A study on heuristics for LIDAR-based wall-following AGVs

Filipa Durão
Computer Engineering MSc
Faculty of Engineerying, UP
Porto, Portugal
up201606640@fe.up.pt

Miguel Duarte
Computer Engineering MSc
Faculty of Engineerying, UP
Porto, Portugal
up201606298@fe.up.pt

Rui Alves
Computer Engineering MSc
Faculty of Engineerying, UP
Porto, Portugal
up201606746@fe.up.pt

***Abstract*—Wall-following robots have been subject of study in robotics for decades. This paper goes into detail on the development and testing of a mobile wall-following robot. Even though there are multiple implementations of this kind of robot, this study focuses on applying the subsumption architecture to create wall-following behaviors. These use linear regression to infer the shape of the surrounding world and collision detection mechanisms to avoid hitting walls. The impact of changing the robot's speed and angular velocity on its behavior was studied, using maps with varying characteristics. An optimal parameterization, which both maximizes the robot's velocity and prevents it from getting stuck or crashing, was found. As such, it was concluded that a subsumption architecture fits the studied problem well.**

***Index Terms*—Wall-Following Robot, Reactive Robot, Mobile Robot, Subsumption Architecture, ROS, TurtleBot, AGV, LIDAR**

I. Introduction

Firstly, the current state of the art relative to moving robots and their following of paths will be studied. Secondly, the achieved implementation's details will be elaborated upon, along with thoroughly detailing its architecture. Thirdly, the experiments to be performed will be described, highlighting the conditions in which they occurred and how they were performed. Following this, the results of the performed experiments as described in the previous section will be explored, correlating the variation of certain values to their impact in the robot's run. Finally, the conclusions reached by this work are explained and potential future work to further this research will be mentioned.

II. State of the Art

The study of moving robots, that either follow strict pre-made paths or that react to the environment around them, is one of the core branches of robotics. Autonomously Guided Vehicles (AGVs) can range from factory plant robots that follow pre-placed rails on the floor to self-driving cars.

The main requirements for an autonomous mobile robot are stated in [1]. Some are coincidental to the ones needed in the robot developed for this project, such as:

- Having multiple sensors, not only to have accurate readings of the environment around the robot but also to make up for the flaws that some sensors can have in their readings
- Be robust, that is, being able to adapt and recover in the face of failure (i.e. a sensor failure)

In [2] it is described how the Reactive Paradigm in robotics works, revolutionizing the mobile robotics' functioning and how it helped to drastically decrease the cost of each robot by thousands of dollars.

On the topic of AGVs, many different applications, such as mining and mine-extraction vehicles, have been the target of study throughout the last decades. For example, in [3], the use of reactive mobile robots to explore mines more safely was studied. This study showed that fully reactive, wall-following robots were a success, only needing human intervention when reaching bifurcation (although it is worth mentioning that in more recent years many other different techniques have been studied, such as the use of machine learning, as discussed in [4]).

Regarding the usage of mobile robots, **Turtlebot** has become famous for its simplicity. In [5], this robot was used to design a personal assistant mobile robot using the operating system ROS. This project was validated through an experiment where a Turtlebot prototype was deployed on the campus of the university performing the study.

Subsumption architecture is a reactive robotic architecture heavily associated with behaviour-based robotics.

Concerning reactive robots' architectures, the subsumption architecture has been widely used in the past decades. In [6], the use of this architecture on the development of robots was studied, concluding that it results in more flexibility than previously existing reactive architectures, since it allows the incremental addition of new behaviours to a robot. However, it is worth mentioning that improvements and alternatives to this architecture have been studied in parallel, as discussed in [7], which explores the decomposition of a robot's control unit using a set of different heuristics.

Finally, the use of linear regression-based methods for quick collision detection is explored in [8]. Through analysis of collision points at a given distance and angle, it is possible to map detected collision structures to a 2D space centered on a robot. The shape of those collision structures can be inferred by linear regression application, improving a robot's perception of the world.

III. Implementation and Architecture

The robot was developed using **ROS - Robot Operating System** (version *Noetic*) [9] and the simulations took place using the **Gazebo** [10] simulator. The source code was written using **Python** (using the **rospy** [11] ROS package).

