

ROBOTICS 2020

Marta García Tornero,¹ Irene Melchor Félix.
mgarciajr@alumnos.unex.es · irmelchor@alumnos.unex.es
University of Extremadura

CHOCACHOCA PRACTICE

Resumen—To perform this practice, the main tasks that have been carried out have been three, which have been served to start the robot, make it move by itself, and finally avoid obstacles.

I. INTRODUCTION

First of all, the student has been provided with a project on GitHub. This project consists of a map with a robot. When executing the project for the first time, the robot is static on the map, and the next step that has been carried out has been its start-up. In order to move the robot around the map, certain components have been installed that allow the robot to move through two options:

- The first way that has been provided is through a joystick, connected by the USB port of the computer.
- The second option using the computer keyboard.

In this case, the two options were tested to check the operation of the robot, with each device that was given us. Next, the MyFirstComp component was created, which will be explained later, to make the robot circulate alone in the room. Once the robot has been configured, the vacuum cleaner component has been started, this will serve to measure the percentage of sweep of the robot, in the desired time. Finally, a first sweep was carried out, thus verifying that the robot only cleaned 20 percent of the room, so our objective is to increase this percentage with a series of methods that we will see later.

II. INITIAL COMPUTE METHOD

Once in situation, the first thing to create is the MyFirstComp component, this component will allow the robot to move on its own without using the joystick or the computer keyboard, as mentioned before. Next, the compute method found in the SpecificWorker.cpp class of said component is modified. In this method the calls to the other methods that have been implemented will be made.

III. INITIALIZEMATRIX METHOD

The first method created in SpecificWorker.cpp is to initialize matrix. This method consists of "transforming" the map through which the robot moves into a matrix, which would go from -2500 to 2500, but since it is more complicated to use negative values, it is decided to add 2500 to each value, so the matrix ranges from 0 to 5000.

All the cells of said matrix are initialized to a false value, so that, as the robot goes through them, they change their value to true, in this way the robot will know through the cell that it has already passed.

IV. TURNMETHOD METHOD

Inside SpecificWorker.cpp a method is created that checks if the distance in front of the robot is less than or equal to the threshold created, (in this case it has been initialized to 300), and, if so, the robot turns at a random angle and call the checkMatrix method, which will be talk about it later. If the distance is greater than the threshold, then the robot continues forward at the highest possible speed.

V. CHECKMATRIX METHOD

The checkMatrix method is also found in the SpecificWorker.cpp class. This method checks, by means of a vector of neighbors, the 8 adjacent cells of the robot, in this way, it will know if all its neighbors have been visited. At the moment in which a neighbor that has not been visited is found, the robot will go in that direction, thus making sure, do not visit a cell that has already been visited previously. If it has already passed through any of the cells, the robot continues forward, and if not, it turns.

VI. FINAL COMPUTE METHOD

The compute method has varied a lot throughout these weeks, since at first it simply consisted of an if-else within the try-catch it contains. Later, a switch was created but, after several tests with the robot, it was eliminated to optimize execution, since with the current implementation it has been proven that it is not the most optimal. Another variation that the current code has undergone was to call a method that would make the robot go forward, but it was also eliminated due to slowdown problems. Finally, within the try-catch, the turnMethod method is called, which is responsible for calling the checkMatrix method, both explained previously.

VII. MISTAKES

At first, the compute method created and initialized the threshold variable to 200, but when modifying the code and adding new methods, this variable has been changed to a higher value, in this case 300, as mentioned above, because If this value was kept at 200, the robot, when moving forward

and turning, would go through some walls or some of the objects that were found along the way.

Another mistake that was made was not slowing down before the robot made a turn, so when approaching a wall or obstacle, the speed was too high and the robot did not have time to turn without colliding with the object. To solve this problem, the robot has been slowed down when turning.

VIII. EXECUTIONS

After carrying out the entire implementation, 5 executions of 120 seconds each have been carried out to study the behavior of the robot. As can be seen in each of the captures, the times oscillate between 32.32 percent and 35.52 percent. So the average obtained between the five executions is 33.856 percent.

