

# ROBOTICA 2020

G1-X1

Marta García Tornero, Irene Melchor Félix.

mgarciajr@alumnos.unex.es · irmelchor@alumnos.unex.es

University of Extremadura

**Resumen**—This document explains the problems and solutions that have been obtained when we have tried to control the movement of a robot in a simulation in the Robocomp environment. Next, we will talk about the different activities proposed throughout the sessions. With these iterations, the robot has been able to have functionalities that it did not have before.

- **Second activity:** Aims to move the robot through the environment, avoiding any object that is on the simulation map.
- **Third activity:** The objective of this iteration is to make the robot go to a specific point on the simulation map, which is given by clicking with the mouse. In this case, obstacles that may exist on the map will not be taken into account.
- **Fourth activity:** This iteration is a combination of the second and third activity, that is, in this case, the robot will have to go to the point indicated by clicking with the mouse avoiding obstacles.

## I. INTRODUCTION

Throughout the course, we will use Robocomp, an open source robotics framework based on component-oriented programming and C++ as the programming language.

Component-oriented programming is a branch of software engineering, based on the decomposition of already formed systems into functional or logical components with well-defined interfaces that are used for communication between components.

In this type of programming, components can be replaced quite easily when the user or the program requires it.

Therefore, taking into account all that has been explained, component-oriented programming is the one that best adapts to the environment to be used.

## II. SECOND TASK

### II-A. Problems

The problems we have faced have been mainly two. The first was that the robot did not move by itself, but via a joystick. The second problem was that the robot did not avoid obstacles and did not have a correct delimitation of the map. When solving the first problem, we encountered another setback. This time the robot swept a very small percentage of the map due to its slow speed. When solving the second problem, we observed that the robot invaded the space of the obstacles. This was because the initial threshold was very low.

### II-B. Solutions

To solve the problems mentioned above, we have followed a series of steps: In the first place, a new component called chocachoca was installed, which is responsible for driving the robot through obstacles avoiding colliding. By installing it, we solved the two main problems. Next, we increased the speed when the robot was going in a straight line. If the robot had to turn, the speed was decreased by half and a turning angle was introduced.

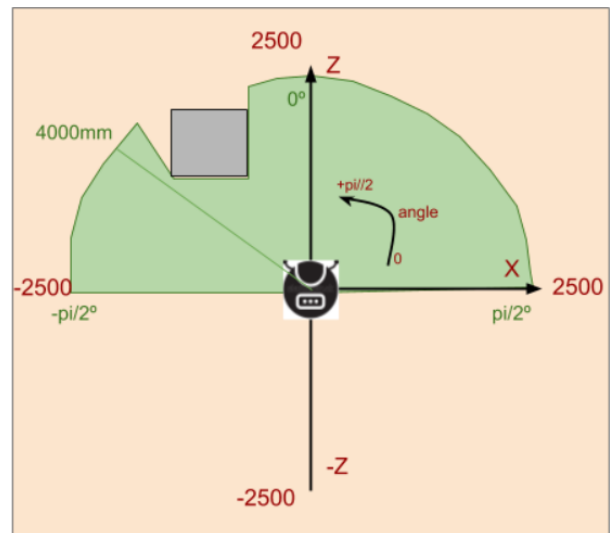


Figura 1.

Despite all these modifications, the robot was still sweeping a low percentage of the map, so we decided to implement an algorithm that checks the boxes adjacent to the robot, to verify if it had passed through them or not. In this way, it would not go through a box already visited.

