Wall-following in a reactive robot with differential locomotion using a laser scanner sensor

César Nogueira
FEUP
Porto, Portugal
up201706828 @edu.fe.up.pt

Filipe Barbosa
FEUP
Porto, Portugal
up201909573@edu.fe.up.pt

Pedro Galvão
FEUP
Porto, Portugal
up201700488@edu.fe.up.pt

***Abstract*—This article discusses the development of an algorithm for a reactive wall-following robot. An algorithm based on trigonometric functions was implemented and tested with a differential drive robot. The testing environment was a 'B'-shaped wall, which the robot should be able to follow, both from the inside and the outside. By adjusting some of the parameters during the testing, we determined the impact of each of them on the lap time and when the robot performs best.**

Keywords—robotics, autonomous navigation, wall-following, reactive robot, differential locomotion robot

I. Introduction

The wall-following problem is widely known and useful for autonomous navigation. It consists of developing an algorithm to enable a robot to drive alongside a wall. In this paper, we explore that problem, mainly the current state of the art and the details of a possible implementation that solves it. We also carry out some experiments and interpret their results.

II. State of The Art

Several solutions to the wall-following problem have been proposed using different methods. Those methods vary a lot both in the type of algorithm used and type of sensors, robot and environment.

Regarding the architecture, more sophisticated solutions can be found using deliberative or behaviour-based approaches [1]. Some algorithms include localization and mapping to allow the robot to build an internal representation of its environment and exploit this information to improve its efficiency [2]. However, for the purposes of this article, we will focus only on reactive architectures.

Applications of reactive algorithms such as the one presented here can be found for example in [3], which details a system for autonomous navigation of underground mining vehicles based on a robust reactive wall-following behaviour.

Different approaches can be used when implementing a reactive algorithm. Some of them include the use of machine learning techniques, such as [4], which uses genetic programming, or Reinforcement Learning, a more common approach, which can be found in recent papers such as [5]. However, for our purposes, good results can be achieved without machine learning, using instead an algorithm with explicit rules based mainly on mathematical equations and trigonometry to keep the robot moving at a fixed distance from the walls.

III. Implementation and Architecture

A. Sensor

We used a Light Detection And Ranging (LiDAR) sensor. This kind of sensor uses Laser beams to detect and estimate distances to obstacles around itself. The LiDAR used is capable of identifying distances 360º around the robot. It has a maximum range of 3.5 m and a minimum range of 0.12 m. Due to this minimum distance, it is important to guarantee that the robot does not come too close to the walls at any moment, because that would make it impossible to detect.

We did not need to use the distance values from all the angles around the robot. Instead, we adopted a simpler approach, dividing the scan into 9 regions and collecting the minimal value from each one of these. These 9 regions are in the frontal and lateral parts of the robot, each one collecting values in a range of 9 degrees. The regions cover the following areas: [-94º, -86º], [-72º, -64º], [-49º, -41º], [-19º, -11º], [-4º, 4º], [11º, 19º], [41º, 49º], [64º, 72º], [86º, 94º].

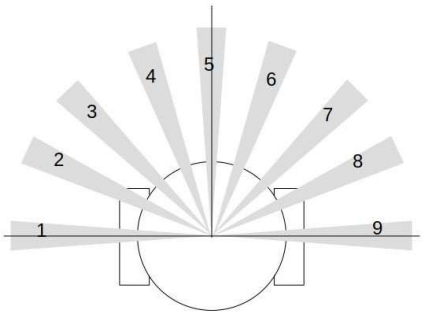

Figure 1 - LiDAR scan regions

B. Tools

We developed the algorithm using **Python** on the **ROS middleware suite**. The functionalities included in ROS allowed us to get input from sensors and send movement commands to the robot. For the simulation, we used **Gazebo**, a 3D robotics simulator, which can be easily integrated with ROS.

The robot model used was the **Turtlebot3** developed by Willow Garage, a robotics research laboratory. It is a low-cost robot equipped with differential driving, its software is open source, and it has integration with ROS.

