Simple Reactive Robot

Carlos Veríssimo
Department of Informatics Engineering
FEUP
Porto, Portugal
up201907716@up.pt

Miguel Amorim
Department of Informatics Engineering
FEUP
Porto, Portugal
up201907756@up.pt

Rafael Camelo
Department of Informatics Engineering
FEUP
Porto, Portugal
up201907729@up.pt

***Abstract*—In an era defined by dynamic and ever-evolving technological landscapes, the development of a REACTIVE (Real-time, Environment-Adaptive, and Contextually-Intelligent) robot under the aegis of the Robot Operating System (ROS) represents a pivotal step forward in the realm of robotics. This ambitious project aspires to create a robotic entity with unparalleled adaptability, reactivity, and intelligence in response to its environment. The simulation is performed in ROS, which runs the Python code and integrates it with the Gazebo simulation environment. The reactive robot is generated in a random start position. It can successfully follow the wall, avoid hitting, and stop in the desired end position.**

***Index Terms*—Robotics, Intelligent, ROS, Gazebo, Simulation, Sensor, Actuator, Navigation, Environment**

I. Introduction

As technology continues to advance, autonomous mobile robots are increasingly finding applications across various sectors, including corporate environments, industrial settings, healthcare facilities, educational institutions, agriculture, and even in everyday households. In the context of mobile robot navigation, many scenarios require the implementation of wall-following behaviors. These behaviors allow the robot to navigate along the contours of walls or obstacles while maintaining a safe and consistent distance from them. These autonomous robots exhibit the capability to operate in diverse environments, which can be characterized by non-linearity and partially real-time observations. To address challenges of this nature, a multitude of techniques and solutions have been explored and studied.

II. State of the Art

Robot navigation is a fundamental aspect of robotics that focuses on enabling robots to move autonomously and safely in their environment. It involves the development of algorithms, sensors, and control systems to guide robots in tasks such as exploration, mapping, path planning, and obstacle avoidance. Here are some key aspects of robot navigation: **Sensors -** Robots rely on a variety of sensors to perceive their surroundings. These sensors include cameras, LIDAR (Light Detection and Ranging), ultrasonic sensors, inertial measurement units (IMUs), encoders, and more. These sensors provide data on the robot's position, orientation, and the presence of obstacles. **Simultaneous Localization and Mapping (SLAM)**: SLAM is a critical technology in robot navigation. It enables a robot to build a map of its environment while simultaneously determining its own position within that environment. This is essential for autonomous navigation and exploration. **Path Planning**: Path planning algorithms help robots find a safe and efficient route from their current location to a target destination while avoiding obstacles. These algorithms take into account the environment's layout and the robot's physical constraints. **Local vs. Global Navigation**: Robots often employ a combination of local and global navigation strategies. Local navigation focuses on short-term decisions, like avoiding immediate obstacles, while global navigation involves planning routes to long-term goals. **Reactive vs. Deliberative Navigation**: Reactive navigation involves immediate reactions to sensory input, while deliberative navigation considers a broader context and plans actions in advance. Many robots use a combination of both. **Dynamic Environments**: Navigating in dynamic environments with moving objects or people requires advanced navigation systems that can predict and adapt to changes in real time.

Robot navigation is a complex and evolving field with applications in areas such as autonomous vehicles, industrial automation, search and rescue, space exploration, and agriculture. As technology continues to advance, robots are becoming increasingly capable of navigating a wide range of environments and scenarios.

III. Robot Implementation and Architecture

A. Robot Specification

For the implementation of the reactive robot, a simulated version of the Turtlebot3 [1] robot. Throughout the development, we mostly used the *burger* model of Turtlebot3 to test our simulation, however, one can change the model just by using a different environment variable. More on that in section III-D.

The robot is equipped with a LIDAR sensor, which is used to detect the distance to the walls and obstacles. The robot also has two wheels, controlled by the *wheel_left_joint* and *wheel_right_joint* joints.

Figure III-A shows the Turtlebot3 Burger in the real world and its simulated counterpart in Gazebo.

