Reactive Robots using ROS2 and Webots

Adriano Machado, Félix Martins, Francisco da Ana, Martim Iglesias
Faculty of Engineering, University of Porto (FEUP)
Porto, Portugal

***Abstract*—This study presents the development of two reactive robots using ROS2 and Webots. One robot follows a wall using reactive control, while a second robot either follows the first or also follows the wall. The project is based on a differential locomotion strategy and ensures no memory is used. A baseline solution and a more robust approach were implemented. The results demonstrate performance in a question-mark-shaped environment, with a discussion on the improvements made to the control strategies.**

***Index Terms*—Reactive Robotics, ROS2, Webots, Autonomous Systems, Differential Locomotion, Behavior-Based Control**

I. Introduction

Reactive robot architectures enable real-time responses to environmental stimuli without relying on memory. This project focuses on developing two memory-less robots: one follows a wall, while the other either follows the first robot or the wall. Implemented in ROS2 and simulated in Webots, this study explores the effectiveness of reactive control strategies for autonomous navigation.

The ability to adapt and respond in real time is crucial for autonomous robots. Behavior-based approaches allow mobile robots to interact with their surroundings without requiring pre-programmed paths or map-based localization. This work extends existing studies on reactive architectures by testing their performance in a specific environment.

II. State of the Art

Reactive control in robotics is based on real-time sensor feedback rather than pre-planned navigation. The subsumption architecture, introduced in [2], is a foundational model for behavior-based robotics. This architecture structures control into layers, where lower layers handle basic sensorimotor behaviors, and higher layers suppress or override them when needed. This layered control system demonstrated how robots could navigate complex environments without needing extensive world models. This was a breakthrough in reactive navigation and directly influenced subsequent research in robotic autonomy.

Murphy's work on AI robotics [4] expands on these ideas, emphasizing the trade-offs between reactive and deliberative approaches. Deliberative architectures involve planning and world modeling, whereas reactive architectures focus purely on real-time stimulus-response behaviors. Murphy highlights cases where combining reactive techniques with higher-level reasoning can improve performance, particularly in structured environments where path planning is beneficial.

Arkin's research in behavior-based robotics [1] builds upon Brooks' subsumption architecture by formalizing behavioral interactions and control strategies. He introduced the concept of schema-based control, where multiple simple behaviors compete or cooperate to generate movement. This method has been widely used in mobile robot applications, particularly in swarm robotics and autonomous exploration tasks.

Modern developments have sought to integrate traditional behavior-based robotics with AI-driven learning methods. Reinforcement learning has been used to refine behavior-based control, allowing robots to adaptively optimize their responses to environmental stimuli. This hybridization helps address some of the limitations of pure reactive control, such as dealing with unpredictable or highly dynamic environments.

Another significant improvement in reactive control strategies is sensor fusion, which enhances a robot's ability to interpret its surroundings. Previous works [1] have incorporated multiple sensor modalities, such as LiDAR, vision, and ultrasonic rangefinders, to increase navigation robustness. These approaches improve reactive decision-making, allowing robots to distinguish between obstacles, terrain types, and dynamic objects.

Our approach focuses solely on reactive robot architectures, specifically designed to navigate a simple environment, drawing inspiration from the behavior-based frameworks of [2] and [1]. By integrating proximity sensors and LiDAR, we leverage distinct sensor modalities to differentiate obstacles and track dynamic targets, thus addressing a key limitation of traditional reactive systems. This design aligns with the trend toward sensor fusion while maintaining the efficiency of real-time control.

III. Implementation and Architecture

For this project, we explored two distinct architectures.

The first approach relies solely on proximity sensors, serving as a baseline implementation. Given the task's apparent simplicity, where robots navigate based on local obstacles, this minimalist strategy may work well. By using minimal sensing, it ensures efficiency while still delivering near-optimal behavior, aligning with the task's objectives.

As we will note in Section IV, this simple task may actually require more complex measurements. To address this, the other implementation uses a LiDAR alongside the proximity sensors. This powerful sensor enables finer-grained control by providing detailed environmental data, allowing a robot to distinguish between a wall and another robot, responding accordingly.

The system in both implementations consists of two autonomous robots with differential locomotion, managed using

ROS2 nodes. The wall-follower robot has the same architecture in both implementations. In addition, in both approaches, the robot-follower is faster than the wall-follower to make the experiments more dynamic and provide better insights.