C. Algorithm

The algorithm is based on the ***Virtual Triangle Wall Follower* algorithm** [6], which uses trigonometric functions. In short, it uses the distances captured by the sensor to create a model of a right triangle, which then allows the calculation of how much the robot must turn to get to the desired distance from the wall.

The algorithm uses two values that must be previously defined: D_{wall} (*wall distance*), which represents how far the robot wants to be from the wall, and *Wall_lead*, which influences the robot's turning speed.

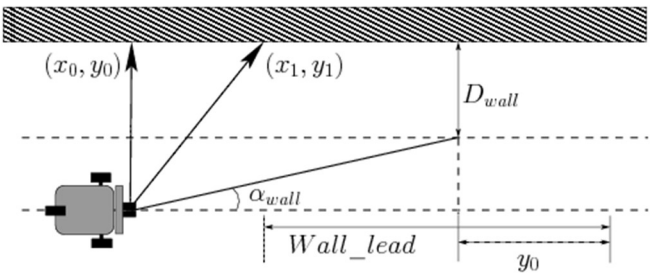

Figure 2 – Wall-following algorithm model
Source: [6]

By getting the distance to the wall from 2 different angles, the length of the catheti of the modelled triangle can be obtained, and consequently the angle closest to the robot. It is calculated using the formula (1).

$$\alpha_{wall} = \text{atan}^2((y_1 - D_{wall}), (x_1 + W_{lead} - y_0)) \quad (1)$$

On top of this, some modifications were applied. Three different states are defined, and the robot goes through them depending on what it senses. In the first state, called the **Wandering** state, the robot randomly changes direction every 6 seconds, until it senses a wall. In the second state, it turns to the wall and drives forward until it is closer to the wall. In the third and final state, it applies the aforementioned formula with a small fix to avoid hitting the wall. If it is too close (0.5 meters) to the wall, the angular speed is increased, as to turn more aggressively.

IV. Experiments

To test the algorithm's ability to follow walls, we presented the robot with two different environments, each one presenting a different challenge. Both maps have a 'B'-shaped wall, that the robot must try to circuit along, both from the outside and the inside.

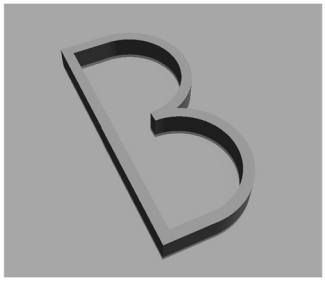

Figure 3 - Plain 'B' Wall

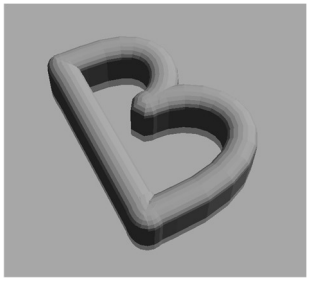

Figure 4 - Rough 'B' Wall

The environments do not have same wall perimeter length but are appropriate to test the robot's ability to go around different types of walls: the plain 'B' wall has smoother and gradual turns, as well as sharp 90º turns, while the rough 'B' wall presents the robot with more inconsistent curves.

The tests that were performed analysed the influence of the *linear speed* and the *wall distance* on the time that it took the robot to go around the whole wall, be it from the outside or from the inside. The following parameters were adjusted for each test:

1) *Linear speed* tests:
   - Ranging from 0.1 m/s to 0.3 m/s, with incremental steps of 0.05 m/s.
2) *Wall distance* tests:
   - Ranging from 0.25 m to 1 m, with incremental steps of 0.15 m.

In each of the experiments, the default value of the other parameters was used: 0.1 m/s for *linear speed* and 0.2 m for *wall distance*.

Only the times of successful runs are registered, being that a *successful run* is a run in which the robot succeeds on going around the whole wall, or in other words, getting back to the position it started moving. It is expected that in some of the experiments, the robot will fail to reach that goal.