The robot itself consists of a **Turtlebot3 Burger** [12] model, featuring a LIDAR which is used as a sensor to measure the distance to nearby objects. The robot has a round shape with about 0.2 m of diameter and features actuator motors to move the wheels using differential locomotion.

The robot's architecture follows the subsumption architectural style, as shown in figure 1. The robot's behaviour, which is to follow a wall or wander until a wall is found, is divided into sub-behaviours:

- The lowest hierarchical layer of subsumption holds the the "Avoid collision" behaviour, in which the robot stops and turns towards a direction without obstacles in its path
- Then, the next layer holds the "Following Wall" behaviour, in which the robot will follow a wall it finds at a certain distance.
- After that, the next layer holds the "Finding wall" behaviour. In this behaviour, the robot has spotted a wall at a far distance and will walk towards it.
- Finally, the last hierarchical subsumption layer holds the the "Wandering" behaviour, in which the robot wanders through the world with random directions until a wall is possibly found

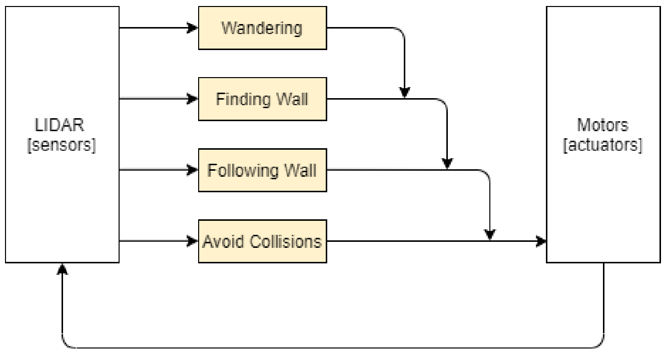

Fig. 1. Subsumption Architecture

The robot consist of a **Reactive Robot** that continuously execute the following reactive loop:

```
sensor_data = getLidarValues()
speed, ang_velocity =
    computeAction(sensor_data)
move(speed, ang_velocity)
```

In order to devise the proper heuristics, parameters and algorithmic details, the **Gazebo world** in figure 2 was conceived.

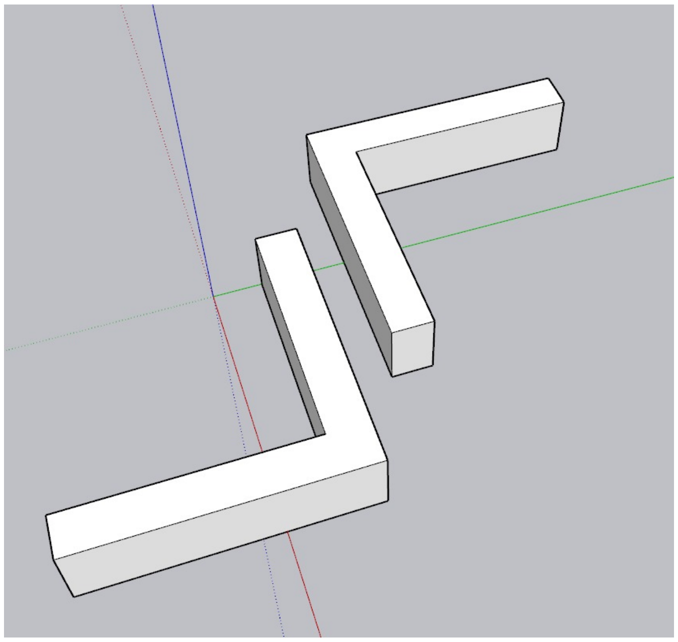

Fig. 2. Double L-shaped Map

It's worth noting that this map features most geometrical wall possibilities, from straight walls to both concave and convex curves.

A. Initial Approach

The first approach was a trivial, *ad hoc* heuristic that consisted of the following steps:

```
WALL_DISTANCE_THRESHOLD = X
if (lidarHasNoReadings()):
    return
d = getClosestDistanceReading()
if (d >= WALL_DISTANCE_THRESHOLD):
    return
if (closestReadingOnLeft()):
    turnRight()
else:
    turnLeft()
```

This behaviour can be achieved by using just two angle sensors, one at the left of the robot and one at the right (in this case, at 90° and at −90° relative to the robot's front).

Although simple, this naive approach works when following straight walls. However, it misbehaves sharp curves or in any kind of U-shaped structures.

