UNIVERSIDAD PABLO DE OLAVIDE SEVILLA ESCUELA POLITÉCNICA SUPERIOR	**Robótica y Visión Artificial** **Ingeniería Informática en Sistemas de Información** **Robotics Challenge**

Robotics Challenge

Objectives

- Forming a complete robot navigation system by joining all the modules developed in EPDs 3, 4 and 5.
- Evaluating your system in simulation and in the real Turtlebot robot.

Assignments

1 Introduction

The final goal of the challenge is to build a functional hybrid architecture for robot navigation. To do that, we have been developing, in previous EPDs, different components of the reactive and deliberative architectures of navigation. Now, it is time to put them all together as can be seen in Figure 1.

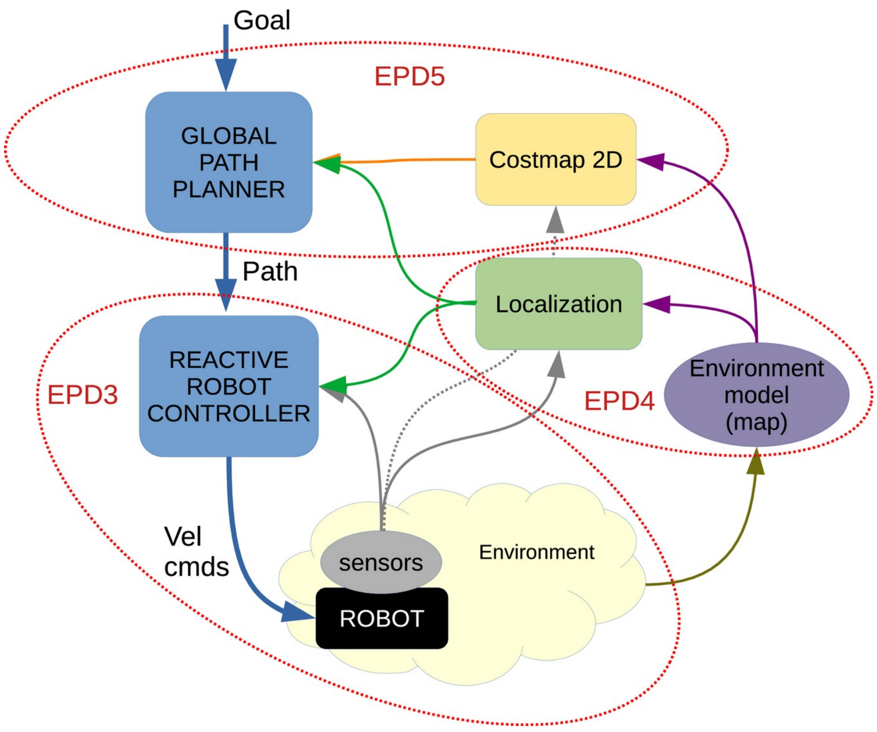

Figure 1. Navigation architecture

2 Challenge description

The challenge will consist in the execution of a set of predefined navigation goals in a given scenario (see Figure 2). We will use a simulated scenario as well as a different real scenario. During the navigation, a set of metrics will be computed in order to evaluate and to compare the students' systems (they will be detailed in the following section 3).

To execute the whole system in simulation, we provide a new ROS package with the launch files to run the simulation environment and the nodes and parameters required to run the system. You can download the package "robotics_challenge" from the directory robotics_challenge in the Aula Virtual. Download it, unzip it, and put it in your workspace.

