

FINAL REPORT

COURSE: MOBILE ROBOTICS

Felipe Snitynski Camillo
felipesnitynski@alunos.utfpr.edu.br

Gustavo Deda Carneiro Pereira
gustavodeda@alunos.utfpr.edu.br

Advisor: João Alberto Fabro
fabro@utfpr.edu.br

February 27, 2025

1. Introduction

The project aimed to develop an autonomous robot using **ROS2** and **Gazebo**, enabling autonomous mapping in simulated environments and navigation via **voice commands**.

2. Methodology

The implementation followed these steps:

1. Setting up the development environment with **ROS2** and **Gazebo**. **ROS2 Humble** was installed on an **Ubuntu 22.04 LTS** system (used for this project).
2. Installing TurtleBot3, using the **waffle** model for mapping and voice-command navigation. The default waffle world was also used for the project—though other worlds can be used as well.
3. Using an autonomous navigation module (Nav2) with **library isolation** via localhost servers to avoid version conflicts between **ros2 – python – nav2**.
4. Implementing a voice recognition module for remote commands.

3. Autonomous Navigation

Autonomous navigation was implemented using route-planning algorithms. By default, Nav2—used in this project—employs the Dijkstra algorithm along with LIDAR sensors for obstacle detection. As a result, the robot can map the environment.

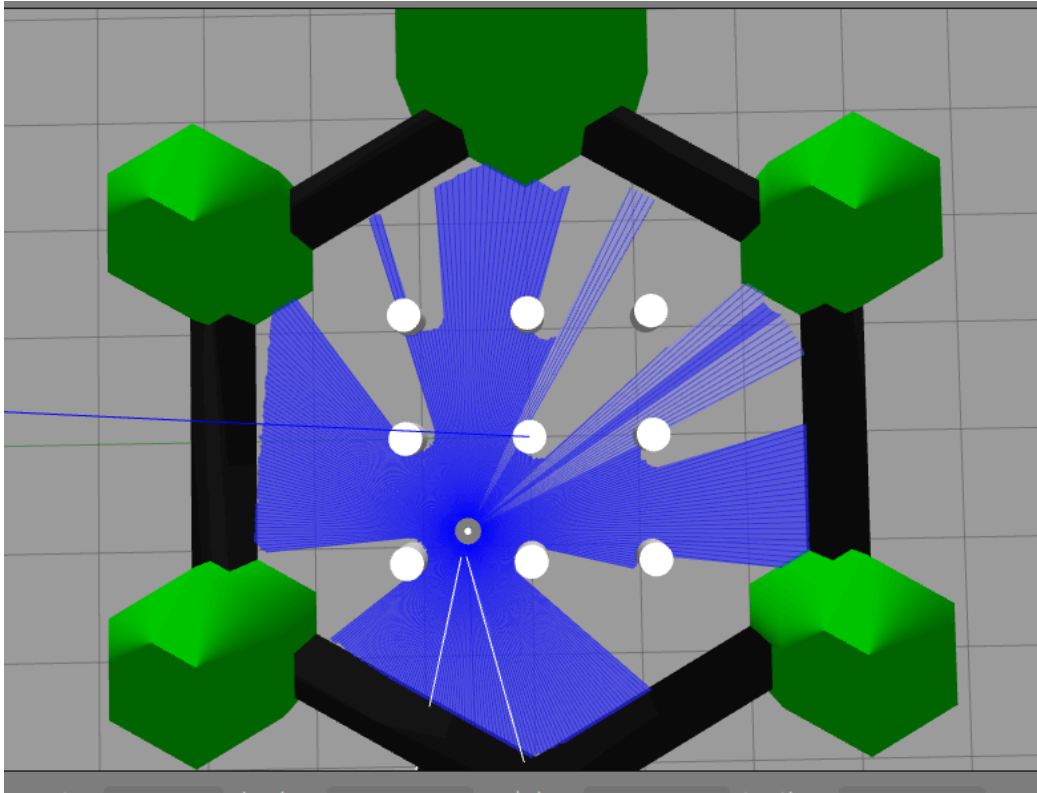
During development, conflicts arose between the Nav2 version and the Python version (Python 3) used in ROS2. To address this issue, a localhost server was used with isolated ROS2 libraries, enabling the use of the navigation packages without interference. The solution is based on the repository:

https://github.com/TakaHoribe/ros2_explorer

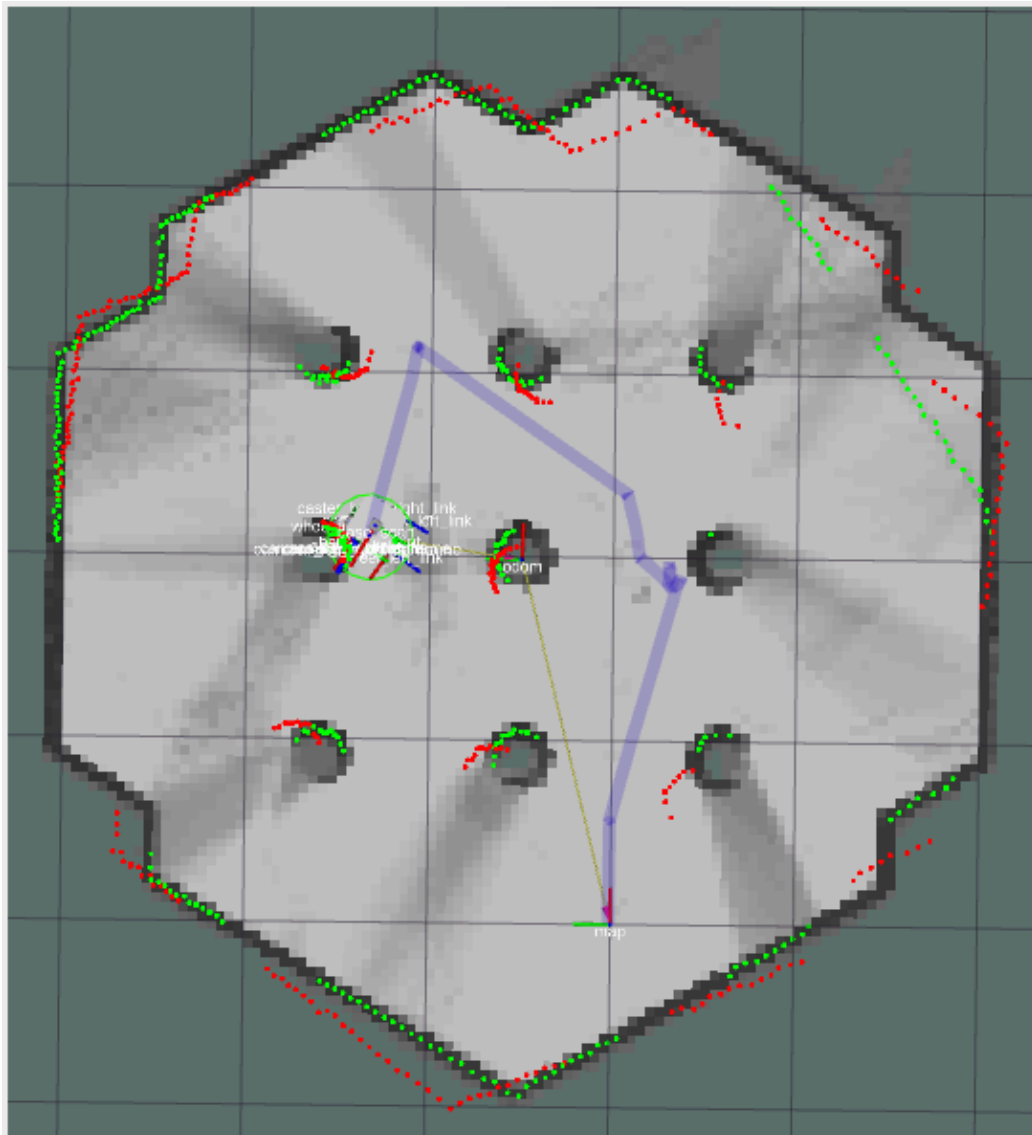
Whenever it was necessary to invoke the autonomous navigation package, a request was made to the localhost server to access the Nav2 libraries. The navigation strategy has two approaches:

- **Random navigation**, which checks if the robot is stuck in a certain location.
- **Wall-following navigation**, enabling the robot to use walls as a guide within the environment.

However, there was an issue with Nav2 when the navigation algorithm should stop: after mapping the environment, the robot stops and returns to the spawn location. The reason why Nav2 is not correctly evaluating the condition (“if”)—which should check how much of the map has been explored—has not yet been identified.



