Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática

Robótica Móvel e Inteligente / Intelligent Mobile Robotics
(Academic year of 2024-2025)

Assignment 1

Robotic challenge solver using the CiberRato simulation environment

September, 2024

---

1 Objectives

In this assignment, each group should develop a robotic agent to command a simulated mobile robot in order to overcome a set of robotic challenges, involving different navigation skills.

The list of challenges is the following:

1. Control: The objective of this challenge is to control the movement of a robot through an unknown closed circuit as fast as possible and without colliding with the surrounding walls
2. Mapping: The objective of this challenge is to explore an unknown maze to extract its map. (No penalties for collisions.)
3. Planning: The objective of this challenge is to explore an unknown maze in order to locate multiple target spots and compute a shortest closed path that allows to visit all of them, starting and ending in the starting spot. (No penalties for collisions.)

Each of the challenges will be detailed later.

2 The CiberRato environment

The CiberRato simulation environment will be used to assess the agent developed to overcome the different challenges. The simulated robot (see figure 1) is equipped with 2 motors (actuating in the left and right wheels) and 3 LEDs (visiting, returning, and finishing LED). In terms of sensors, it includes a GPS, a compass, four obstacle sensors, a beacon sensor, a ground sensor, a collision sensor, and a line sensor. The available sensors depend on the challenge to be solved.

The simulated robot navigates in a delimited rectangular arena, 14 units tall and 28 units wide, with the diameter of the robot being the unit of measurement. This navigable area can be seen as a bi-dimensional array of fixed-size cells, each cell being a square with a side length equal to twice the diameter of the robot (2 units). So, the maximum size of the competing arena is 7-cells tall and 14-cells wide. A maze is defined by putting thin walls (0.2 units wide) between adjacent cells, which can be detected using the obstacle sensors. Each target spot has a unique identification number, which is detectable by the ground sensor.

The source code of the CiberRato simulation environment can be found at:

`github.com/iris-ua/ciberRatoTools`

Instructions to compile the tools are available at the **README.md** file of the repository.

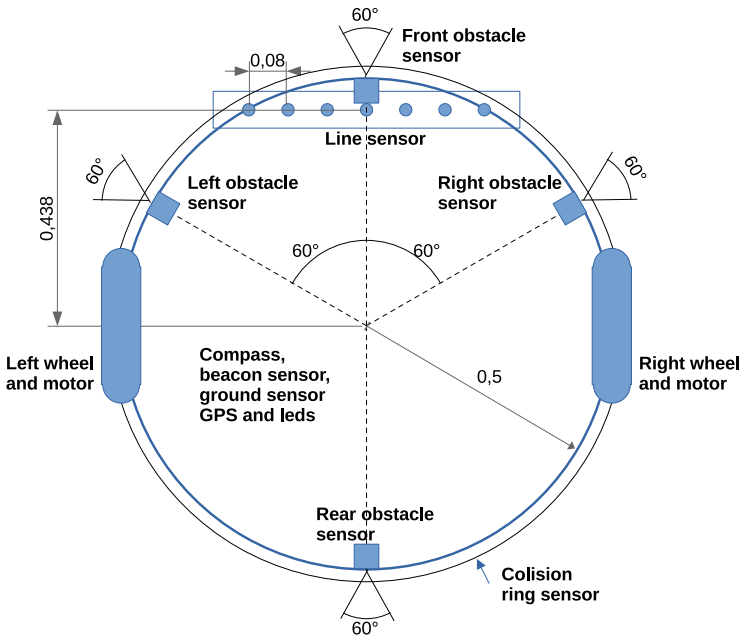

Figure 1: The simulated robot.

3 Challenges' description

3.1 C1 – Control challenge

As stated before, the objective of this challenge is to develop an agent suitable to control the movement of a robot through an unknown closed path as fast as possible and without colliding with the surrounding walls. Figure 2 shows an example of a maze suitable for this challenge, which will be available during development. As is visible in the figure, there can exist intersections, in which case the robot must always go ahead. At the start, the robot will be aligned with the X (horizontal) axis, pointing to the right, which corresponds to the North direction. GPS, beacon and compass sensors will not be available. The idea behind this challenge is to use the obstacle sensors and the motors to control the movement of the robot in the maze. There are checkpoints along the closed path, which can be used to determine whether the robot is traveling along the track in the correct direction. These checkpoints are sensed by the ground sensor, and therefore they can be verified by the agent.

The robot must complete as many laps as possible on an unknown path during a given time. Scoring (and so grading) will be based on the number of cells covered along the closed path in the given time.