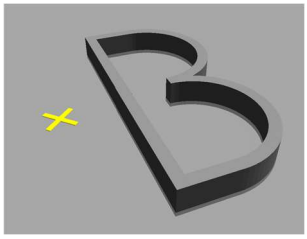

Figure 5 - Plain 'B' Wall Outside Start Position

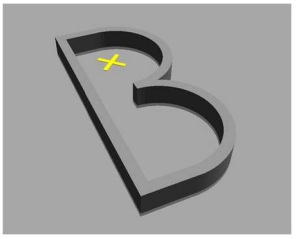

Figure 6 - Plain 'B' Wall Inside Start Position

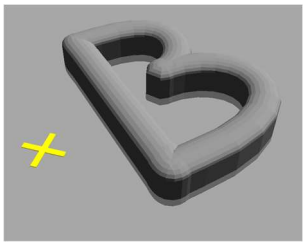

Figure 7 - Rough 'B' Wall Outside Start Position

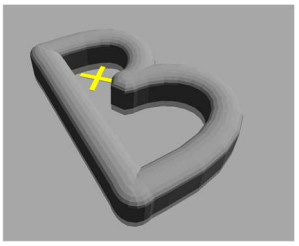

Figure 8 - Rough 'B' Wall Inside Start Position

rqt_graph was used to plot the graphs that are presented in the results. All registered times are based on Gazebo's simulation time and not real time, so that no external factors (like computer lag) influence the results.

V. Results and Discussion

The presented results consist of one trial run for each scenario presented in the previous section, by adjusting either the *linear speed* or the *wall distance*.

A. Linear speed

TABLE I. Linear Speed Experiment Results

Linear Speed (m/s)	Lap Times (s)			
	Plain 'B' Wall		*Rough 'B' Wall*	
	Inside	*Outside*	*Inside*	*Outside*
0.10	181.429	221.692	115.314	192.689
0.15	121.864	149.051	78.911	129.169
0.20	92.709	112.757	60.228	97.844
0.25	74.652	N/A[a]	N/A[a]	80.822[b]
0.30	N/A[a]	N/A[a]	N/A[a]	N/A[a]

[a] Robot got stuck and didn't finish lap.
[b] Possible outlier.

Increasing the speed resulted in reduced lap times. However, with increasing speed, comes less time for the robot

to react, making it more error prone. This is visible at a speed of 0.3 m/s, where the robot was unable to complete the lap in any of the scenarios, since it got stuck. At a speed of 0.25 m/s we got similar results, since it got stuck most of the times. With this same speed outside the rough 'B' wall, subsequent tests resulted in the robot getting stuck, even though it was able to complete the lap in the test presented in the table.

At a speed of 0.2 m/s is where a relatively good success rate was noticed, while maintaining somewhat low lap times. Although lap times were considerably higher, 0.1 m/s is the ideal speed for avoiding collisions.

B. *Wall distance*

TABLE II. WALL DISTANCE EXPERIMENT RESULTS

Wall Distance (m)	**Lap Times (s)**			
	Plain 'B' Wall		***Rough 'B' Wall***	
	Inside	***Outside***	***Inside***	***Outside***
0.25	177.668	223.805	114.122	194.886
0.40	170.432	234.271	110.565	204.826
0.55	N/A[c]	241.553	N/A[c]	212.773
0.70	N/A[c]	273.487	N/A[c]	240.798
0.85	N/A[c]	318.438	N/A[c]	272.931
1.00	N/A[c]	346.200	N/A[c]	298.778

[c.] Robot got stuck on the upper section of the 'B' and didn't finish lap.

As it was expected, when following the wall from the outside, the robot was faster when the wall distance was decreased (less distance to travel). Likewise, when on the inside, the lap times got better when the wall distance was increased (once again, less distance to travel).

However, due to the space limitations inside the 'B', increasing the wall distance too much resulted in failure, as the robot was not able to pass through the middle part of the letter and got stuck circling around the upper section. In the case of the higher values, it couldn't even leave its starting position and just rotated in place.

When following from the outside, the robot was always able to finish the lap. However, with a wall distance of 0.70 m and higher, it moved very frantically, not being able to consistently follow the wall parallel to it, even sometimes doing sharp 180º turns and having to correct its direction afterwards (Figure 9). That effect diminished as the parameter value was lowered (Figure 10). We believe this happened due to the *Wall_lead* parameter, that was left unchanged despite the *wall distance* parameter changes.

