Wall Following Robot Using ROS and Gazebo

Maria Teresa Chaves
up201306842@fe.up.pt

Pedro Santos
up201101789@fe.up.pt

***Abstract*— This article presents the design of an autonomous robot simulation as a basic development of a wheeled mobile reactive robot.**

I. INTRODUCTION

A reactive robot has multiple instances of Sense-Act couplings, which are current processes, called behaviors, that takes the local sensing data and compute the best action to take independently of what the other processes are doing. [1]

An essential tool on a robot development is a robot simulation. Gazebo is a simulator that makes possible to rapidly test algorithms, robots, perform regression testing, and train AI system using realistic scenarios. [2]

On the simulation of our wall following robot we used TurtleBot model that is a personal robot kit. This kit consists of a mobile base, 2D/3D distance sensor, laptop computer or SBC(Single Board Computer), and the TurtleBot mounting hardware kit. [3]

This project aims to simulate a reactive robot that wanders at random on a map until it finds a wall, after that the robot follows the wall forever.

The robot runs on three possible maps, a D shaped map in which the robot wanders at random inside the wall, a D shaped map in which the robot wanders at random outside the wall and a map in which the robot wanders at random between a small D and a large D.

II. ARCHITECTURE

A. TurtleBot Characteristics

Fig. 1. TurtleBot in Gazebo

The laser range sensor of Turtlebot has:

- 640 number of points
- Increment in angles that is $0.00163669rad$
- Minimum angle that is $-0.521568rad$
- Maximum angle $0.524276rad$
- Minimum range (distance) that is $0.45m$
- Maximum range (distance) that is $10m$

B. The Algorithm

Our first task was to know where the wall was. For that we had used the laser range finder in order to get the minimal distance values. This way we had the distance to the closest wall and the angle relative to the robot.

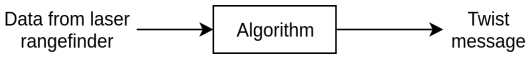

Fig. 2. Algorithm input and output

The increment in angles scanned by the laser is $0.00163669rad$. The minimum angle scanned by the laser is -0.521568 rad. The maximum angle scanned by the laser is 0.524276 rad The minimum range (distance) the laser can perceive is 0.45 m The maximum range (distance) the laser can perceive is 10m

After that we created a control system for the wall-following behavior. The robot has linear and angular velocity values for each given moment. To determine in which direction the robot should turn at each instant, so that it remains parallel to, and a certain distance from, the wall it is following we used the PID controller algorithm. For the maximum linear velocity at which the robot may move at each moment we used normalization.

C. PID Controller Algorithm

To keep the same distance and angle to the wall, we used PID controller algorithm, which changes the angular velocity to maintain these values.

This algorithm has three values: the proportional value; the integral value and the derivative value. The proportional value determines the reaction to the current error. The integral value determines the reaction based on the sum of recent errors. Finally the derivative value determines the reaction based on the rate at which the error has been changing. [7]

The proportional term, sometimes called gain, makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p called proportional gain. The proportional term is given by:

$$P_{out} = K_p e(t) \tag{1}$$

Where P_{out} is the proportional term of output, K_p is the proportional gain, a tuning parameter, e is the error and t is the time or instantaneous time (present).

The contribution of the integral term, sometimes called reset, is proportional to both the magnitude of the error and the duration of the error. Summing the instantaneous error

over time (integrating the error) gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain and added to the controller output. The magnitude of the contribution of the integral term to the overall control action is determined by the integral gain, K_i. The integral term is given by:

$$I_{out} = K_i \int_0^t e(t)dt \tag{2}$$

Where I_{out} is the integral term of output, K_i is the integral gain, an tuning parameter, e is the error, t is the time or instantaneous time (present).

The rate of change of the process error is calculated by determining the slope of the error over time (i.e. first derivative with respect to time) and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain K_d. The derivative term is given by:

$$D_{out} = K_d \frac{d}{dt} e(t) \tag{3}$$

Where D_{out} is the derivative term of output, K_d is the derivative gain, a tuning parameter, e is the error and t is the time or instantaneous time (present).

The final form of the PID algorithm is given by:

$$u_PID(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t) \tag{4}$$

D. Normalization

To determine the linear velocity that the robot should move at each moment we used the laser range distances.

Using the limits of the laser range distances, maximum value and minimum value, we determine the percentage of the maximum linear velocity it should take, using the normalization of the error (absolute value) with the limit between 0 and 1.5 corresponding to the limit 0% and 100%.

To calculate the percentage of the maximum velocity that the robot should move at each moment we used the expression:

$$u_{Norm}(t) = \frac{e(t) - min(x)}{max(x) - min(x)} \tag{5}$$

Where $u_{Norm}(t)$ is the velocity percentage output, $min(x)$ is the minimum value of the interval that will be the minimum error value (0) and $max(x)$ is the maximum value of the interval that will be the maximum error value (1.5).

