A reactive two-wheeled robot to follow walls

Filipa Ivars Silva
Faculty of Engineering of University of Porto
Porto, Portugal
ivars.filipa@gmail.com

***Abstract*—Autonomous Vehicles popularity is growing. Reactive robots have their place in the development of new technology to potentiate Autonomous Vehicles capabilities. We present wall-following algorithm and propose a two-wheeled reactive robot to analyze it. The analysis is concluded over five different environments and the capacity to properly follow the wall in its shape representative way is evaluated and further improvements are proposed.**

***Index Terms*—laser, reactive, robot, ROS, wall, wall-following**

I. Introduction

As the popularity of Autonomous Robots arises, also the potential applications number grows to fit the society needs [1]. We are proposing a simple algorithm to empower a robot to follow walls.
The robot structure is introduced as a two-wheeled robot with a laser sensor to detect the external environment obstacles. The algorithm that interprets the laser input and produces a reaction to the robot is presented as a states machine.
Five different environments are used and and the robot performance in them is analyzed. These worlds aim to reproduce several environments to test with different angles between walls and if the length of the wall impacts the straight line following stability. The results of these experiments are analyzed and improvements are suggested at the end.
The environments and the respective experiments were built and performed in Gazebo, and the algorithm was implemented in ROS using Python language.

II. The Robot Model Structure

As mentioned, our proposal is a reactive robot which use a laser sensor to obtain responses to stimulus from the environment. The robot model structure is represented in the Figures 1 and 2. It's composed by two wheels, one caster wheel, one chassis and one laser sensor, with their respective links and joints.
The laser sensor is simulated using the Gazebo ROS plugin *libgazebo_ros_laser*.

A. Assumptions

In order for us to create a representation of a robot that can interact with the environment, it was required to create some assumptions about the sensor such as the division of the 180 degrees angle perceived by the sensor into five regions, as represented in the Figure 3. For the purpose of our strategy to simplify the way our robot finds the wall for the first time, we assumed that the wall will be located at the right side of the robot once the simulation starts. This decision was due to the assumption that the wall will be followed by the robot having the wall at its right side. Therefore, the state machine will be simplified if the first state to find the wall is to search for it by navigating always to the right. This way, this state could be reused every time the robot looses the position of the wall on its right, then it will continue to search for the wall, going forward and right.
This assumption proved limiting because, if the wall isn't on the robot right side, it will keep searching in looping circles not being able to ever find the wall (see Figure 4).

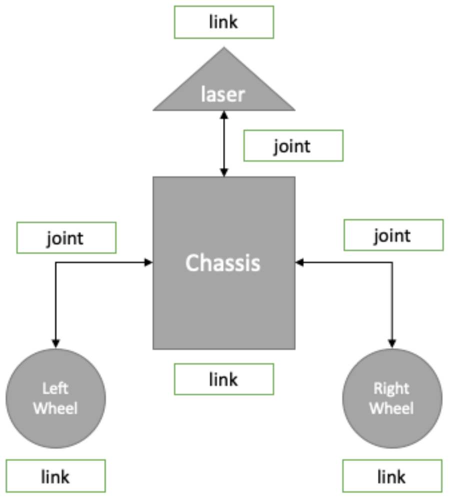

Fig. 1: The robot model structure

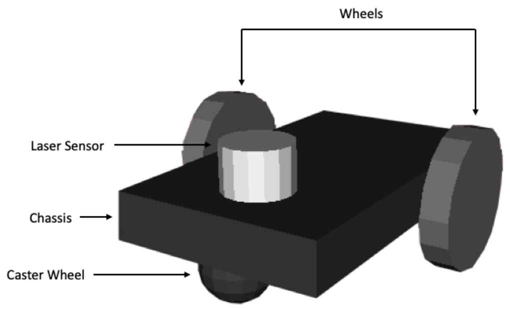

Fig. 2: The robot

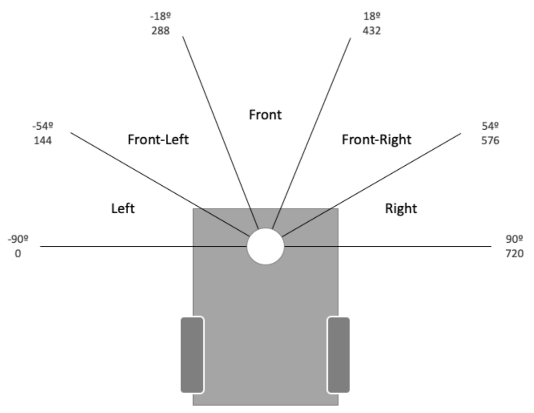

Fig. 3: The laser scan representation

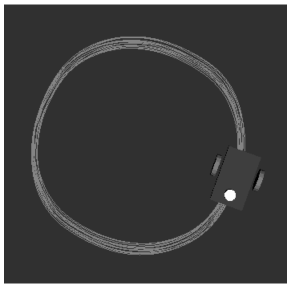

Fig. 4: Robot trajectory searching for a wall located out of reach

III. The Algorithm

As stated before, our algorithm starts with a 'always-right' strategy to find the wall. Subsequently, once the wall is detected in the front, it will turn left so that the wall remains on the robot's right side. This way, we were able to ensure that the wall will be followed by the right side of the robot considering the minimum and maximum distances of 1 m and 0.8 m, respectively. The algorithm can be represented by the tables I and II.