B. Linear Regression Approach

A more robust approach should use an orientation line instead of a single point in space in order to decide the robot's orientation. Since the LIDAR sensor features a large number of distance beams, it is possible to take into account the N closest distance measures to the robot and map those distances to a 2D space (with origin on the robot's center) using simple trigonometry:

```
foreach pair<distance, angle>:
    x = distance * cos(angle)
    y = distance * sin(angle)
```

Then, using those points in the Robot-centered 2D space, it is possible to compute the linear regression for those points. The guiding vector of the computed linear regression may then be used as an indicator of the optimal robot's orientation. Thus, by pivoting towards the same orientation as the linear regression line, the robot traverses in a path that is parallel to the wall.

Although this method results in correcting the robot's orientation, it does not directly force it to keep a given distance from the wall. This is achieved by combining both the aforementioned approach and the one described in the previous subsection.

It is worth noting that the linear regression lines fits both straight walls as concave and convex sharp curves, as showcased in figures 3, 4 and 5.

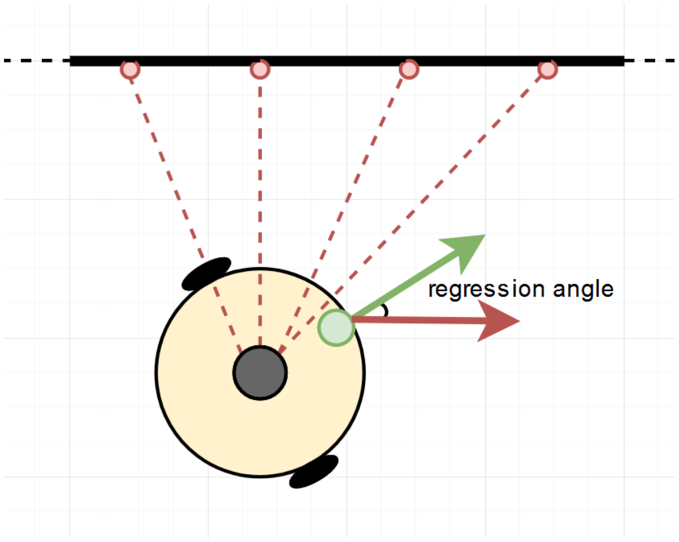

Fig. 3. Straight wall Linear Regression

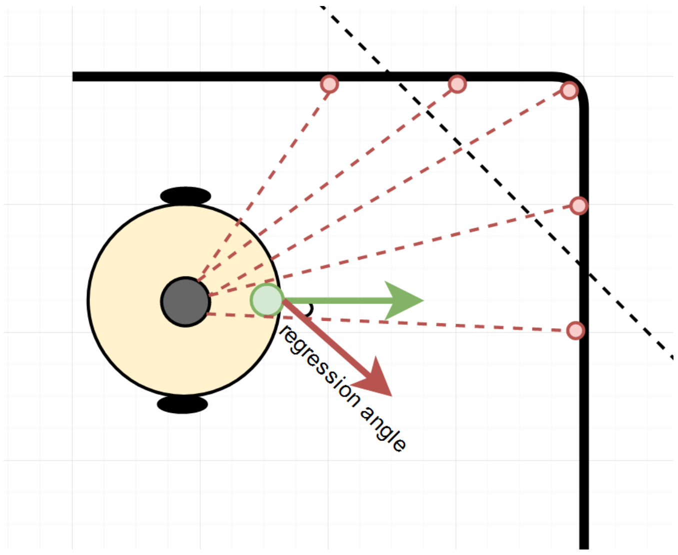

Fig. 4. Concave sharp curve Linear Regression

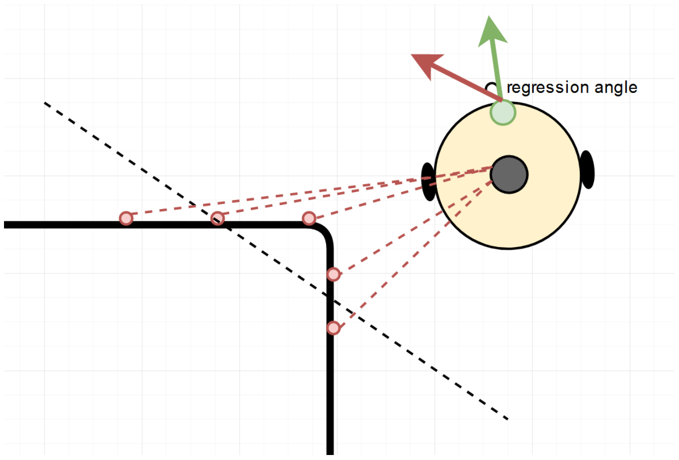

Fig. 5. Convex sharp curve Linear Regression

C. *Collision Detection Approach*

Although the two previous approaches result in the robot smoothly following a wall at a fixed distance, it is missing control behaviour when a collision is imminent.

