# Wall Following Reactive Robot

Luís Costa, João Loureiro e João Sousa

***Abstract*— Advances in robotics motivate the need for stable and reliable autonomous movement. Using ROS and the STDR simulator, a wall following robot was designed and implemented, using a laser scanner for wall detection. Reported experiments include following along a square section indefinitely, a D shaped section and stopping the at the center, a circle shaped section and a section made up of an assortment of rectangles, in order to test different aspects of its behavior.**

KEYWORDS

Reactive Robot, ROS, STDR Simulator

## I. INTRODUCTION

**Reactive systems** are a relatively recent development in robotics that has redirected artificial intelligence research. This new approach emerged from the dissatisfaction with existing methods for producing intelligent robots response and a growing awareness of the importance of looking at biological systems as a basis for constructing intelligent behavior. Reactive robots are also know as **behavior-based robots** since they are instructed to perform through the activation of a collection of low-level primitive behaviors. Complex behavior builds up on those simple tasks through the interaction of the environment in which the robot finds itself. This project aims to simulate a reactive robot with a simple *wall-following* behavior. Using **ROS** framework with ***STDR Simultor*** package, a procedure was implemented and tested. However, reactive robots often depend on sensors to collect information and avoid obstacles, the detection was done using a laser sensor that can determine the distance from the right and left sensor to the wall that was being followed. Another advantage of using laser sensors is the reduction of the **2D** problem, since the robot only has to follow one wall chosen from the info gathered using both sensors.

## II. DEVELOPMENT

### A. Goal and Vision

The goal of the project is to develop a simple reactive robot able to navigate in a *D* shaped map. It should spawn in a randomly assigned position of the map and follow a straight line until it finds a wall. By then, the robot should be able to continuously follow it by adjusting its trajectory according to the wall. With that, we intend to, using a simple behavior based architecture, develop a robot that can actuate based on the information gathered from its laser sensors.

### B. Technical Architecture

The application was developed under **ROS** (Robot Operative System), using a fairly common package named ***stdr_simulator*** which implements a distributed server-client based architecture backed by a **GUI** for visualization purposes. The *stdr_simulator* package includes several sub packages from which we name a few relevant for our development.

- ***stdr_gui***: An interface for visualization purposes
- ***stdr_launchers***: Files loader, such as robots and maps
- ***stdr_resources***: Provides descriptor for robots and maps

The robot logic and movement was developed using **C++**, using a subsumption algorithm explained on section *F*.

### C. Map Creation

The map of the simulation was built using a *D* shaped *png* image extracted online. It was later resized to a higher resolution (for optimized results) and edited according to our preference. Blacker pixels are considered inaccessible locations whereas whiter ones are areas where the robot may move. In addition, a `yaml` file was created with the image description and specifications so that the *stdr_simulator* could load it as a map.

We created 3 *D* shaped maps for 3 different testing scenarios. The *extra* D-map allows the robot to freely move inside the *D* contour, the *internal* D-map allows the robot to move freely inside the D and the *external* D-map allows the robot to move freely outside the D.

(a) Extra D-map

(b) Internal D-map

(c) Internal D-map

### D. Robot Design

The reactive robot was modeled as a simple kinematic unicycle with state:

*This work was not supported by any organization

[1]H. Kwakernaak is with Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7500 AE Enschede, The Netherlands `h.kwakernaak at papercept.net`

[2]P. Misra is with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA `p.misra at ieee.org`

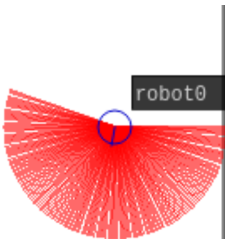
Fig. 2: Robot state

$$q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

where (x,y) is the Cartesian location of the robot's center within the world model and $\theta$ is the robot's yaw angle as measured from the wall, or the world model's X-axis.

The robot features 2 main characteristics defined as the **sensors** and the **controller**. The **sensor** refers to all methods required to pull data from the robot's environment, interpret a wall model, and develop a state estimate q of the robot's position relative to the wall. The **controller** dictates the movement of the robot based on the given wall model and an the robot's forward speed.