Gazebo screenshot of the robot navigating



RViz screenshot of the robot during mapping

4 Voice Command

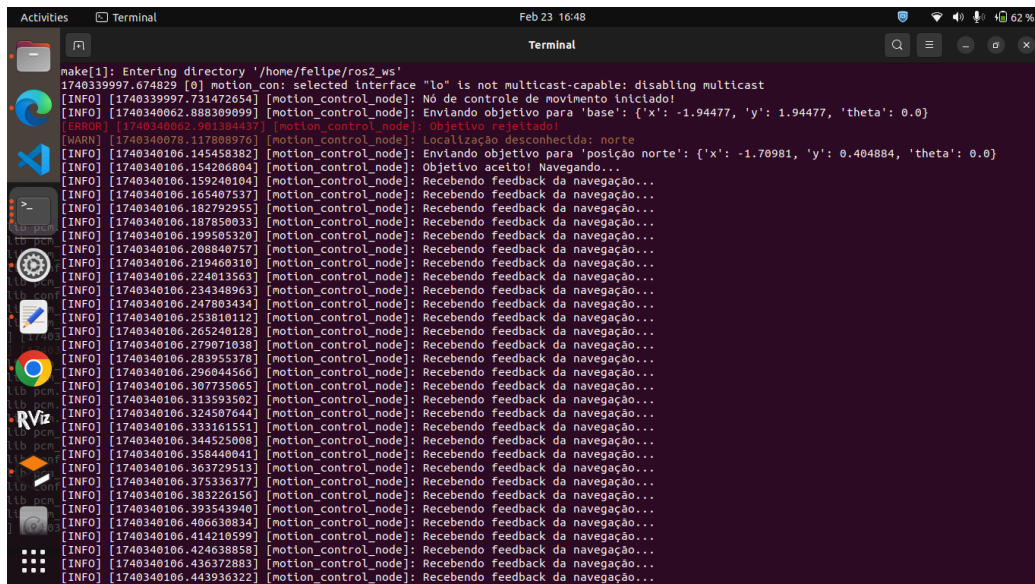
A voice command system integrated with **ROS2** was developed, allowing the robot to receive and execute spoken user instructions. Below is an overview of how the system was structured.

4.1 Capturing and Publishing Commands

- The `voice_command_node.py` package is responsible for capturing voice commands.
- It continuously operates, capturing the words spoken by the user.
- Every 5 seconds, the captured words are sent to a node named **“voice”**.

4.2 Processing and Executing Navigation

- A second package reads the messages published by the **“voice”** node.
- This package checks if the received command matches any location (**waypoint**), which is defined by the map coordinates from the autonomous navigation stage (for example, if the user says “room,” the package searches its dictionaries for “room” and then sends the coordinates to **Nav2** to start navigation to that location).
- Upon reaching the destination, the robot stops and waits for new commands.



```
make[1]: Entering directory '/home/felipe/ros2_ws'
1740339997.674829 [0] motion_con: selected interface "lo" is not multicast-capable: disabling multicast
[INFO] [1740339997.731472654] [motion_control_node]: Nó de controle de movimento iniciado!
[INFO] [1740340002.688309899] [motion_control_node]: Enviando objetivo para 'base': {'x': -1.94477, 'y': 1.94477, 'theta': 0.0}
[ERROR] [1740340002.801349417] [motion_control_node]: Objetivo rejeitado
[WARN] [1740340078.117808976] [motion_control_node]: Localização desconhecida: norte
[INFO] [1740340106.145458382] [motion_control_node]: Enviando objetivo para 'posição norte': {'x': -1.70981, 'y': 0.404884, 'theta': 0.0}
[INFO] [1740340106.154206804] [motion_control_node]: Objetivo aceito! Navegando...
[INFO] [1740340106.159240104] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.165407537] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.182792855] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.187850833] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.199505320] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.208840757] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.219460310] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.224013563] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.234340963] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.247803434] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.253810112] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.265240128] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.279071038] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.283955378] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.296044566] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.307735965] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.313593592] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.324507644] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.333161551] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.344525008] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.358440841] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.363729513] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.375336377] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.383226156] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.393543940] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.406630834] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.414210599] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.424638858] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.436372883] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.440936322] [motion_control_node]: Recebendo feedback da navegação...
[INFO] [1740340106.452222288] [motion_control_node]: Recebendo feedback da navegação...
```

