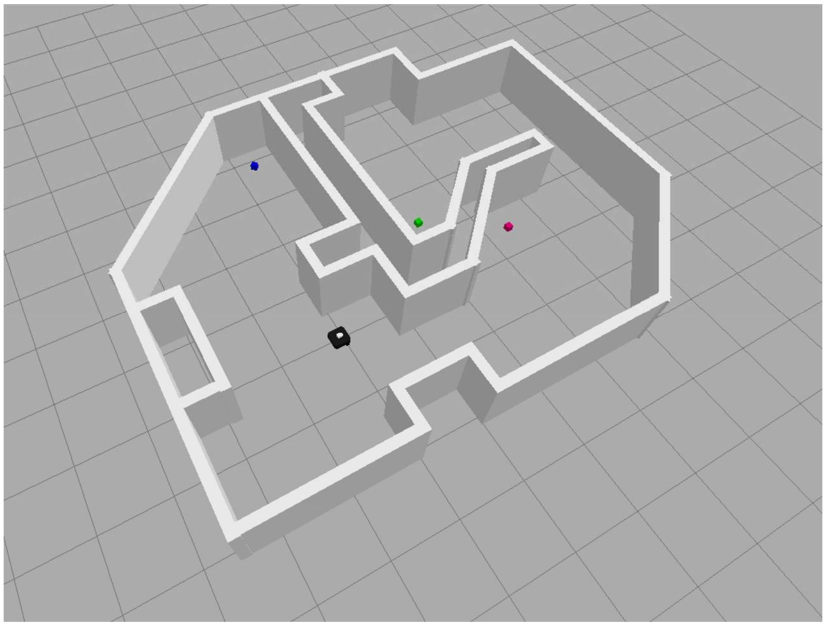


Fig. Gazebo

3.7 Natural Language Processing and Command Dispatch Flow

Natural language input provided by the user is first processed by the ROSA Agent. The agent uses a **LLaMA-based large language model (LLM)** integrated through **pydantic.ai** to interpret user intent and classify the request as either a ROS2 CLI command or a navigation action. The LLM is used strictly for intent interpretation and structured output generation, not execution.

Once the intent is identified, the ROSA Agent converts the request into a structured **JSON message** that explicitly specifies the execution type and parameters. This message is passed to the client component, which handles network communication and sends the JSON to the messaging layer using a request–reply pattern.

The NATS messaging layer transports the message to the ROS Bridge, where the command is executed within the ROS2 environment. Execution results are returned through the same path and displayed to the user. This flow ensures that natural language interpretation and robot execution remain fully decoupled and controlled.

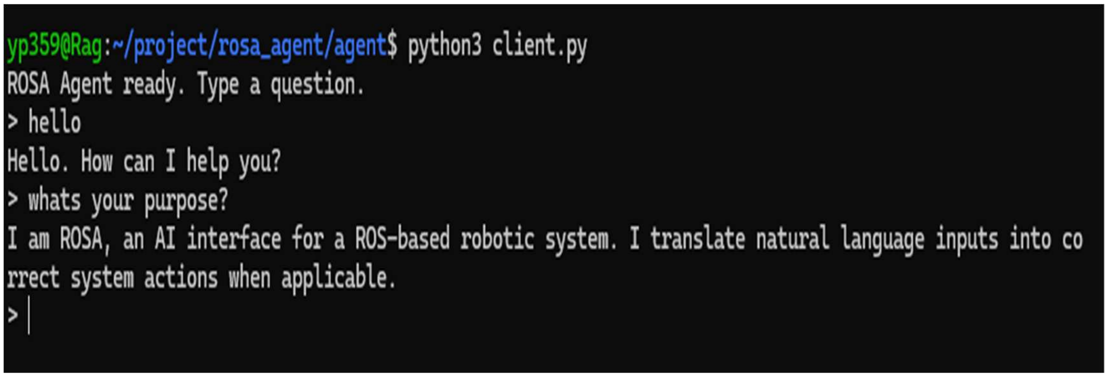
4. System Validation and Demonstration

4.1 Natural Conversation Handling

Before issuing any executable commands, the system is tested using normal conversational input. In this example, the user greets the system and asks about its purpose. The ROSA Agent responds conversationally, describing its role as an interface for a ROS-based robotic system.