At the table I, the position *left* was intentionally omitted because it's not considered in our algorithm since the robot searches the wall going into right and follow the wall with it on the robot's right side. The *fleft* region is enough to detect obstacles on the robot's left side. It represents the decision making algorithm according to the sensor's input.

A. Improvements to the first version

Our robot and simulation models, and an embryonic version of the algorithm was based on a proposed project that can be found online [1]. This algorithm was a combination of simple navigation and obstacle avoidance algorithms. Our algorithm first version didn't consider the right region, having the other three as enough to follow the walls (Figure I). However, the robot didn't take into consideration the minimum distance to the walls. Therefore, after a wall corner turn, depending on the angle it could find the wall, it might get too close to the wall in order to follow it but then get stuck performing the next turn.

[1]at *www.theconstructsim.com*

TABLE I: Algorithm machine states table

Case	fleft	front	fright	right	State
1					0
2		X			1
3			X		4
4.1				X-a	2
4.2				X-b	3
5.1			X	X-a	2
5.2			X	X-b	3
6	X				0
7			X	X	1
8		X	X		1
9	X	X	X		1
10	X		X		0

[a]fleft: front-left; fright: front-right

TABLE II: Algorithm's actions by cases table

State	Action	Description
0	Find wall	go front and right
1	Turn left	go left
2	Follow wall	go front
3	Turn left soft - fix follow wall left	go front and soft left
4	Turn right soft - fix follow wall right	go front and soft right

To avoid this issue, a logic was created to ensure the robot considers both the maximum and minimum distances, corresponding to the cases 4.1, 4.2, 5.1 and 5.2. *X-a* and *X-b* are indicators to detected if the distance to the wall is inferior to the maximum distance for both cases, but greater than the minimum distance for *X-a* and inferior than the minimum distance for *X-b*. The *X-a* represents a wall following respecting the distance to the wall, so the robot will simply move forward, while the *X-b* means that the robot is too close to the wall and, so, it will turn softly to the left while moving forward since it still means it found the wall.

The case 4 was also added to the the first version of this algorithm since once we add a strategy to consider the region 'right', if the sensor detects the region 'fright' but doesn't detect the 'right', then it's probably too far from the wall. Therefore, it will turn right softly in order to be closer to the wall.

At the Figure 5 one can observe the State Diagram of the wall-following algorithm to better understand the implemented strategy.

IV. Experiments

As mentioned, the first version of our wall-following algorithm couldn't perform in some cases in which turning around a wall had a too tight angle, such as thin walls. The current version was similar to the this one except the previous didn't consider the cases 3, 4.1, 4.2 and 5.2, and, consequently, the states 3 and 4.

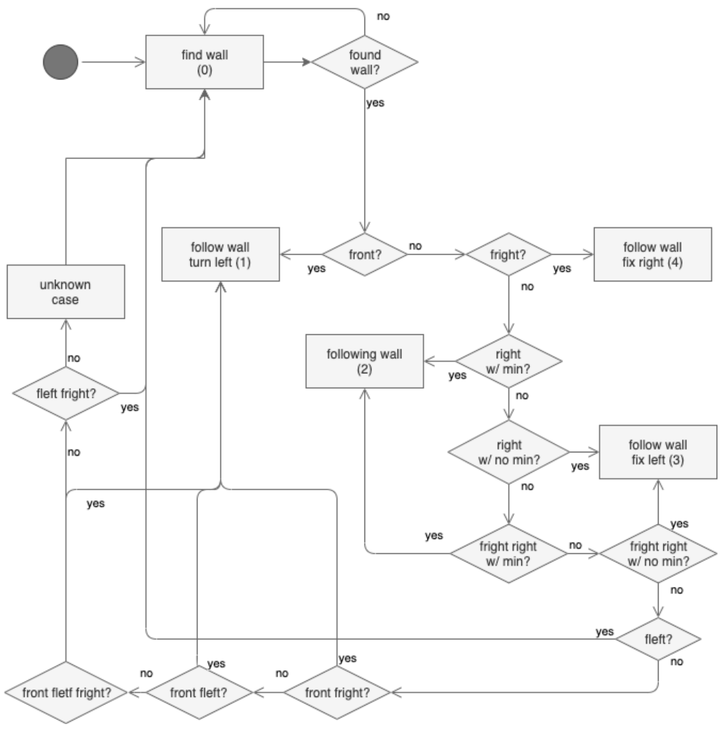

Fig. 5: The wall-following algorithm state diagram

After this improvement, the robot was not only able to turn around thin walls, but also was observed a much more linear path while following the walls respecting the maximum and minimum distances to them, even for the corners.