Thus, a control mechanism for frontal collision detection was implemented:

```
COLLISION_THRESHOLD = X
f = getFrontalDistance()
if (f < COLLISION_THRESHOLD):
    route = findFreeRoute()
    turnRobot(route)
```

The *free route* consists of the direction in which the robot is allowed to make progress, that is, that features a wall that can be followed. This route is computed by using a linear regression method as the one described in subsection III-B.

D. Other Improvements

1) Wandering Behaviour: When the robot is not able to detect a wall to follow, there are different behaviours it could take, such as simply holding still or moving in a straight line. However, these alternatives are unlikely to result in success in completing the robot's goal (that is, following a wall).

Thus, when a wall is not found, the robot traverses the map at maximum velocity, turning it random directions. Whenever a wall is detected at a far distance, the robot attempts to reach the wall in order to fulfil its goal:

```
if (findWall() == null):
    move(MAX_SPEED, getRandomAngVelocity())
else:
    if (wallIsFar()):
        approachWall()
    else:
        followWall()
```

It is worth mentioning that this wandering + moving towards a faraway wall also consist of a failure recovery mechanism, that is, if the wall-following behaviour fails for whichever reason and causes the robot's path to highly deviate from the wall-following path, the robot will pivot towards the wall to resume its correct behaviour.

2) Motion-smoothing factors: Finally, in order to improve the robot's speed while traversing a given map, different motion factors were taken into account:

- When the robot is following a straight line without any curves, an acceleration factor is added
- When the robot is following a smooth curve (with a low turning angle), an angular acceleration factor is added
- When the robot is faced with a tight corner (U-shaped or similar), a speed breaking factor is added

The aforementioned improvements allow the enhancement of the robot's movement behaviour while respecting safety measurements and distances to avoid collision.

IV. Experiments

In order to analyse the effectiveness and efficiency of the implemented solution, several experiments were performed. They aimed to evaluate the robot's behaviour in several aspects.

Firstly, a qualitative approach to see how the robot performs in several different scenarios was created: The robot would be launched in several different situations with different goals. The most relevant behaviours in this scenario are the ability to wander and find a wall which it will follow; and the ability to perform wall-following cycles without any errors in its behaviour (by errors several aspects can be considered, from simply being slightly off-course to following an incorrect path or bumping into the wall).

Additionally, its speed in completing a given course will be studied. This is a quantitative approach given that minimising the *lap time* is desirable, but errors in the course should be avoided.

In order to study the robot's behaviour under different conditions (straight walls, curved and convex walls, different types of angle turns, *et cetera*), the map in figure 6 was conceived and taken into account.

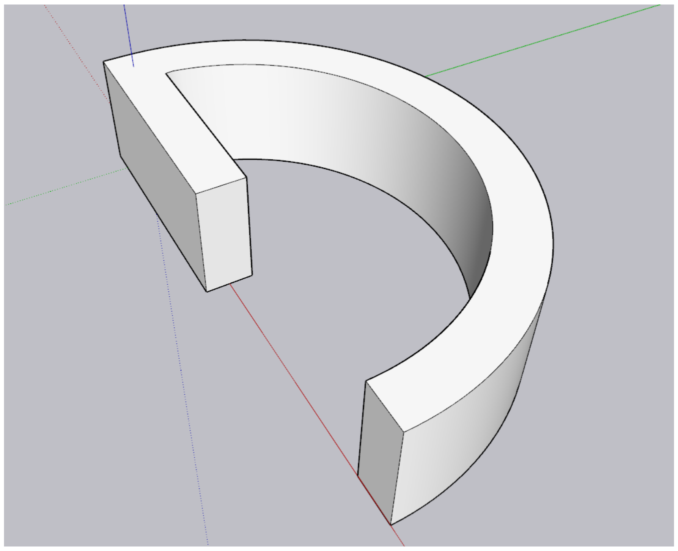

Fig. 6. Studied Map

Regarding the developed experiments:

- The experiments took place using the **Gazebo** simulator and **ROS Noetic**, making use of a **TurtleBot3** robot;
- The starting orientation pose of the robot (traversing the map in the clockwise versus counter-clockwise direction) was random;
- A run is considered *successful* if the robot is able to follow the wall through the entire map without deviating from the wall-following path for more than 3 seconds. The successful run's traversed path may be observer in figure 7 (generated using **Rviz** [13], a 3D visualization tool for ROS).
- A run is considered *unsuccessful* if the robot becomes stuck in a wall or if it deviates from the wall-following path for more than 3 seconds, although the robot features mechanisms to recover from either case, continuing to follow a wall as expected;

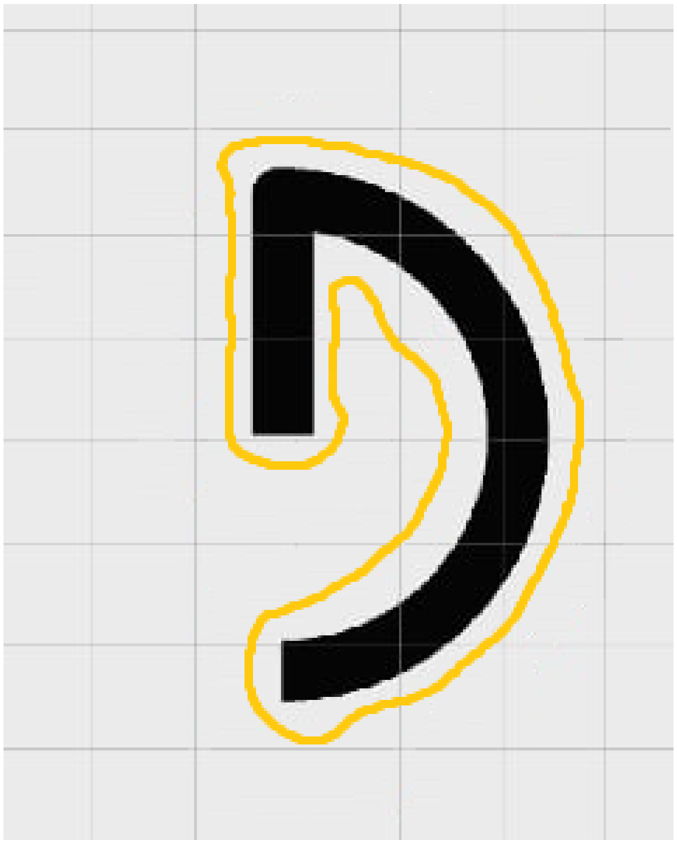

Fig. 7. Successful run traversed path

Several parameters will be taken into account in order to provide a more complete overview of the robot's behaviour:

- Quantitative measures:
  1) Map's wall thickness
     - Ranging from $0.2\ m$ to $0.6\ m$
     - Incremental steps of $0.1\ m$
  2) Robot's speed
     - Ranging from $0.2\ m/s$ to $0.32\ m/s$
     - Incremental steps of $0.04\ m/s$
  3) Robot's angular velocity
     - Ranging from $\pi/3\ rad/s$ to $2\pi/3\ rad/s$
     - Incremental steps of $\pi/9\ rad/s$
- Qualitative measures:
  1) Percentage of *successful* runs

V. Results and Discussion

The presented results consist of the average of **15 trial runs for each scenario**, which results in a total of **600 trials** (that is, **40** unique testing scenarios).

A. Impact of Robot Speed and Wall Thickness on Lap Time and Success Rate

The robot's increasing speed resulted in lower lap times regardless of the wall thickness, as visible in figure 8.