No commands are generated, and no messages are sent to the messaging layer or ROS2 environment during this interaction. This confirms that the system can distinguish between casual conversation and actionable requests.

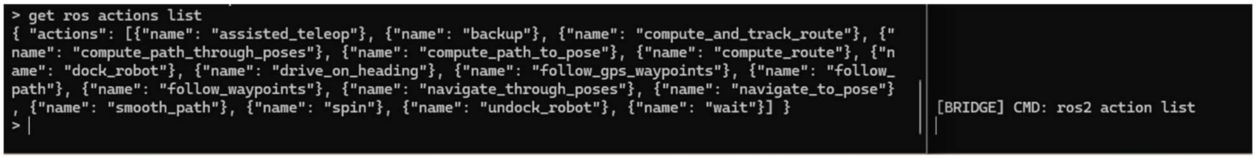
This behavior is important for safety and usability, as it ensures that non-actionable input does not trigger unintended system execution or robot motion.



```
yp359@Rag:~/project/rosa_agent/agent$ python3 client.py
ROSA Agent ready. Type a question.
> hello
Hello. How can I help you?
> whats your purpose?
I am ROSA, an AI interface for a ROS-based robotic system. I translate natural language inputs into co
rrect system actions when applicable.
> |
```

Fig. Natural Conversation

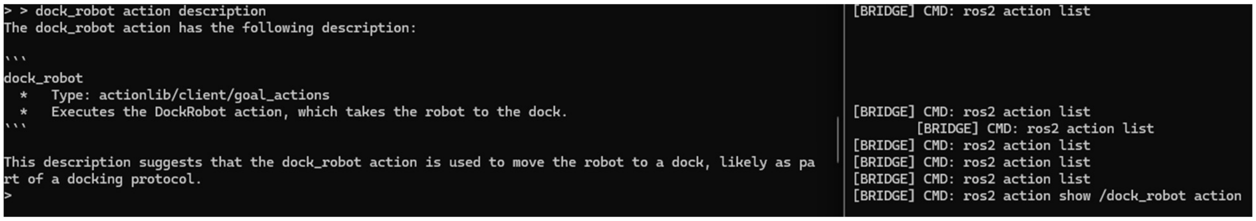
4.2 Validation of ROS2 CLI Command Execution



```
> get ros actions list
{ "actions": [{"name": "assisted_teleop"}, {"name": "backup"}, {"name": "compute_and_track_route"}, {"
name": "compute_path_through_poses"}, {"name": "compute_path_to_pose"}, {"name": "compute_route"}, {"n
ame": "dock_robot"}, {"name": "drive_on_heading"}, {"name": "follow_gps_waypoints"}, {"name": "follow_
path"}, {"name": "follow_waypoints"}, {"name": "navigate_through_poses"}, {"name": "navigate_to_pose"}
, {"name": "smooth_path"}, {"name": "spin"}, {"name": "undock_robot"}, {"name": "wait"}] }
> |
```

```
[BRIDGE] CMD: ros2 action list
```

Fig. Rosa_Client – Ros_bridge CLI communication on action list



~~~
> > dock_robot action description
The dock_robot action has the following description:

```
dock_robot
  *   Type: actionlib/client/goal_actions
  *   Executes the DockRobot action, which takes the robot to the dock.
```

This description suggests that the dock_robot action is used to move the robot to a dock, likely as pa
rt of a docking protocol.
>
~~~

```
[BRIDGE] CMD: ros2 action list





[BRIDGE] CMD: ros2 action list
        [BRIDGE] CMD: ros2 action list
[BRIDGE] CMD: ros2 action list
[BRIDGE] CMD: ros2 action list
[BRIDGE] CMD: ros2 action list
[BRIDGE] CMD: ros2 action show /dock_robot action
```

Fig. Rosa_Client – Ros_bridge CLI communication on specific action Dock_Robot

This subsection validates the system’s ability to interpret and execute **ROS2 CLI commands expressed in natural language**, while ensuring that no navigation or robot motion is triggered.