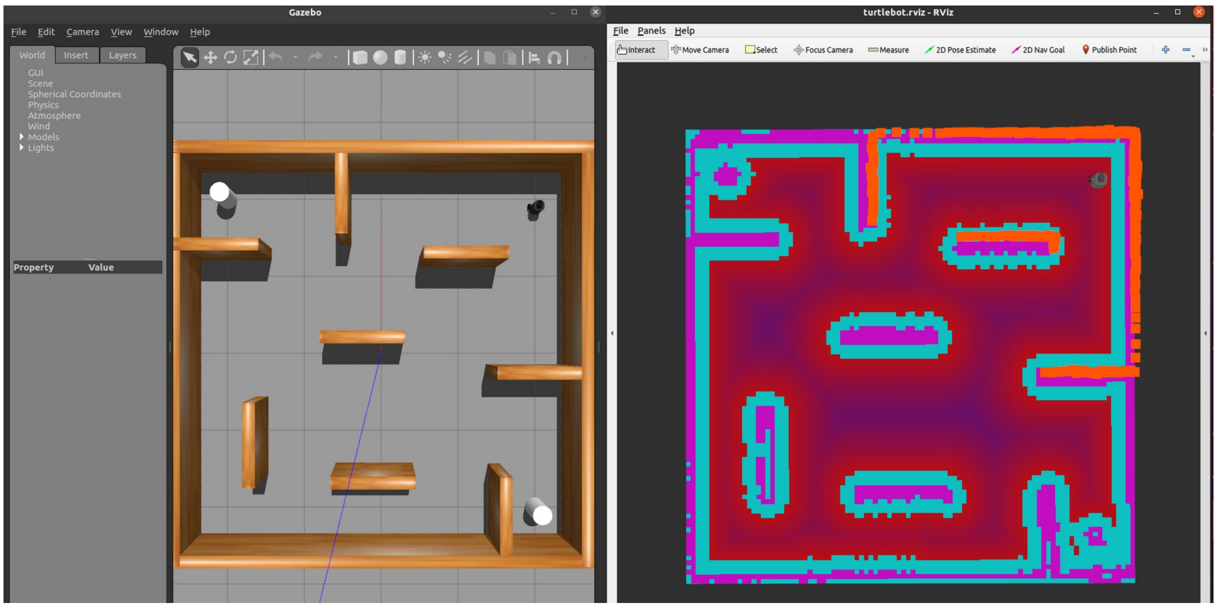

Figure 2. Simulated scenario and the associated costmap.

The robotics_challenge package contains:

- config:
  - *navigation_goals.yaml* → list of predefined navigation goals employed in the challenge.
  - *robotics_challenge.rviz* → a Rviz configuration file to visualize all that we need.
- launch:
  - *robotics_challenge.launch* → main launch file that includes the other launch files to run the simulation and the localization besides launching the nodes costmap_2d and your nodes *path_planer.py* and *robot_controller.py* from your other ROS packages.
  - *evaluation.launch* → launch file to run the evaluation program.
- scripts:
  - *evaluation.py* → main execution ros node. This node will be in charge of sending the navigation goals to your system while computing the navigation metrics.
  - *robotics_challenge.bash* → bash script to run the whole system. We use it to avoid the problem with the launching order of launch files. So that this bash will first set the enviroment variable TURTLEBOT3_MODEL and then it will run the launch file *robotics_challenge.launch*

To run the navigation system in a simulated scenario, you must firstly execute:

```
$ rosrun robotics_challenge robotics_challenge.bash
```

Then, to start the evaluation, open a new terminal and execute this:

```
$ roslaunch robotics_challenge evaluation.launch
```

3 Evaluation

The evaluation program will read the set of goals indicated in the file *navigation_goal.yaml*. Following the indicated order, it will send the first goal to the navigation system and will start to compute the metrics. The program will check when a goal has been reached in order to send the following one till no more goals are available in the list.

3.1 Robot controller constrains

In order to fairly evaluate all the navigation systems, we need to establish a set of parameters to be used for all the students. Therefore, **all the robot controllers must fulfill the following constrains:**

- **The maximum robot linear velocity allowed is 0.25 m/s**
- **The maximum robot angular velocity must be 1.2 rad/s**
- **A goal is considered to be reached when the distance to it is equal or less than 0.15 m**
- **The minimun distance measured with the laser sensor to consider a collision must be equal or less than 0.12 m**

With these restrictions, the evaluation program will compute a set of metrics for comparison of the system. Besides these metrics, the complexity of the methods implemented will be taken into account in the evaluation.