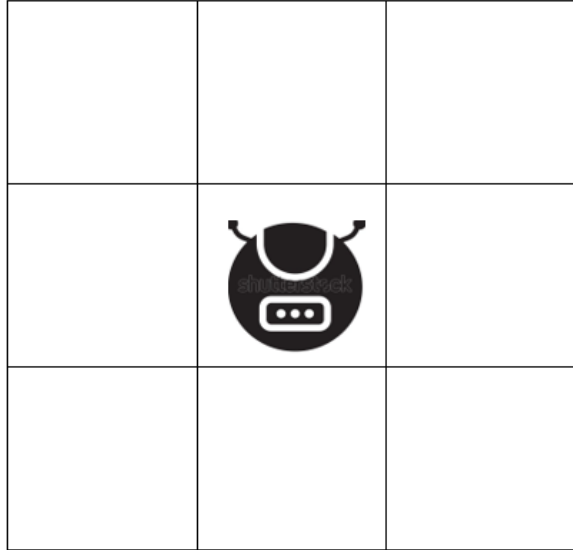


Figura 2. Adjacent matrix

### II-C. 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. Execution screenshots

### III. THIRD TASK

#### III-A. Problems

In this iteration, the first problem that could be found was to go to a given point on the map by clicking the mouse.

To do this, we needed to adapt the coordinates of the robot to the map, since the robot and the map do not share the same reference coordinate system, so we come across another obstacle.

Another problem we encountered was that the robot have to stop when it reached the point, it could not continue advancing, since it had already reached the target.

The last problem was to get the robot to have a continuous acceleration, since if not, it takes its highest speed and when it is close to the target it is not able to brake.

#### III-B. Solutions

To solve the problems explained in the previous section, the first thing we do is calculate the angle assuming that the robot is at the origin of coordinates, with the formula that we can see in the following figure, where beta will be our angle.

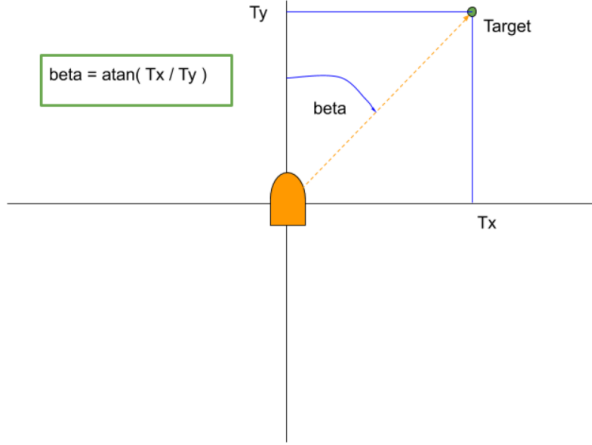


Figura 4.

Obviously, the robot will not always be at the origin of coordinates, so we are going to assume that it is at position (0,1), but without rotation, as we can see in the following image. The solution we can obtain is a geometric vector difference between the current position and the target position.

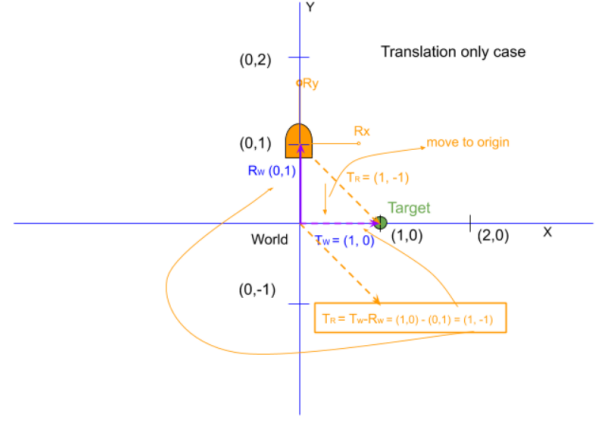


Figura 5.

Now, assuming the robot has rotated, we use the same geometric formula as above, but also adding the transpose matrix. The R is the clockwise rotating matrix built from alpha:  $\cos(a) \sin(a) -\sin(a) \cos(a)$ .

$$TR = RT(TW - RW) \quad (1)$$

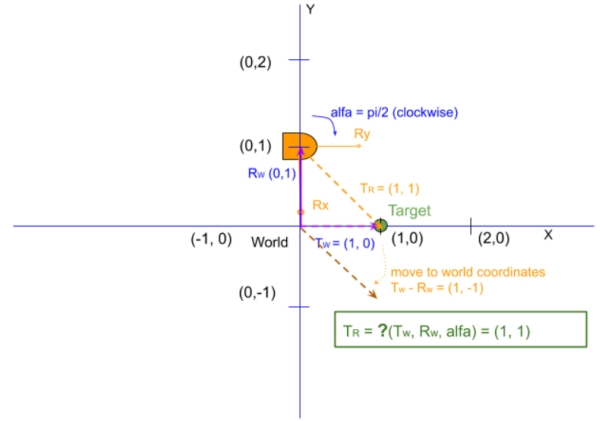


Figura 6.