In the first example, the user requests a list of available ROS actions using natural language (e.g., “get ros actions list”). The ROSA Agent correctly classifies this request as an informational CLI command and translates it into the corresponding ROS2 command. The command is routed through the messaging layer and executed by the ROS Bridge, returning a structured list of available actions.

In the second example, the user asks for details about a specific action, namely dock_robot. The system again interprets the request as a CLI query and executes the appropriate ROS2 command to retrieve the action description. The returned output provides information about the action type and its intended functionality.

Across both cases, execution logs confirm that the commands are handled entirely through the ROS2 CLI pathway. No navigation actions are triggered, and the robot remains stationary throughout. This demonstrates that ROSA can safely support system inspection and diagnostics through natural language input while maintaining a strict separation from physical execution.

4.3 Validation of Navigation Command Execution

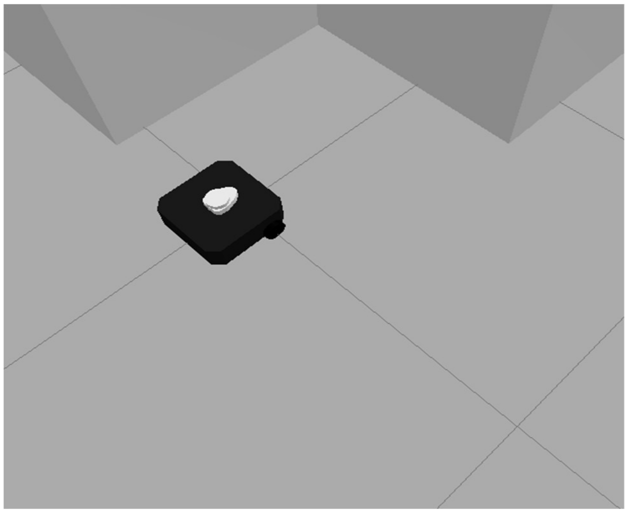
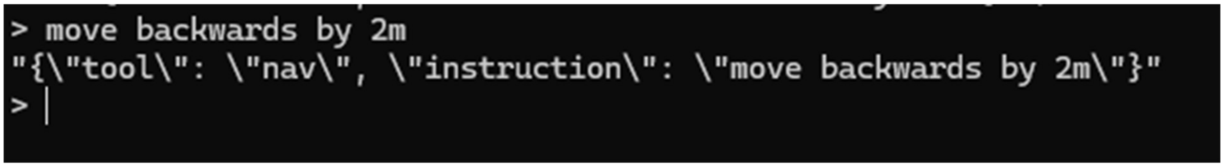
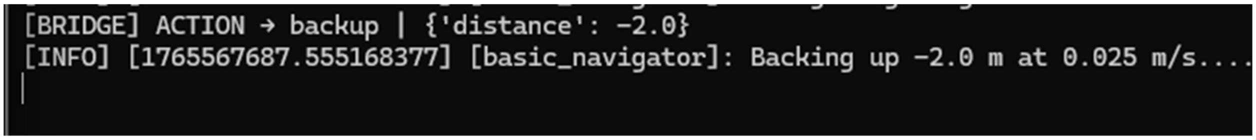


Fig. Robot Initial Position

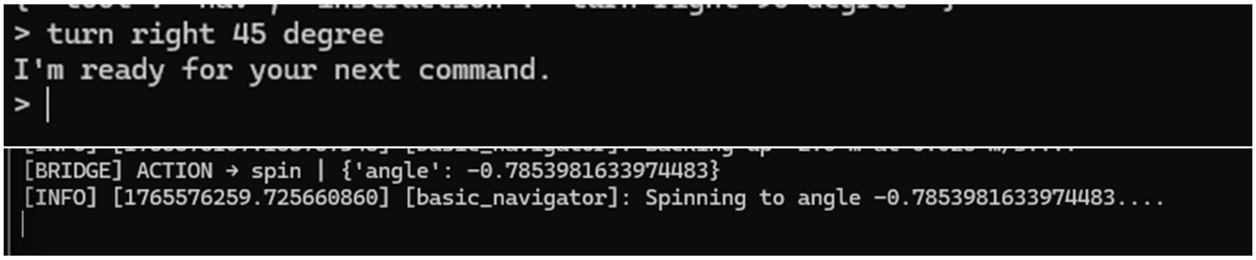


```
> move backwards by 2m
"{\"tool\": \"nav\", \"instruction\": \"move backwards by 2m\"}"
> |
```



```
[BRIDGE] ACTION → backup | {'distance': -2.0}
[INFO] [1765567687.555168377] [basic_navigator]: Backing up -2.0 m at 0.025 m/s....
|
```