A. Baseline Architecture

This baseline architecture uses only proximity sensors, adjusting the angular velocity according to the measurements and the predefined linear velocity. Both robots use two proximity sensors, one pointed left and the other pointed right. The corresponding ROS2 topics and nodes are illustrated in Fig. 1.

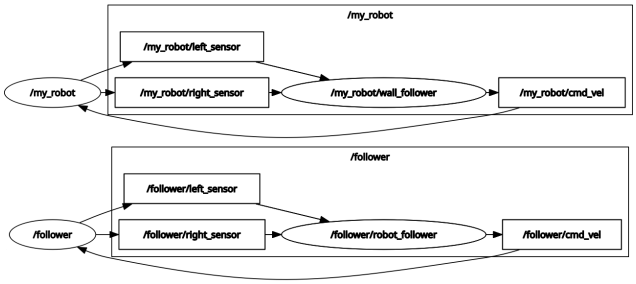

Fig. 1. Baseline architecture: ROS2 nodes and topics as shown in rqt graph. `my_robot` refers to the wall-follower robot, while `follower` denotes the robot-follower.

1) Wall Follower Robot: Maintains a constant linear velocity while dynamically adjusting its angular velocity based on the distance sensor measurements. The wall-following algorithm functions by continuously monitoring the distance to the nearest obstacle and making real-time corrections to maintain a parallel trajectory. Specifically, if the robot detects that the wall is too far away—beyond a predefined distance threshold—it turns right to move closer to the wall. On the contrary, if it is too close, it turns left to create more space. The angular velocity of these turns scales proportionally with the predefined linear velocity, enabling sharper corrections. As a result, the robot consistently follows walls in a designated direction.

2) Robot Follower: Operates similarly to the wall-follower. It keeps a constant linear velocity and only adapts its angular velocity based on the distance sensors. It follows either a wall or the other robot. In addition, when it detects the other robot is too close, it avoids a collision by treating the robot in a similar way to how it responds to a wall. Otherwise, if the other robot is out of range, the robot follower simply reverts to wall-following behavior.

B. Improved Architecture

Compared to the baseline approach, this architecture adds a LiDAR to the follower robot. The wall-follower robot, however, remains very similar, with only minor adjustments, such as a modified linear velocity. The corresponding ROS2 nodes and topics are shown in Fig. 2.

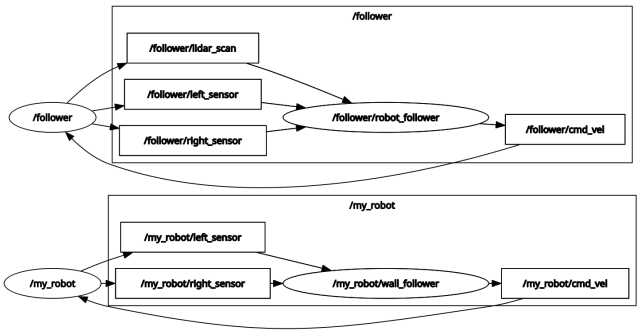

Fig. 2. Improved architecture: ROS2 nodes and topics as shown in rqt graph. `my_robot` refers to the wall-follower robot, while `follower` denotes the robot-follower.

For the follower robot, instead of just using proximity sensors to detect walls and the robot, treating both similarly, we use the LiDAR sensor to detect a point cloud. The LiDAR used has a fixed field of view of $90°$ and a defined range limited to 0.5 meters, without rotation. Using the generated point cloud, we apply a robot detection algorithm to determine the relative angle of the other robot, enabling the follower to turn towards it.

In this architecture, proximity sensors handle wall detection, while the LiDAR focuses on identifying the other robot. Additionally, if the follower detects the other robot too close, it will stop to avoid collisions. To make the environment more dynamic, enabling more interactions between the robots, we set the wall-follower's linear velocity lower than that of the follower robot.

1) Robot Detection Algorithm: This algorithm needs to leverage LiDAR measurements to determine the relative angle to another robot, given that it is within detectable range. Each LiDAR measurement provides distance data across the environment, discretized by small angular increments. From these measurements, we derive point coordinates using the distance, the LiDAR's minimum and maximum angles, and its angular increment.

The algorithm begins by clustering nearby points. We group consecutive points into the same cluster if the distance between them falls below a specified threshold. This process effectively aggregates points belonging to continuous objects, such as walls or the robot.