To build the final controller you have two options:

1. A state machine with TURN and ADVANCE states
2. A simpler direct controller. The idea is to calculate the value of the forward speed and the rotational speed directly from beta and dist, so that they are coupled by mathematical equations.

We made the second option.

Now we need to slow down the robot, so we use the following Gaussian function:

$$y = \exp(-x^2/s) \quad (2)$$

This formula corresponds to the `reduceSpeedIfTurning()` method.

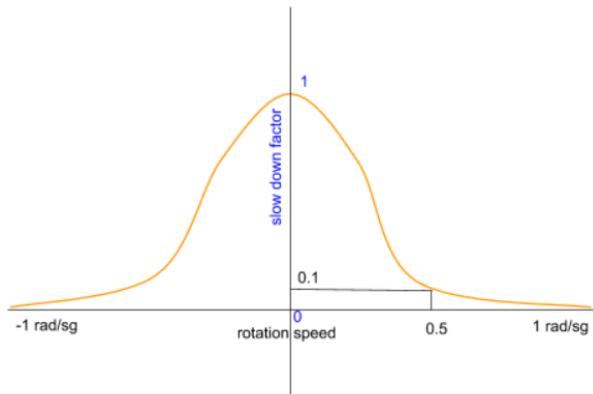


Figura 7.

The other factor to calculate is a linear function of the distance to the target.

The final speed formula to introduce it to the robot would be the following:

$$advspeed = (1000 * reduceSpeedIfTurning * reducesSpeedIfCloseToTarget) \quad (3)$$

For the `reduceSpeedIfCloseToTarget()` method, we use the min function.

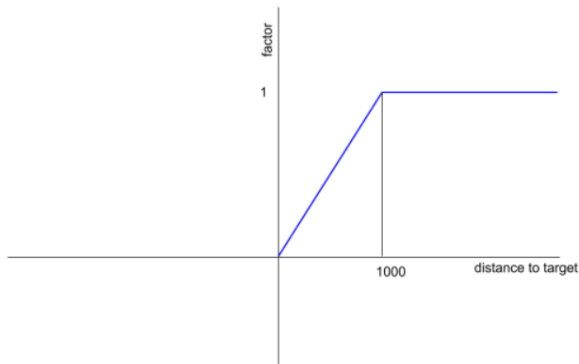


Figura 8.

### III-C. Executions

To demonstrate the functionality of this implementation, we have decided to record a demo video.

<https://www.youtube.com/watch?v=wU4YChe2Oio>

#### IV. FOURTH TASK

##### IV-A. Problems

The problem that we have encountered has been that the robot must avoid the obstacles that stand between it and the click made by the user. Due to this problem, we must find a solution that allows us to avoid these obstacles and finally, allow the robot to reach the selected target.

##### IV-B. Solutions

To solve the problems mentioned above, we have been provided with three possible algorithms:

- Bug Algorithm
- Potential Field Method
- Dynamic Window Approach

In this case we have opted for the third option, Dynamic Window Approach.

First, the laser polygon must be drawn and the possible points that the robot could access.

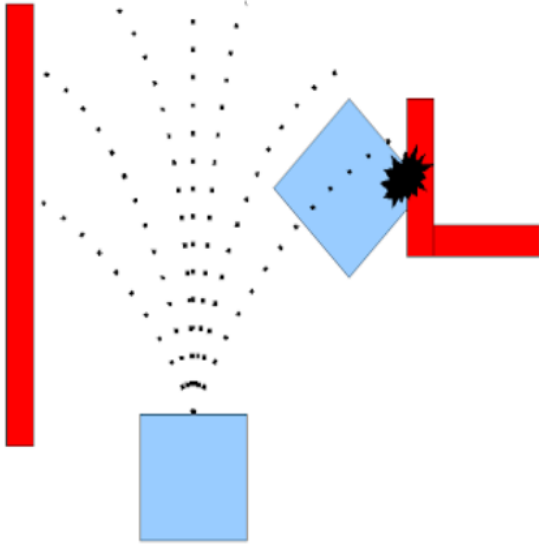


Figura 9.