Fig. Navigation command issued from the client and executed by the ROS Bridge(backward movement- 2m)



```
> turn right 45 degree
I'm ready for your next command.
> |
```

```
[BRIDGE] ACTION → spin | {'angle': -0.7853981633974483}
[INFO] [1765576259.725660860] [basic_navigator]: Spinning to angle -0.7853981633974483....
|
```

Fig. Navigation command issued from the client and executed by the ROS Bridge (Rotate 45 degree).

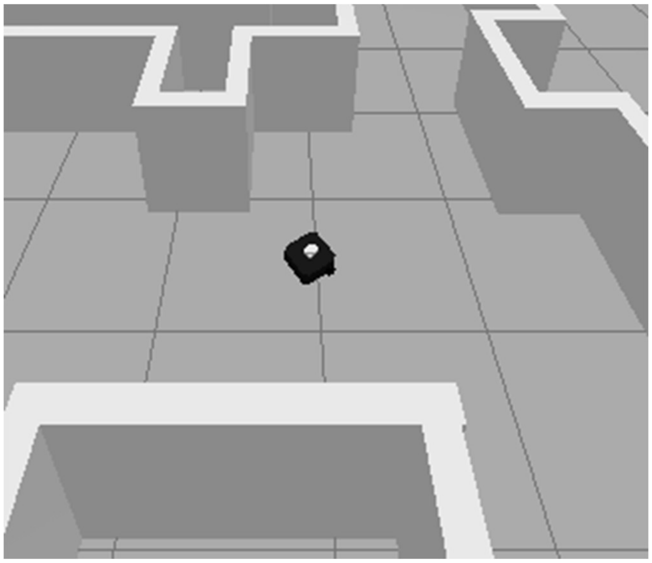
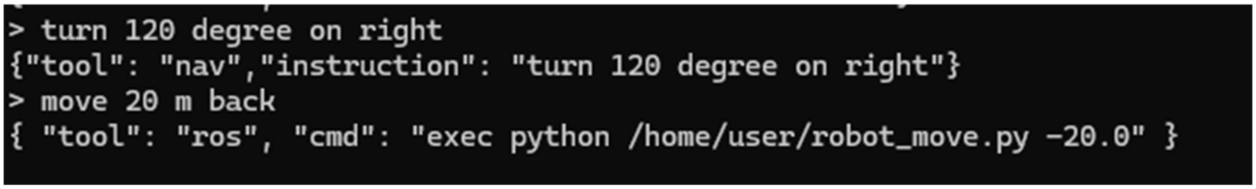
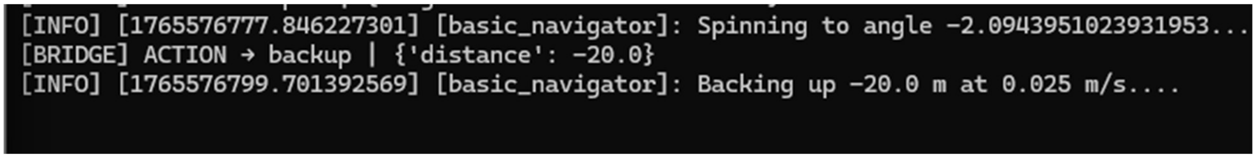


Fig. Robot in Gazebo after Backward move and 45 degree rotation

In a different simulation test-



```
> turn 120 degree on right
{"tool": "nav","instruction": "turn 120 degree on right"}
> move 20 m back
{ "tool": "ros", "cmd": "exec python /home/user/robot_move.py -20.0" }
```



```
[INFO] [1765576777.846227301] [basic_navigator]: Spinning to angle -2.0943951023931953...
[BRIDGE] ACTION → backup | {'distance': -20.0}
[INFO] [1765576799.701392569] [basic_navigator]: Backing up -20.0 m at 0.025 m/s....
```

