Delivery 2 HOOVER

Teammates: Cristian García, María Salguero
Prof. Pablo Bustos

Lab Date: 03/10/18
Due Date: 07/10/18

Problem Statement

Simulates the movement of an hoover.

This robot must avoid obstacles in the map of the simulation and it has to go across the greatest area of the map in 5 minutes.

Research

The robot's movement implementation is based on the detection of obstacles by a laser. The laser measures the distance between the robot and the obstacles found in front and at the sides of it. When an obstacle is encountered, the robot makes a turn.

Initially, the robot's movement implementation had an error. The range of detection of the laser was very high, so the robot detected itself all the time and kept spinning.

At the beginning, the detection threshold had a value of 200. We changed this value to two new variables to avoid this error: "*FrontThreshold*", used to compare distance between the robot and an obstacle in front on it; and "*LateralThreshold*", used to compare the distance between the robot and an obstacle at any side on it.

Optimum Solution

The goal of this delivery is for the robot to go across, at least, 50 percent of the map.

We explain the solution in *Implementation* section.

Implementation

We marked a distance of 335 milimeters in front of the robot and 235 milimeters at the sides of it to detect obstacles.

As the aim of this project is that the robot goes across the maximum percentage of the map in the lowest time possible, we created a variable that controls the speed of spin. We obtain a better and more natural robot's turn movement. We increased the rotation (in radians) too.

Originally, the implementation of the robot's movement picked up the distance between obstacles and the robot with a laser located at the middle of the robot. Using a lambda function, it sorted an array with these distances from the smallest to the greatest.

We decided to split this array in 3 parts, dividing the range of light of the laser in 3 different areas. We can choose which part we want to scan first. In this way, we use the lambda function to sort each area independently.

We wanted to scan the front area first of all. We created two iterators that help us divide the distances stored in the array. "*medio*.^{a}nd "*medio2*.^{a}re the iterators that mark the beginning and the end, respectively, of the central area in front of the robot.

After that, we call sort method with the lambda function that sort each area established of the array. At the end, each iterator contains the smallest distance found and the angle where it is regarding the center of the robot.

Analyzing the left area of the robot, if the distance returned is smaller than "*LateralThreshold*", we check if the angle of this distance is lower than -1.35 radians, what would mean that the robot is almost parallel to a wall. In this case we make sure that the robot will not make a big turn (since it may not have enough space to complete it without hitting the wall) by setting the sleep time to a relatively small value, so that it turns at its default rotation speed but for a shorter time.

In case that, analyzing the right side of the robot, the distance returned is smallest than "*LateralThreshold*", we check if the angle of this distance is bigger than 1.35 radians. In this case, as before, we put the main thread of execution to sleep a short time and turn the robot to the left side..

If these distances are not smaller than "*LateralThreshold*", it means that there aren't worries about a possible collision between the robot and an obstacle at its sides. We have to check front area. We check if the distance returned is smaller than "*FrontThreshold*". In this case, there is an obstacle in front of robot. If the angle of this distance is smaller than 0, the robot turns to the right. If the angle of this distance is bigger or equal to 0, the robot turns to the left.

Finally, if none of these checks were true, there is no obstacle close by, so the robot moves forward. We obtain the closest distance to the robot using the std::min function and multiply it by a factor of 2.5 (to avoid making the robot too slow) to obtain the speed. Since this value cannot be greater than 1000, we limit it, and then make the robot move.

Final Evaluation

After 5 runs we have obtained a mean of 56 percent of the surface travelled. Our results are the following:

- First run : 59 percent
- Second run: 56 percent
- Third run : 57 percent

- Fourth run: 51 percent
- Fifth run: 57 percent