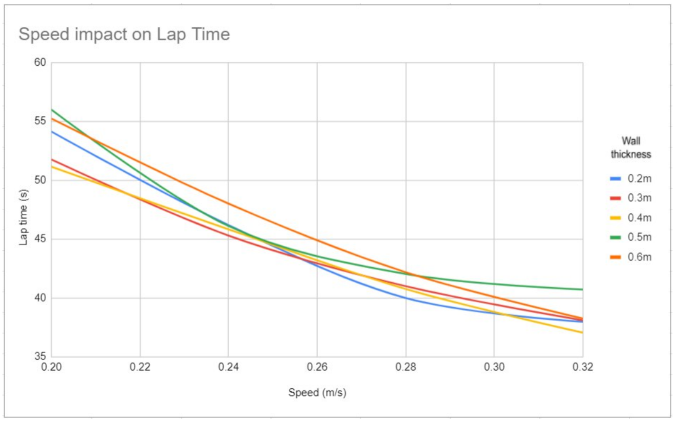

Fig. 8. Speed Impact on Lap Time

However, as the robot speed increases, the more likely it is for the sensors to be prone to errors. Moreover, with higher speed / acceleration the robot tends to have less time to react to adversities (such as imminent collision), which results in non-optional lap success rates, as visible in figure 9.

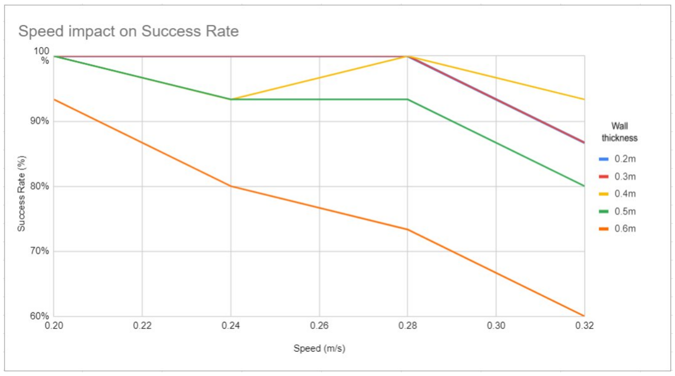

Fig. 9. Speed Impact on Success Rate

Thus, the experiments show that even though speed reduces lap time, the balance between a fast speed while maintaining a peak performance success rate is achieved at a speed of $0.28\ m/s$.

Additionally, it is possible to conclude that errors caused by unmanageable speed values have a bigger impact on the lap success rates of maps with higher wall thickness.

B. Impact of Angular Velocity and Wall Thickness on Lap Time and Success Rate

On the one hand, the robot's increase in angular velocity resulted in lower lap times regardless of the wall thickness, as visible in figure 10. However, this metric has a much smaller impact on lap time than the variation of the robot's speed.

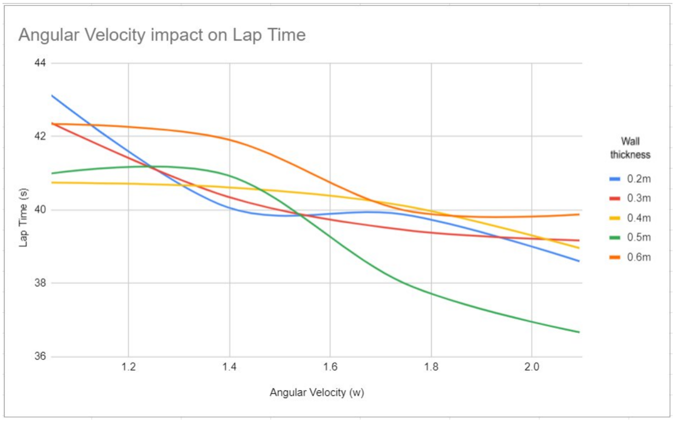

Fig. 10. Angular Velocity Impact on Lap Time

On the other hand, the failure rate increases proportionally to the robot's angular velocity, as visible in figure 11. The increasing of angular velocity has a higher impact on the robot's failure rate than speed does, given that the very quick direction adjustments may often cause unpredictable / unwanted behaviour.