Fig. Navigation command issued from the client and executed by the ROS Bridge (Turn 120 degree and backward movement of 20)

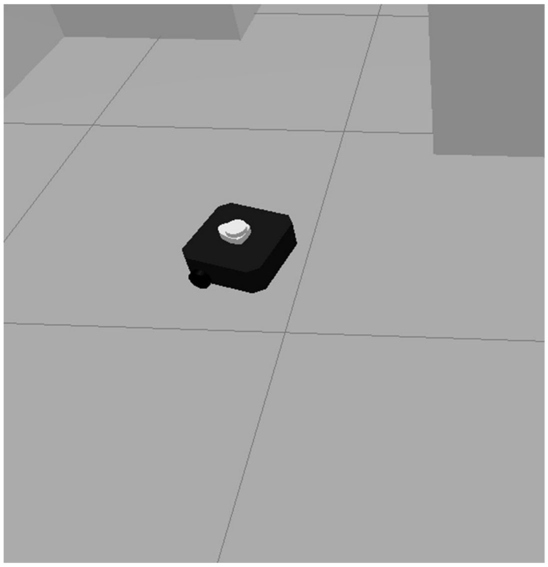


Fig. Robot in Gazebo – Turn 120 and move 20m back

This subsection validates the system's ability to translate natural language navigation commands into correct and observable robot motion within the ROS2 simulation environment.

The demonstration begins with the robot in its **initial stationary position**. At this stage, no commands have been executed, establishing a clear baseline for evaluating subsequent movement.

In the first navigation test, the user issues the command "**move 2 m back.**" Then **"turn right 45 degree."** The ROSA Agent interprets this input as a navigation action, converts it into a structured instruction, and routes it through the messaging layer to the ROS Bridge. Execution logs confirm that the navigation stack performs a controlled spin to the specified angle. The robot's orientation visibly changes in the simulation, matching the intended backward movement and 45-degree rotation.

In different simulation test, the user issues a compound navigation instruction: **"turn 120 degree on right"** followed by **"move 20 m back."** The system again classifies these as navigation commands and executes them sequentially. The robot first performs the specified rotation and then moves backward by the requested distance at a regulated speed. Both actions are confirmed through execution logs and visible robot motion.

Across all navigation tests, the robot behavior aligns precisely with the issued natural language commands. No unintended actions are observed, and execution occurs only after explicit navigation instructions are provided. This validates that the system can reliably handle navigation commands while maintaining controlled and predictable robot motion.

5. Limitations and Future Work

While the ROSA system demonstrates reliable natural language control for ROS2 CLI queries and navigation commands, several limitations remain.

Intent classification is currently effective for the tested commands but may require additional safeguards for more complex or ambiguous input. Future improvements could include stricter validation or confidence checks before executing navigation actions.

The system executes commands sequentially, which simplifies control but limits support for concurrent or long-running tasks. Extending the design to handle asynchronous execution would improve scalability.

Navigation support is currently limited to basic motion commands. Future work could expand this to higher-level behaviors such as waypoint navigation while preserving controlled execution.

Overall, these limitations outline a clear path for improving robustness, safety, and capability in future versions of the system.

6. Conclusion

This project presented **ROSA**, a controlled natural language interface for interacting with ROS2 systems. The system demonstrates that natural language input can be safely and reliably translated into ROS2 CLI queries and navigation commands without granting direct execution control to the language model.

By separating interpretation, communication, and execution into distinct components, ROSA ensures predictable behavior and improves system safety. Validation results show that the system correctly distinguishes between conversational input, informational CLI commands, and navigation actions, executing only when appropriate and producing observable, deterministic outcomes.

The work highlights that natural language interfaces for robotics do not need to be fully autonomous or conversational to be effective. Instead, a constrained, modular design can provide usability benefits while maintaining the control and traceability required in robotic systems.

Overall, ROSA establishes a practical foundation for future extensions toward more advanced behaviors, improved intent handling, and stronger safety mechanisms, while preserving a clear and debuggable execution pipeline.

7. References

AI Agents for Robotics – Engineering AI Agents

Pydantic AI

2410.06472

pantelis/turtlebot-maze