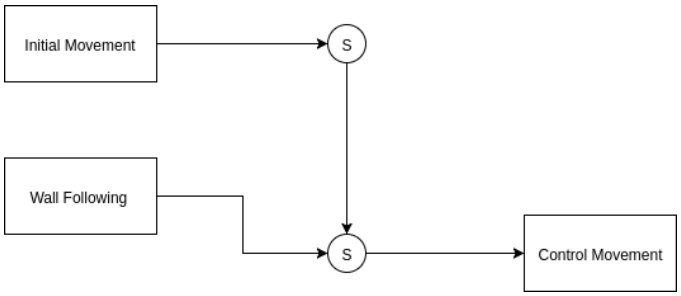
Fig. 3: Robot representation




### E. Architecture

The robot has been implemented using a **subsumption** architecture. Behavior-based robots are built up out of a set of independent, simple behaviors. Behaviors are defined by what triggers them (sensor reading) and the action taken. In this case, the sensors measure the distance between the robot and the wall(s). If the measurement varies inappropriately, then the angle/direction the robot is facing must change, accordingly to the goal (follow the wall).

Fig. 4: Behavior diagram

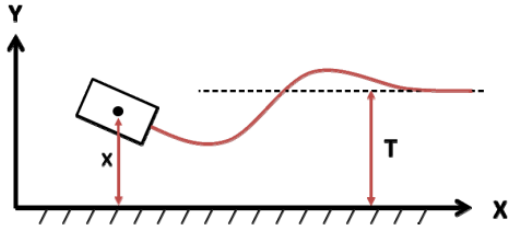



### F. Algorithm

Wall following is a simple maze algorithm that most of us have learned when we were a child. The algorithm defends that it is possible to solve a maze by following a wall indefinitely. This will always work as long as there is an entrance and an exit in the same plane or board. Imagine that we decide to take left and start to *wall follow*. The algorithm can be described as something like this:

- Go straight, start the movement
- When you run into a wall, turn right (remember we are using the left-hand approach)
- Stop when you reached the end.

Fig. 5: World model described by a Cartesian coordinate system




For a better understanding, there is a picture that illustrates the world model. The world model represented on the figure above assumes that there is always a wall to follow within the view of the laser scanner. The goal of the model represented is to follow an imaginary line at a distance T from the wall. The linear velocity is constant and the angular velocity is controlled by the following equation:

$$\omega = (-k * (sin(\alpha) - (x - T))) * v$$

where $v$ is the linear velocity of $0.4m/s$, $k$ is an intuitive variable based on the number of experimental observations. $k$ was set to 20. $\alpha$ is the angle between the front of the robot and the imaginary line that is being followed, $x$ is the distance between the robot and the wall and $T$ is the ideal distance. Each ray takes into account the nearest point of the wall within it's range. The angle between the ray and the robot can be calculated since we know the angle increment of each laser ray. We use this angle to calculate $\alpha$ because our imaginary $T$ runs in parallel with the wall. So, our best scenario is $\alpha = 0^o$ and our worst case scenario is $\alpha = 90^o$.

Fig. 6: Piece of code that represents the world model described above

```
if(min_distance <= scan.range_max) // within range of sensors
        {
            cmd.linear.x = 0.4;
            cmd.angular.z = (-3 * (sin(dTr(alpha)) - (min_distance - 1.46))) *
cmd.linear.x;
        else //initial movement
        {
            cmd.linear.x = 0.3;
            cmd.angular.z = 1.3;
        }

        cmd_vel_pub.publish(cmd);
    }
```

## III. Results

In this section is possible to see the stability of the algorithm regarding the distance to the wall and the $\alpha$ angle.

During the development, we noticed that the robot would only stabilize at 1.54m so we reduced to 1.46, our new and forged ideal location. Observing the following graphs is possible to see that the distance is stable in all the cases/tests. The same does not happen with the angle and the instability becomes higher when the wall is a curve. We'll discuss more about this topic on **Limitations** Section.

Fig. 7: Distance Stability

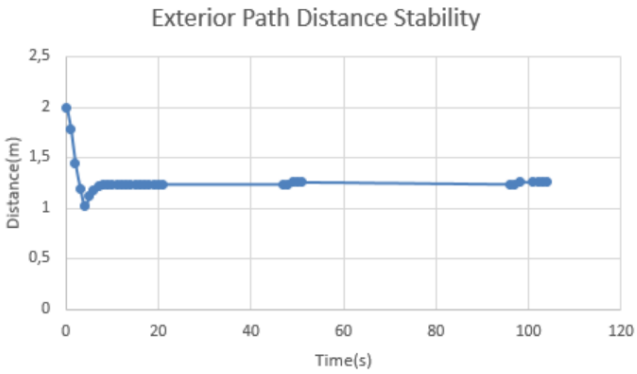
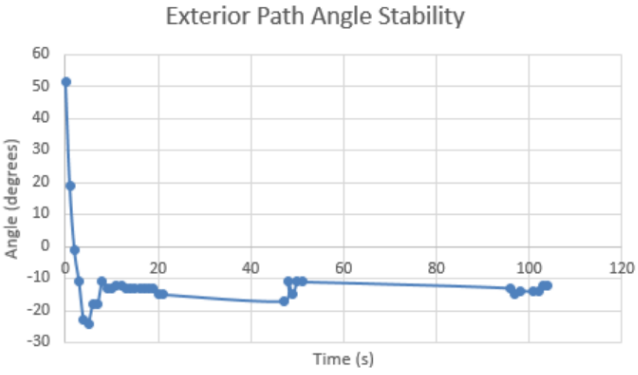