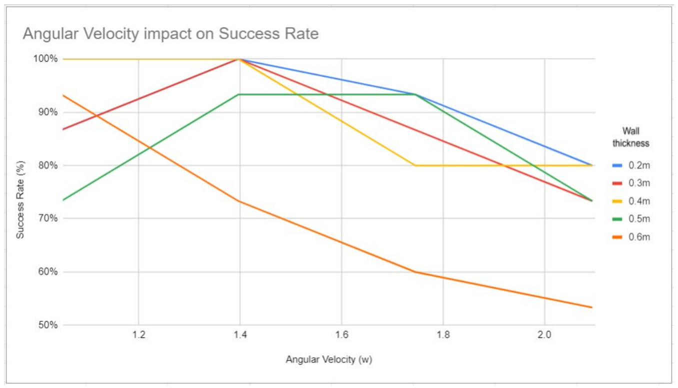

Fig. 11. Angular Velocity Impact on Success Rate

Thus, the experiments show that although the angular velocity metric does not have a very meaningful impact on lap time, higher values of this metric are likely to result in low success rates. The angular velocity value that achieved peak performance for most wall thickness values was $4\pi/9\ rad/s$

It is also possible to conclude that high values of angular velocity resulted in errors regardless of the wall thickness, although that effect has a higher impact on maps with higher wall thickness.

C. Stuck Robot

As discussed in the previous two subsections, it is possible for the robot to become stuck when high acceleration values are being used.

When it comes to maps with a low wall thickness (that is, less or equal to $0.4\ m$), the robot is unlikely to get stuck even when subject to higher speed and angular velocity values.

However, as both the wall thickness and angular velocity values increase (that is, for wall thickness values greater than $0.4\ m$ and angular velocity values greater than $4\pi/9\ rad/s$), the robot is more prone to getting stuck. This happens due to the fact that high values of wall thickness result in sharper curves and less space in the inner region of the map. In this region, as speed and angular velocity increase, the more likely the robot is to collide with a wall and get stuck, even though the mechanism described in subsection III-C is highly successful in preventing this anomaly.

It is worth mentioning that the implemented solution features mechanisms to cope with robot failure (both path deviation and getting stuck), as detailed in subsection III-D.

VI. Conclusions and Future Work

A reactive robot with a subsumption-based architecture is able to achieve satisfactory results for simple tasks such as following a wall.

The implemented solution is capable of following walls of different shapes (from straight walls to sharp curves) due to the usage of linear regression-based methods for inferring the surrounding world's shape. The collision detection behavior results in the robot avoiding getting stuck in most situations and the wandering behaviour allows for both recovering from path deviation as well as finding wall-following paths at far distances.

The developed experiments - which took into account both quantitative (such as the robot's speed, angular velocity and map's wall thickness) and qualitative (completing a lap without deviating from the correct path for a long period of time) factors - showed that a good compromise (in maps with wall thickness varying from $0.2\ m$ to $0.6\ m$) between quick lap times (of about $40\ s$) and low failure rates (below 3%) was achieved when speed averaged at $0.28\ m/s$ and angular velocity averaged at $4\pi/9\ rad/s$.

Finally, future work should include the study of these metrics in maps with different shapes (possibly randomly generated).

Furthermore, including other methods to infer the surrounding world shape (such as the use of quadratic and other types of regression) should be taken into account, in order to achieve a more realistic perception of the robot's surroundings.

References

[1] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.
[2] R. Murphy, *Robotic Paradigms*. MIT Press, 2000, p. 2–10.
[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] S. Madi and A. Baba-Ali, *Classification Techniques for Wall-Following Robot Navigation: A Comparative Study*, 01 2019, pp. 98–107.
[5] A. Koubâa, M. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, and A. Mahmoud, "Turtlebot at office: A service-oriented software architecture for personal assistant robots using ros," in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2016, pp. 270–276.
[6] Rosenblatt and Payton, "A fine-grained alternative to the subsumption architecture for mobile robot control," in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 317–323 vol.2.

[7] R. Peter Bonasso, R. James Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack, "Experiences with an architecture for intelligent, reactive agents," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, no. 2-3, pp. 237–256, 1997.

[8] S. Brandao, M. Veloso, and J. P. Costeira, "Fast object detection by regression in robot soccer," in *Robot Soccer World Cup*. Springer, 2011, pp. 550–561.

[9] "Ros noetic." [Online]. Available: https://wiki.ros.org/noetic

[10] Osrf, "Gazebosim." [Online]. Available: http://gazebosim.org/

[11] "Rospy." [Online]. Available: https://wiki.ros.org/rospy

[12] "What is a turtlebot?" [Online]. Available: https://www.turtlebot.com/

[13] "Rviz." [Online]. Available: https://wiki.ros.org/rviz