Next, we attempt to identify circular shapes within each cluster using a RANSAC-based circle-fitting method [3]. For each cluster, we randomly sample three points in several iterations to define a circle and count the number of other points (inliers) lying within a small threshold of its boundary. The circle with the highest inlier count is selected as the best fit. Additionally, for a circle to be classified as a robot, it must exceed a minimum inlier threshold and have a radius approximately matching that of the wall-follower robot, which is 0.045 meters.

Once the robot is detected, we use a proportional control strategy to orient the robot towards the angle of the corresponding circle's center. Using the LiDAR's angular

references, we calculate the target angle and set the robot's angular velocity accordingly.

This approach relies on several hyperparameters, including various thresholds. We will explore its results in Section V.

IV. Experiments

The primary goal of this project was to develop two robots capable of navigating a question-mark-shaped environment. One robot consistently follows the wall, while the other either follows the wall or tracks the first robot. Rather than always pursuing the nearest object, we configured the second robot to prioritize tracking the first robot when it is detectable within a short range. This design choice adds complexity to the experiment, as the wall is almost always the closest object.

At first glance, this task appears straightforward and potentially achievable with basic sensors. However, it requires distinguishing between a wall and another robot, a challenge that proximity sensors alone cannot reliably address.

For these experiments, we assumed the robots begin near the wall they are intended to follow. Otherwise, starting too far from the wall could lead them to either track an unintended wall or fail to follow any wall entirely.

Fig. 3. Question-mark-shaped environment

Fig. 3 illustrates the environment used in our experiments. The question mark was designed in Blender and then imported into Webots, while also adding proper physical properties required for capturing sensor data.

To evaluate the effectiveness of the reactive control strategies, we measured the loop completion time. This was recorded manually, as automating this measurement was unnecessarily complex. Furthermore, we equipped each robot with a colored pen matching its own color, enhancing the visualization of their traveled paths and enabling a qualitative assessment of their performance.

V. Results and Discussion

The performance of our reactive robots was evaluated through a series of experiments navigating around a question mark-shaped obstacle. The study focused on several key aspects of robot movement and interaction, revealing insights into trajectory control, collision avoidance, and navigation strategies.

A. Trajectory Characteristics

Loop time analysis revealed an inversely proportional relationship with linear velocity. As demonstrated in Table I, changing the linear velocity from 0.06 to 0.12 nearly halved the loop completion time. This observation highlights the significant impact of velocity settings on robot navigation.

Linear Velocity	Loop Time
0.06	2'32"
1.12	1'16"
0.08	1'56"

TABLE I
Loop Time measurements for different linear velocities

B. Trajectory Comparison: Baseline vs. Advanced Approach

Fig. 4 illustrates the trajectories of robots using two different navigation approaches when circumnavigating a question mark-shaped obstacle positioned at the map's center.

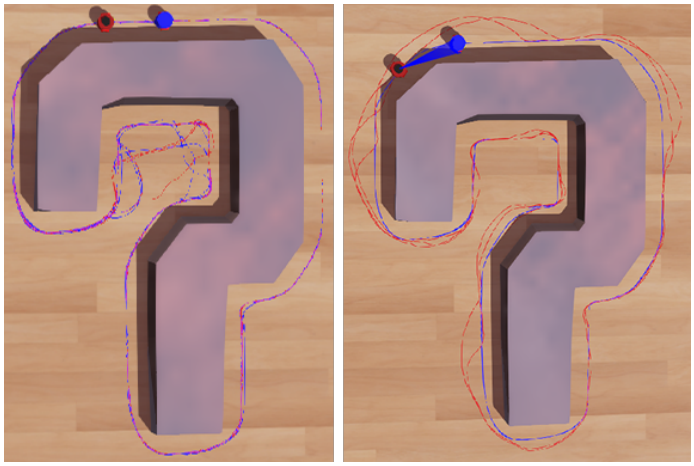

Fig. 4. Robots trajectories after a few loops. The left image results from the baseline architecture, while the right image represents the trajectories in the improved architecture

The baseline approach demonstrated consistent exterior movement, maintaining a parallel trajectory at a fixed distance from the wall. However, in the interior sections with reduced space, robots exhibited increased deviations to avoid collisions. This led to potential complications in navigation, particularly in these confined areas.