Robot receiving the location and executing navigation

This integration of speech recognition with the navigation and control modules allows the robot to be commanded in an intuitive manner, automating locomotion according to user instructions. Note that the robot does not recognize keywords within sentences, only direct sentences that match the stored location entries. For example, if the user says “go to room” but only “room” is stored, it will not recognize “go to room” unless the user precisely says “room.”

```
class MotionControlNode(Node):
    def __init__(self):
        super().__init__('motion_control_node')
        # Subscreve os comandos de voz (nomes de locais)
        self.subscription = self.create_subscription(String, '/cmd_voice', self.command_callback, 10)
        # Cria o action client para navegação
        self._action_client = ActionClient(self, NavigateToPose, 'navigate_to_pose')

        # Mapeia nomes de locais para coordenadas (x, y e ângulo theta em radianos)
        self.locations = {
            'posição leste': {'x': 2.12593, 'y': -1.70981, 'theta': 0.0},
            'posição oeste': {'x': 2.08585, 'y': 2.3629, 'theta': 0.0},
            'base': {'x': -1.94477, 'y': 1.94477, 'theta': 0.0},
            'posição norte': {'x': -1.70981, 'y': 0.404884, 'theta': 0.0},
            'posição sul': {'x': -1.94477, 'y': 1.94477, 'theta': 0.0},
        }

    self.get_logger().info("Nó de controle de movimento iniciado!")
```

Example of valid waypoints for navigation

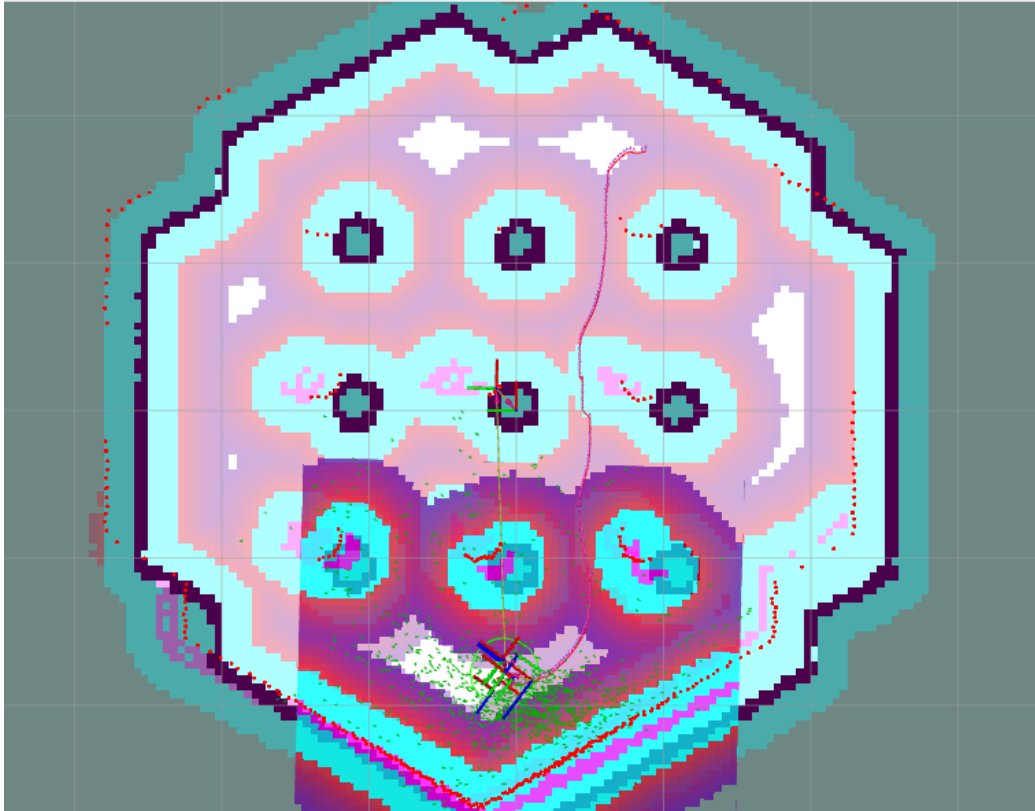


Image of the robot going to the defined waypoint

5. Libraries Used

Several libraries were used to ensure the robot's functionality:

- **rclcpp**: For C++ programming in ROS2.
- **nav2**: A framework for autonomous navigation in ROS2.
- **gazebo_ros**: Interface for ROS2-Gazebo integration.
- **speech_recognition**: For processing voice commands.

All libraries were installed following the standard procedures of each package, except for Nav2, which was installed via the server.

6. How to Run

To compile and run the project, follow these steps:

Run **makefile_navigation** inside the ros2_ws directory to start autonomous navigation:

css

```
make -f makefile_navigation all
```

Run **makefile_voice** inside the ros2_ws directory to start voice commands:

css

```
make -f makefile_voice all
```

7. Results

The following results were achieved:

- The robot was able to move autonomously in the simulated environment.
- Partial completion of mapping, requiring manual stopping of autonomous mapping.
- Robot control via voice commands, providing an intuitive form of interaction.

Project Repository

<https://github.com/Feimac/RoboticaMove!>