To arrive at all the possible solutions that the robot can access, the different speeds of advance and rotation must be calculated for each possible movement, applied to its current state.

The coordinates of the new position  $P$ , seen from the center are:

$$x_c = r * \cos(w * dt) \quad (4)$$

$$y_c = r * \sin(w * dt) \quad (5)$$

where:

$$r = V/W. \quad (6)$$

Viewed from the robot's coordinate system:

$$x = r - r * \cos(w * dt) \quad (7)$$

$$y = r * \sin(w * dt) \quad (8)$$

and the new robot angle:

$$\alpha = w * dt \quad (9)$$

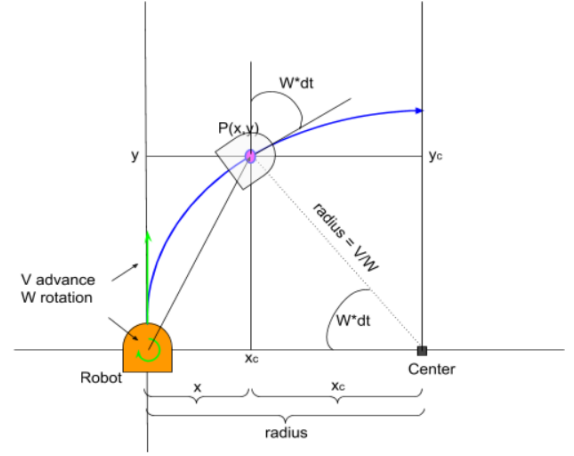


Figura 10.

Once these points have been drawn, a series of steps must be followed:

1. First the set of possible positions must be calculated, to do this the `calcularPuntos()` method has been implemented. Once these points have been calculated, the `obstacles()` method has been implemented, which will avoid any object that is in its path. After performing the previous methods, the laser polygon and the point cloud are drawn with the `drawThings()` method. In this way we can see if the points are being calculated correctly, and we can see if the point cloud is not drawn in the place where the objects are.

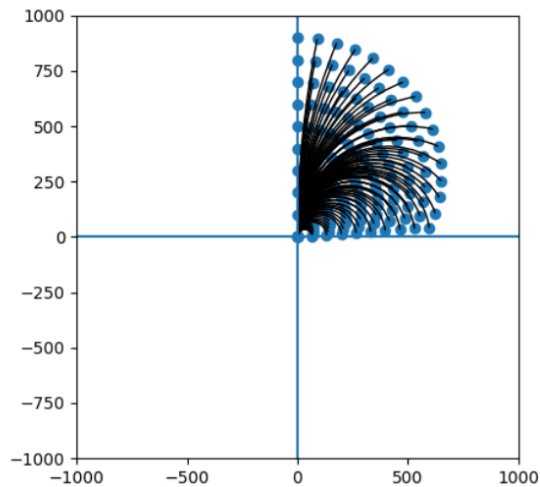


Figura 11.

2. When these positions have been calculated, the most optimal point must be chosen from two criteria to reach the target. These criteria are that the point is within the laser polygon and that it is as close as possible to the target. In this case, a method called `sort()` has been created, which will be in charge of carrying out these tasks.
3. Once these steps have been carried out, the calculated values must be entered into the robot.

#### IV-C. Executions

To demonstrate the functionality of this implementation, we have decided to record a demo video.

<https://www.youtube.com/watch?v=4vO2hAlgJhA>

#### V. CONCLUSION

In conclusion, we can add that it seems to us a very interesting and enriching subject, but at the same time very complicated and that it requires a lot of different knowledge, such as physics and mathematics, among others. We think that being a simulation it is probably easier than if we had to do it in real life. Despite all that has been said, we are convinced that the acquired knowledge will be of great help in the future.