In contrast, the advanced approach incorporating a LiDAR sensor significantly improved navigation performance. This implementation resulted in a more uniform trajectory, maintaining consistent parallelism with walls throughout the entire perimeter. Notably, the advanced approach eliminated trajectory deviations in the central region of the question mark, primarily due to the follower robot halting when it detects the other robot in close proximity.

C. Collision Avoidance and Robot Interaction

In the initial implementation, robots frequently encountered collision and stalling issues. Experimental observations revealed that collisions typically occurred after a few turns, with the frequency influenced by the specific geometry of the question mark. The sharp edges of the obstacle contributed to trajectory perturbations, exacerbating collision risks.

Collision hotspots were predominantly located in the central, more confined region of the question mark, where limited space restricted the robots' ability to deviate from each other. In some instances, robots would become temporarily stuck against walls—a consequence of collision avoidance movements rather than inherent wall-following trajectories.

The advanced LiDAR-based implementation effectively mitigated these challenges. The key innovation was the follower robot's ability to dynamically adjust its speed, slowing down or stopping when detecting proximity to the leading robot. This adaptive behavior substantially reduced collision probabilities and improved overall navigation reliability.

D. LiDAR Robot Detection

The approach used in the improved architecture to detect a robot is not completely robust. In fact, it has many hyperparameters which we adjusted in order to have better performance. While it avoids getting both robots stuck, it sometimes leads to false negatives. This means that the other robot may be contained within the LiDAR point cloud, partially or not, but our algorithm does not detect it. However, it avoids false positives in general, so it is still able to follow the other robot successfully and stop in case it detects it too close.

E. Velocity and Movement Characteristics

It is important to note that the current implementation maintains constant linear velocities, with no provisions for linear acceleration, except when stopping in the LiDAR approach. This simplifies the control mechanism but may limit the robots' adaptive capabilities in more complex environments.

VI. Conclusions

This study demonstrates the effectiveness of reactive architectures in autonomous navigation, particularly in structured environments where immediate responsiveness to sensory input is critical. The implementation of a wall-following and a robot-following behavior in ROS2 and Webots allowed for real-time adaptation without reliance on memory or map-based localization. Our results show that while simple proximity sensor-based navigation is feasible, the integration of LiDAR significantly improves robot differentiation and obstacle avoidance, reducing instances of collision and deadlock.

However, the experiments also highlight some limitations. The LiDAR-based robot detection algorithm, while functional, is susceptible to occasional false negatives. Additionally, the purely reactive nature of the robots means that they lack the ability to handle long-term navigation challenges, such as re-engaging with their intended trajectory after a significant disturbance.

Despite these limitations, this work reinforces the viability of behavior-based robotics for real-time autonomous navigation and provides a foundation for further improvements in reactive control strategies.

VII. Future Work

While this work successfully developed two fully reactive robots capable of navigating the environment presented, certain limitations highlight areas for improvement. One key limitation is that the robots must start near the wall they intend to follow. Otherwise, they may follow the closest wall arbitrarily or fail to navigate if too far from any walls.

Future work could explore ways to enhance the robots' reactivity, within the constraints of not introducing memory. For instance, future improvements could include:

- Improvement of wall-following algorithms to make the robots' behavior more robust in unpredictable environments
- Refinement of LiDAR-based detection: The current robot detection algorithm is parameter-sensitive and sometimes fails to recognize the other robot. Improving the robustness of the detection, potentially by optimizing RANSAC thresholds or incorporating additional heuristics, could enhance performance.
- Improving real-world applicability: The current experiments were conducted in a structured setting with controlled obstacles. Future work should evaluate performance in more complex and dynamic environments, where sensor noise and unexpected interferences could impact reactive navigation. Testing under these conditions would provide deeper insights into the system's robustness.
- Hybrid reactive-deliberative models: While this study focused on purely reactive control, incorporating limited predictive elements could enhance adaptability without compromising the system's real-time responsiveness.

Further research could also investigate alternative reactive navigation strategies beyond wall-following, enabling the robots to navigate more complex environments while remaining fully reactive. Expanding the range of tested environments and fine-tuning sensor-based behaviors could further improve performance and reliability.

Acknowledgments

We thank the Topics in Intelligent Robotics course instructors and FEUP for supporting this project.

References

[1] R.C. Arkin. *Behavior-based Robotics*. Bradford book. MIT Press, 1998.
[2] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986.
[3] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.
[4] R. Murphy. *Introduction to AI Robotics*. A Bradford book. MIT Press, 2000.