B. The Environment

The simulation environment is a question-marked shaped world. It was created using the Gazebo building editor tool.

[1] https://www.turtlebot.com/turtlebot3/

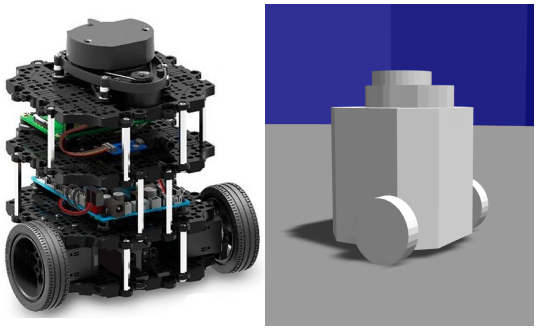

Fig. 1. Turtlebot3 Burger in the real world (left) and in Gazebo (right)

Project specifications specify that the bottom is straight with an edgy shape.

Below is a visualization of the environment, in Gazebo:

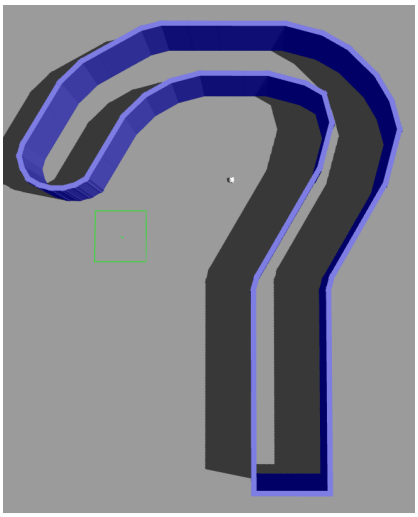

Fig. 2. The simulation world

C. Packages

Inside our workspace, two packages were created: *my_reactive_robot* and *my_controller*.

The first package:

- Launches Gazebo, by running both *gzserver.launch.py* and *gzclient.launch.py*
- Publishes the robot state with the *robot_state_publisher* node
- Spawns our Turtlebot3 robot in the simulation environment, with the *spawn_entity.py* service, from the package *gazebo_ros*

The second package, as the name suggests, is responsible for controlling the robot. It contains the following nodes:

- *scan_to_velocity_node*: This node subscribes to the *scan* topic, which contains the LIDAR data, and publishes the velocity commands to the *cmd_vel* topic, which controls the robot's wheels.
- *spin _randomly _node*: This node is a workaround that we created to allow for a random initial robot orientation. It spins the robot randomly for a few seconds and then stops it.

The robot's initial position and pose are random, but are constrained to the round area of the question mark.

D. How to Run the Simulation

To run the simulation, you will need the following ROS packages/dependencies:

- *turtlebot3_gazebo*: This package contains the Turtlebot3 models and the Gazebo plugins.
- *gazebo_ros*: This package contains the Gazebo ROS interface.

Assure that you have the necessary dependencies and that you have sourced your ROS distribution's setup file. Please refer to the README file in the project's repository, if you're having any issues with this process.

Build the package and source the setup file:

1) `cd FEUP-RI-PROJ` - Go to the project's root directory
2) `colcon build --symlink-install` - Build the packages
3) `source install/setup.bash` - Source the setup file
4) `export TURTLEBOT3_MODEL=burger` - Set the Turtlebot3 model. You can change this to *waffle* or *waffle_pi*.

Run the simulation:

1) Open two terminal windows.
2) Run `ros2 launch my_reactive_robot launch.py` in one of them.

This will launch Gazebo and spawn the robot in the simulation environment. Please wait for Gazebo to load before running the next command.

In the other terminal window, run `ros2 run my_controller controller_node`. This will launch the nodes that control the robot.

Please note that we set up Gazebo so that it starts with a paused state. This means that the simulation will not start until you press the play button in Gazebo's GUI.

E. Implementation

IV. Experiments

V. Results and Discussion

VI. Conclusions and Future Work

VII. Acknowledgments

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.