3.2 Metrics

The set of metrics computed automatically by evaluation program are:

- ***goal_reached***. Boolean value to indicate whether the final goal was reached (True) or not (False). If the final goal is not reached. The other metrics will not be considered. [initial value=False]
- **distance_traveled**. Distance in meters that the robot has traveled from its initial position to its final position [initial value= 0.0]
- ***elapsed_time***. Time in seconds that has lasted from the beginning of the evaluation to the end [initial value= 0.0]
- ***min_dist_to_obs***. Minimum distance, in meters, measured by the laser range finder sensor [initial value= 100.0]
- ***avg_dist_to_obs***. The average value of the minimum distances (m) to obstacles measured with the laser sensor at each time step [initial value= 0.0]
- ***collision_penalty***. A counter of the collisions with obstacles. We consider that the robot has collided when we detect a minimum distance equal or less than 0.12 meters in the laser readings. We add 1 for each time step in which that situation is detected [initial value= 0]
- ***lin_vel_penalty***. For each time step in which the robot exceeds the maximum linear velocity allowed, a penalty will be added. [initial value= 0]
- ***ang_vel_penalty***. For each time step in which the robot exceeds the maximum angular velocity allowed, a penalty will be added [initial value= 0]

Moreover, we will consider the cases in which the robot gets stuck and it needs manual intervention. In these cases, we can move the robot a bit in the simulator (or the real robot in real scenarios), and the teacher will manually add 1 to the metric manual_intervention_penalty.

- ***manual_intervention_penalty*** [initial value=0]

Besides that, there is a **time limit** to complete the task (200 seconds). If this time is reached, the program stops and generates the metrics with *goal_reached=False*.

3.3 Scenarios for evaluation

We will evaluate the metrics obtained in three different scenarios:

- **Simulated static scenario**
  We will use the static scenario presented in Figure 2.

- **Simulated dynamic scenario**
  In this case, we will use the same simulated scenario, but some “non-mapped” obstacles will be introduced in order to evaluate the skills for obstacle avoidance.

- **Real scenario**
  The real Turtlebot robot in a real scenario will be employed in this case. We will use a the same game board employed in the First Lego League contest presented in Figure 3. This board will be installed in the Service Robotics Lab, and it will be employed also for testing your algorithms. Two Turtlebot robots and a laptop will be available. The students must ask the teacher for a time slot to come to the Lab to test their algorithms. They just need to copy their ROS packages (robot_controller and/or path_planner) to the workspace in the lab laptop. The map and localization will be provided.

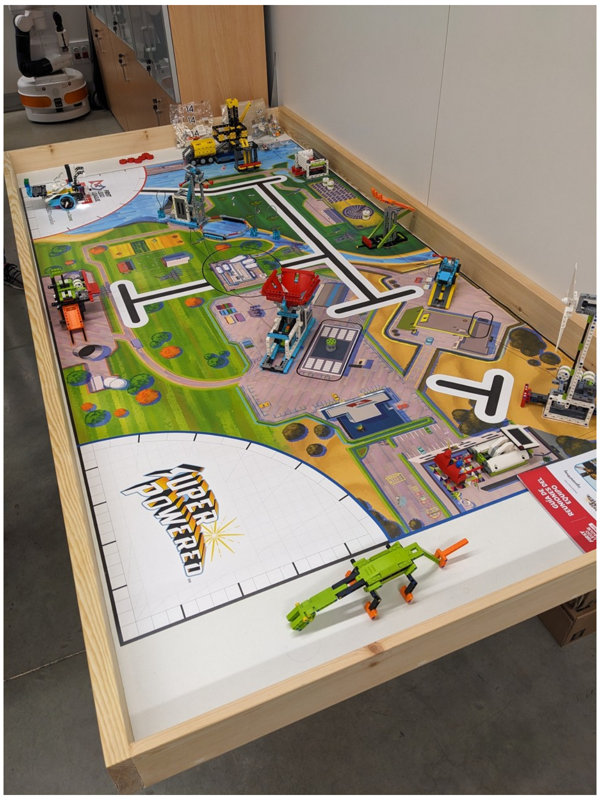

Figure 3. Game board employed in the evaluation with the real robot.