A. The worlds

We were able to create five worlds for the interest of experimenting our algorithm:

- The 'I' world, as the simpler one represented by a single wall;
- The 'L' world, to observe a 90 degrees angle turn;
- The 'T' world, to observe, also, a 90 degrees angle turn but with a smaller wall before turning again around the wall to observe if with less time and distance to recover the minimum and maximum distances, it could be able to turn around the wall;
- The 'W' world, to follow a non-constant angle and distance in each turn;
- The 'X' world, to follow a non-constant angle and distance in each turn, but with lesser wall length between turns.

B. Experiments Results and Discussion

Although the experiments were done in the five worlds, we represent in the Figures 6, 7 and 8 the three most significant cases in which our robot followed walls at the 'T' world, 'X' world and 'W' world, respectively.
It can be observed that, in the 'T' world, the trajectory shape is very close to the world. At the 'X' and 'W' worlds, the trajectories weren't so close to expected as the 'T' world. In the 'X' we see the trajectory more curved after an inner corner. This was due to the effort to recover the minimum distance to the wall. And in the 'W' world we can see the same curvature but, also, if the wall after an inner corner is long enough, it tries to recover the minimum distance and then it gets too far from the wall and tries to recover the maximum distance.

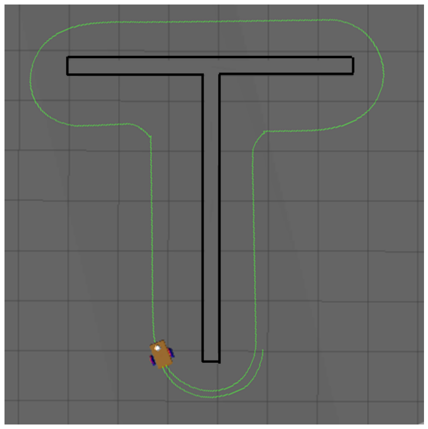

Fig. 6: The robot trajectory following a T-shaped wall

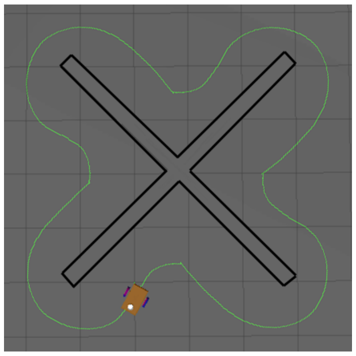

Fig. 7: The robot trajectory following a X-shaped wall

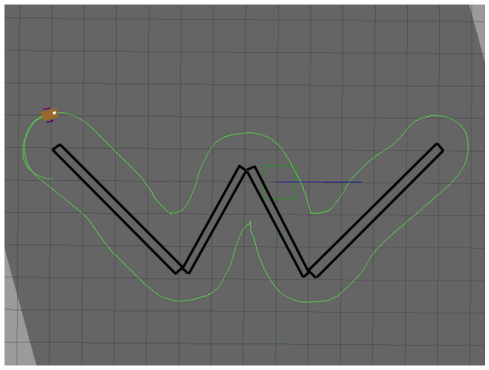

Fig. 8: The robot trajectory following a W-shaped wall

In the table III, we can observe the average time to perform a full round. The average was calculated from ten repetitions and a full round is considered to be from the position of the first time the robot finds a wall until the position in which the same point is reached again.

TABLE III: Experiments time until the robot performs a full round

Experiment	**Time (minutes)**
'I' world	01:13
'L' world	01:31
'T' world	01:48
'W' world	02:59
'X' world	02:16

The shorter the maximum and minimum distances to the wall, the closer the trajectory becomes to the wall shape regardless of the time consumed to perform a full round.
For time and trajectory performance purposes, the proposed algorithm can be improved in order to soften the recovery actions from the states 3 and 4. Our experiments suggests that a combination of different angular and linear velocities might have positive impact over the trajectory performance.

V. Conclusion

We introduced our robot model and its components and the laser interpretation from our algorithm. A detailed description of our algorithm was provided using a state diagram.
We also approached a first version of our wall-following algorithm, successfully improved issues regarding tight turns around wall in outer corners and presented it as our final version algorithm. And performed experiments in five worlds with 'I', 'L', 'T', 'W' and 'X' shapes, provided the results and discussed them.
Our algorithm fairly follows the proposed worlds walls in which we tried to represent a great variety of angles. In the tested environments, the wall-following trajectory is constant over the time and didn't show evidences of not being able to perform a full round.
Further improvements were suggested to stabilize the trajectory.

References

[1] Luis Sánchez Crespo, Anil Sánchez Mahtani, and Aaron Sánchez Martinez. Learning Ros for Robotics Programming. Birmingham, UK: Packt Publishing Limited, 2015.