Fig. 8: Angle stability. Variations during time




## IV. Limitations

After watching the results it's time to understand what could've been done better. We can see that the algorithm shows great instability around interior corners, similar to the initial adjustment. When approaching to the corner there will be a sudden shift, since the minimum distance to the wall will change from one of the sides to the front which will make the "turn" angle higher. This effect is usually called ***Wall Disappearance*** and there are some ways to solve it, using **Wall Loss Compensation** algorithms. We could start by calculating the wall curvature for a more stable and smooth movement.

## V. Conclusion

The exterior D Path shows that the robot is stable after the first adjustment and there is some angle stability when the robot reaches the corners. In the interior D Path we can see distance stability on the corners as well but the angle stability is highly unstable. This happens due to the constant need to adjust to the almost circular shape. The Double D path seems to be stabilized with minimum adjusts on the angle stability due to the two walls that are scanned by the robot. Since the sensors range can reach both of the walls, it is needed constant adaptation from the robot to stay near the wall and follow the path. As improvements it would be desirable to implement a starting random movement to better test the approach towards an obstacle rather than having to alter the map itself or the STDR launch files. Even though turning around inner corners was not one of the objectives of the project, it would be desirable to increase that stability as well.

Fig. 9: Distance stability on Exterior Path

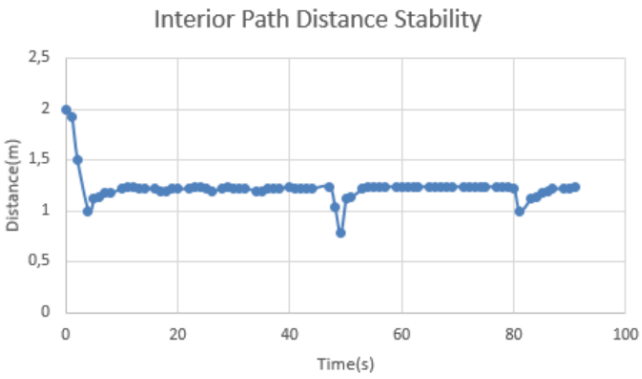
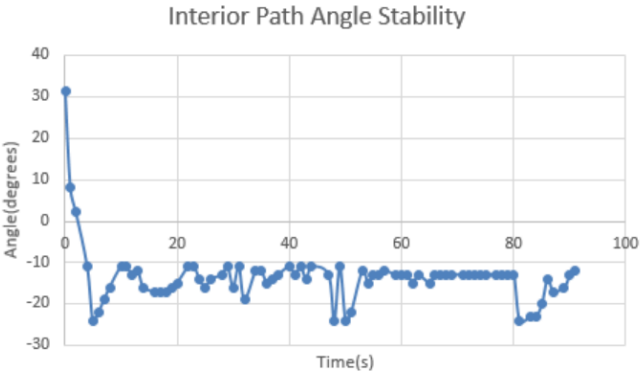



Fig. 10: Angle stability on Exterior Path




## VI. References

## References

[1] Karl Bayer, "Wall Following for Autonomous Navigation", 2012, available at: https://www.seas.upenn.edu/sunfest/docs/papers/12-bayer.pdf

[2] Paul Reiners, "Robots, mazes, and subsumption architecture", December 2007, available at: https://www.ibm.com/developerworks/library/j-robots/

Fig. 11: Angle stability on Double D Path

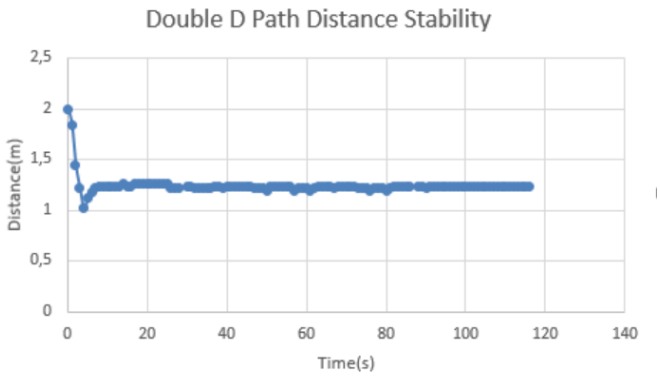



Fig. 12: Angle stability on double D path