E. Angular Velocity

The angular velocity $[v_a(t)]$ is given by the PID controller output $[u_{PID}(t)]$ and the maximum velocity of the robot $[MAX\_VEL]$:

$$v_a(t) = u_{PID}(t) * MAX\_VEL \tag{6}$$

F. Linear Velocity

The linear velocity $[v_l(t)]$ is given by the absolute error normalization output $[u_{Norm}(t)]$ and the maximum velocity of the robot $[MAX\_VEL]$:

$$v_l(t) = u_{Norm}(t) * MAX\_VEL \tag{7}$$

When the center of the robot is too close to the wall a new normalization is done, with the minimum and maximum limits between $DIST\_WALL$ and 0 that will give a negative linear velocity to make the robot move away from the wall.

III. RESULTS

The simulation of the robot that follows a wall runs into three different maps. In each map the reactive robot wanders at random until it finds a wall, after that the robot follows the wall forever.

A. Map "D" Robot Inside

Map "D" robot wanders at random until find wall, then follows wall forever (inside wall).

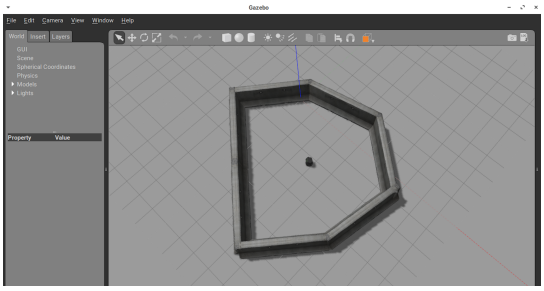

Fig. 3. Map "D" Robot Inside

B. Map "D" Robot Outside

Map "D" robot wanders at random until find wall, then follows wall forever (outside wall).

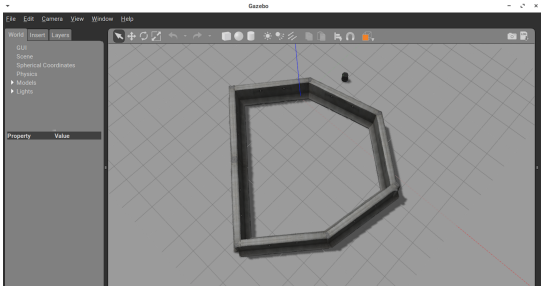

Fig. 4. Map "D" Robot Outside

C. Map "DD" Robot Between

For extra merit, we created randomly a robot in between a small D and a large D arena.

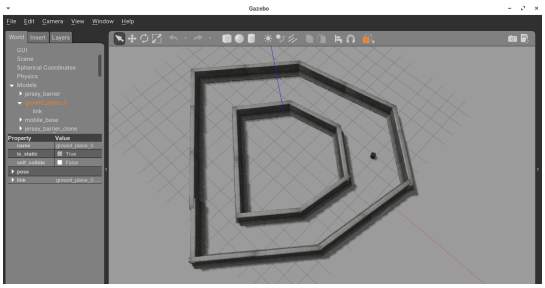

Fig. 5. Map ”DD” Robot Between

IV. CONCLUSIONS

The simulation of the reactive robot that follows a wall has been implemented. The sensor data are used to measure and obtain the distance and orientation of the robot. Then, these data are utilized in the PID controller algorithm and for the normalization, in order to control the angular velocity and the linear velocity, respectively. Our robot runs successfully into three different maps, the ”D” shaped map with the robot inside, the ”D” shaped map with the robot outside and the ”DD” shaped map with the robot between the two Ds.

REFERENCES

[1] Robotic Paradigm, Wikipedia, https://en.wikipedia.org/wiki/Robotic_paradigm

[2] Gazebo Simulator, http://gazebosim.org/

[3] TurtleBot, http://www.turtlebot.com/

[4] Obstacle Avoidance Using TurtleBot, https://www.mathworks.com/help/robotics/examples/obstacle-avoidance-using-turtlebot.html

[5] TurtleBot PID, https://w3.cs.jmu.edu/spragunr/CS354_F16/labs/pid_lab/pid_lab.shtml

[6] An Autonomous Wall Following Robot, http://academic.csuohio.edu/simond/courses/MehtaReport.pdf

[7] PID Wall Follower, https://www.scribd.com/document/44206872/PID-Wall-Follower

[8] Intelligent Systems: Technology and Applications, https://books.google.pt/books?id=CjkS_KjbvgsC

[9] Toward Dynamical Sensor Management for Reactive Wall-following, http://repository.upenn.edu/cgi/viewcontent.cgi?article=1690&context=ese_papers

[10] Wall Following for Autonomous Navigation, https://www.seas.upenn.edu/sunfest/docs/papers/12-bayer.pdf

[11] A ”Getting Started” Guide for Developers Interested in Robotics, http://learn.turtlebot.com

[12] Robotic simulation scenarios with Gazebo and ROS, https://www.generationrobots.com/blog/en/2015/02/robotic-simulation-scenarios-with-gazebo-and-ros/