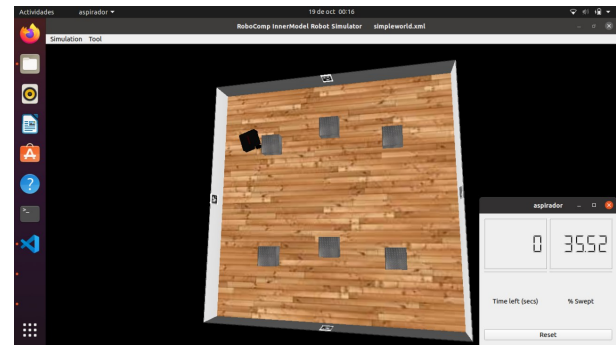


Figura 3. 35,52 percent

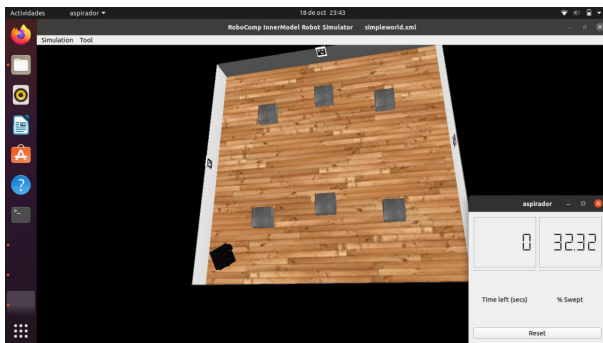


Figura 1. 32,32 percent

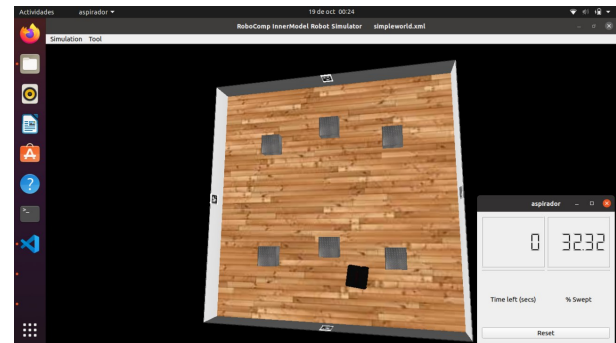


Figura 4. 32,32 percent

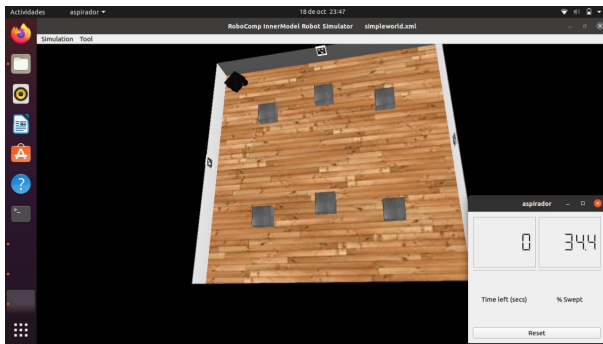


Figura 2. 34,4 percent

IX. CONCLUSION

As a conclusion, it can be added that the percentages obtained can be improved, so we will work on it throughout the sessions. We believe that to be the first contact with the subject, it is not bad at all, even so we will try to achieve the highest percentage possible.

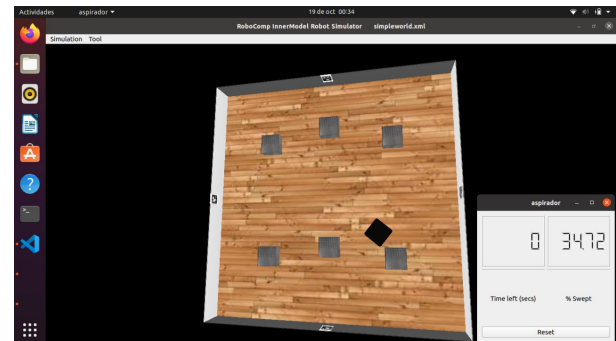


Figura 5. 34,72 percent

GOTOXY PRACTICE

Resumen—To carry out this practice, the main task is to make the robot go to a point on the map that the user clicks.

X. GOTOXY COMPONENT

For this release a new component called GOTOXY had to be created. This component will allow us to obtain the values once the user clicks somewhere on the map that is being simulated.

XI. COMPUTE METHOD

In this method the first thing we do is check if the target flag is active, if so, we get the values.

Then we obtain the angle and the distance to travel. Later, we perform the formula, which uses the methods discussed below.

$$advspeed = (1000 * reducespeedifturning * reducespeedifclosetotarget) \quad (1)$$

We have created a switch in which we have two states, state 0 checks if the distance is less than 30. In this case, the robot sets the flag to false and will stop, since it has reached its destination.

What state 1 does is enter the speed calculated with the advspeed formula and beta rotation angle.

XII. REDUCE SPEED IF TURNING

This section talks about slowing down when the robot turns. In this method the Gaussian function that can be seen below has been implemented.

$$\exp((-pow(rotSpeed, 2))/y) \quad (2)$$

The rotSpeed parameter contains the value of beta. We have created this variable to not modify this value. The value of τ has been calculated with the following function:

$$y = (-x/\log(s)) \quad (3)$$

By parameters enters $s = 0.1$ and $x = 0.5$.

XIII. REDUCE SPEED IF CLOSE TO TARGET

This method limits the speed of the robot based on the distance to the target with which it enters the function.

The min function returns the smallest of the given values. If the two values are equal it will return the first factor.

This function performs a linear function of the distance to the target. It will be 0 if the distance is 0, so the robot will stop, and it will be 1 if the distance is greater than 1000.