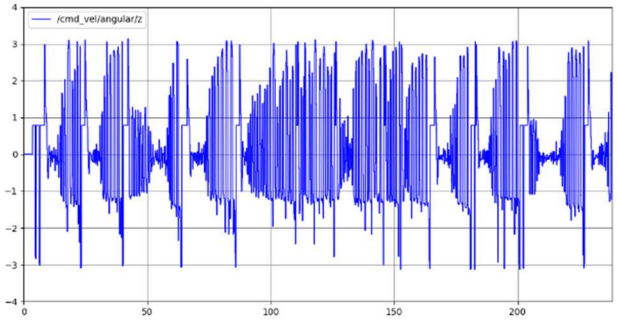

Figure 9 - Angular Velocity Variation by Time while following rough 'B' wall from the outside with a wall distance of 0.70 m

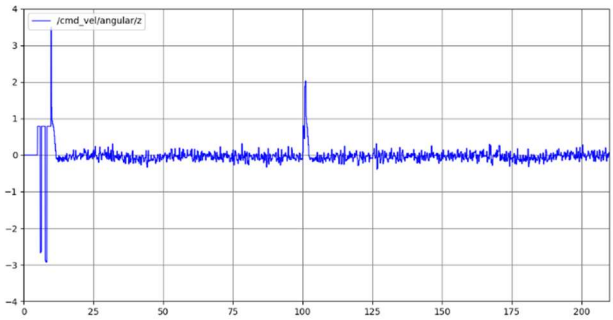

Figure 10 - Angular Velocity Variation by Time while following rough 'B' wall from the outside with a wall distance of 0.55 m

For both maps, the wall distance values that resulted in the best loop times were low, around 0.40 m and 0.25 m. It would be possible to go lower, but discouraged since the minimum distance that the LiDAR can detect is 0.12 m. If by any chance the robot got to close to the wall, it wouldn't behave as intended and would simply crash into the wall (it would not know that the wall exists).

VI. CONCLUSIONS AND FUTURE WORK

In this study we were able to successfully achieve our predefined goals and program a robot that in most cases can complete the circuit around the map, both from inside and outside. The few exceptional cases in which it did not complete this task were cases in which parameters, such as the distance to the wall were configured in inconvenient values that made it impossible to keep the desired distance in parts of the map. In such cases the robot failed, but it was already expected.

A possible improvement to the algorithm would be to develop a mechanism to solve these error cases, allowing the robot to pass between two close walls without problems despite its desired distance to the wall being too high. A possible fix would be to temporarily reduce this desired distance when faced with this kind of situation.

Other improvements could be done by using specific information about the map. It would be possible to use the knowledge about our maps to develop specific rules, telling the robot how to behave in certain locations. Likewise, during the wandering state, where the robot is searching for the wall, some kind of memory could be implemented, so that it does not search the same area more than once.

REFERENCES

[1] A. Adriansyah and S. Amin, "WALL-FOLLOWING BEHAVIOR-BASED MOBILE ROBOT USING PARTICLE SWARM FUZZY CONTROLLER", Jurnal Ilmu Komputer dan Informasi, vol. 9, no. 1, p. 9, 2016.

[2] M. Finkelstein and B. Hunte, "Wall Following with Collision Avoidance and Mapping Using a Laser Range Finder", 2008.

[3] J. Roberts, E. Duff, and P. Corke, "Reactive navigation and opportunistic localization for autonomous underground mining vehicles," Information Sciences, vol. 145, no. 1-2, p. 127–146, 2002.

[4] R. Dain, Applied Intelligence, vol. 8, no. 1, pp. 33-41, 1998.

[5] C. Chen, S. Jeng and C. Lin, "Mobile Robot Wall-Following Control Using Fuzzy Logic Controller with Improved Differential Search and Reinforcement Learning", Mathematics, vol. 8, no. 8, p. 1254, 2020.

[6] T. Bower, "5.5.1.4. Wall Following - Robotics Programming Study Guide", Robotics Programming Study Guide. [Online]. Available: http://faculty.salina.k-state.edu/tim/robot_prog/MobileBot/Algorithms/WallFollow.html. [Accessed: 26- Nov- 2021]