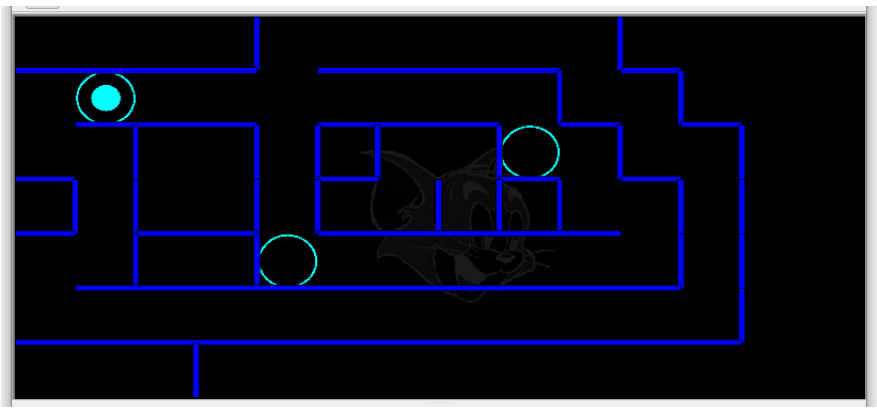

Figure 2: Example of a maze for challenge C1.

To start a test of this challenge, execute:

./startC1

3.2 C2 – Mapping challenge

As stated before, the objective of this challenge is to explore an unknown maze in order to extract its map. Figure 3 shows an example of a maze suitable for this challenge, which will be available during development. At start, the robot will be aligned with the X (horizontal) axis. GPS sensor and compass will be available, without noise. By using these sensors, the agent can know, at every cycle, where the robot is positioned and oriented on the maze. Hence, localization on the maze will not be a problem. Navigation can be done using the approach followed in challenge C1, but it is easier to do it using the noiseless GPS and compass. Collisions with walls will not be penalized.

The idea behind this challenge is to use the obstacle sensors to determine where are the walls throughout the entire maze. A strategy should be followed that ensures that the entire maze is explored.

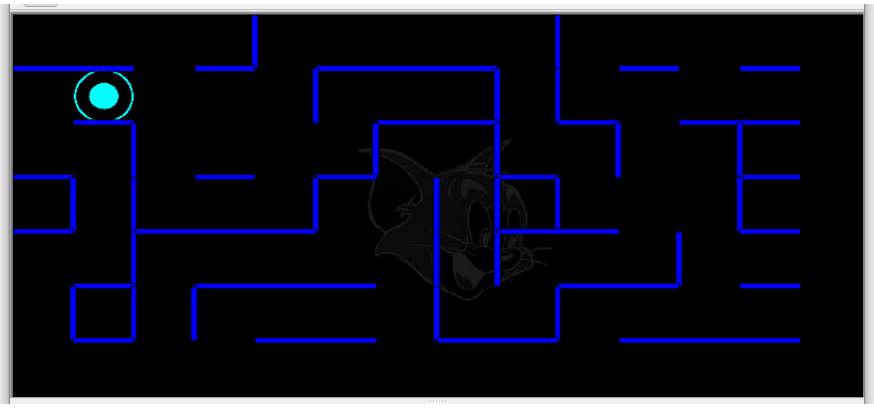

Figure 3: Example of a maze for challenge C2.

To start a test of this challenge, execute:

./startC2

3.3 C3 – Planning challenge

As stated previously, the objective of this challenge is to explore an unknown maze to locate several target spots and calculate the shortest closed path that allows one to visit those target spots, starting and ending at the starting spot. Figure 4 shows an example of a maze suitable for this challenge, which will be available during development. At start, the robot will be aligned with the X (horizontal) axis. GPS sensor and compass will be available, without noise. By using these sensors, the agent can know, in every cycle, where the robot is positioned and oriented on the maze. Hence, localization on the maze will not be a problem. Navigation can be done using the approach followed in challenge C1, but it is easier to do it using the noiseless GPS and compass. Collisions with walls will not be penalized.

The idea behind this challenge is to explore the maze just until the target spots are localized and the shortest closed path is identified. Therefore, the maze does not necessarily need to be explored in its entirety.

To start a test of this challenge, execute:

./startC3

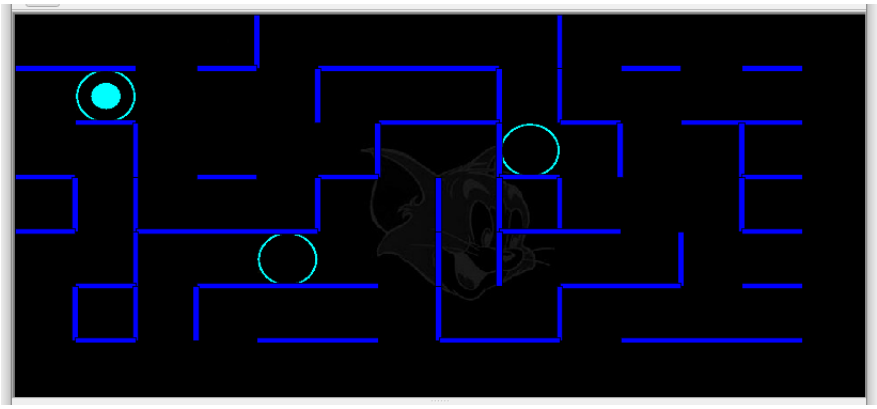

Figure 4: Example of a maze for challenge C3.

4 Movement model

Consider that the robot's pose is given by (x, y, θ), where x and y define the robot position and θ specifies the robot orientation. When the command sent to the simulator at step t is `DriveMotors(`in_t^l, in_t^r`)`, then the following equations determine the new robot pose.

An IIR filter is applied to each of the powers provided by the agent (in_t^l and in_t^r) that models the inertial characteristics of the motors and generates the effective powers that will be applied to the motors, corresponding to

$$out_t = \frac{in_i + out_{t-1}}{2} * \mathcal{N}(1, \sigma^2) \tag{1}$$

where out_t is the power applied at time t, out_{t-1} the power applied at time $t-1$, and $\mathcal{N}(1, \sigma^2)$ Gaussian noise with mean 1 and standard deviation σ.

Then, the movement is splitted in a translation of the robot position, considering its current orientation, followed by a the change of the orientation of the robot. For the translation one has

$$lin = \frac{out_t^l + out_t^r}{2} \tag{2}$$

$$x_t = x_{t-1} + lin * cos(\theta_{t-1}) \tag{3}$$

$$y_t = y_{t-1} + lin * sin(\theta_{t-1}) \tag{4}$$

and for the rotation

$$rot = \frac{out_t^r - out_t^l}{D} \tag{5}$$

$$\theta_t = \theta_{t-1} + rot \tag{6}$$

where D is the robot diameter (1 in the CiberRato environment). This provides the new robot pose (x_t, y_t, θ_t) at the next step, in case no collisions occur. If the new pose involves a collision, the simulator only applies the rotational component.

5 Scoring

5.1 C1 – Control challenge

For challenge 1, the simulator arguments should include `--scoring 1`. This way, the score that is displayed in the Viewer depends on the path length that is traversed by the robot over the line. The final assessment will be based on the scores of the agents for a fixed amount of time.

5.2 C2 – Mapping challenge

For challenge 2, the simulator arguments should include `--scoring 2`. This way, the simulator will create file `mapping.out` which represents the map of the environment. This file is created in the `simulator` directory. Your agent should create a file showing the robot's internal view of the map using the same format. This agent map file should have the extension `.map` (see `run.sh`, available on the Moodle website, for an example on how to create that file).

To check the correctness of a map file, you can use the `mapping_score.awk` script, executing

gawk -f mapping_score.awk mapping.out ≪***path-to-your-map-file***≫

in the `simulator` directory.

The map file is a text file that contains 27 lines, each with 55 characters. The center of the file always represents the starting position, marked as `I`. Below you can find an example of a map file.

```
1
2
3
4
5
6
7
8
9
10
11                    - - - - - - - - - - - - - -
12                   |XXXXXXX|XXXXXXXXX|XXXXXXXXX|
13                    - - X - X - - - X X - X - X
14                   |XXIXXXXXX|XXXXX|X|XXXXXXXXX|
15                    X - X X X X - - X - X - - X
16                   |XXX|XXXXXXX|XXX|XXX|XXX|XXX|
17                    - X X - X - X X - X X X - X
18                     |X|XXXXX|XXX|X|  |XXXXX|XXX|
19                    - X - - - X X X - - X X - X
20                   |XXX|XXXXXXXXX|X|XXXXX|XXXXX|
21                    X - X - - - X X X - - X - X
22                   |X|  |X|XXXXXXX|XXX|XXXXXXXXX|
23                    X - X X - - X - - X - - - X
24                   |XXXXXXXXXXXXXXXXXXXXXXXXXXX|
25                    - - - - - - - - - - - - - -
26
27
```

The final assessment will be based on the map score and on the time taken to finish the challenge.

5.3 C3 – Planning challenge

For challenge 3, the simulator arguments should include `--scoring 3`. This way, the simulator will create file `planning.out`, which includes the map of the environment and additional information on the best path to visits all targets.

Each execution of your agent should create a path file that specifies how to go from the starting position, visit all targets and return to the starting position. This file should have the extension `.path` (see `run.sh`, available on the Moodle website, for an example on how to create that file).

To check the correctness of the path file, you can use the `planning_score.awk` script, executing:

gawk -f planning_score.awk planning.out ≪***path-to-your-path-file***≫

in the `simulator` directory.

The path file is a text file that contains, in each line, the x and y coordinates of the center of the cells that are visited by the path. Coordinates are relative to the starting position and are measured in robot diameters. The starting and ending coordinates should always be `0 0`. Below you can find an example of a path file, written in 3 columns to save display space.

```
0 0
2 0
4 0
6 0
6 2
8 2
10 2
12 2
14 2
14 0
14 -2
16 -2
16 -4
18 -4
```

```
20 -4
20 -6
20 -8
18 -8
16 -8
16 -10
14 -10
12 -10
10 -10
10 -8
10 -6
8 -6
6 -6
4 -6
```

```
2 -6
2 -8
2 -10
0 -10
-2 -10
-2 -8
-2 -6
0 -6
0 -4
0 -2
-2 -2
-2 0
0 0
```

The final assessment will be based on the path score and on the time taken to finish the challenge.

6 Assessment

The assessment of this assignment will be composed of 2 components: an execution test of the agent and a presentation.

- The agents will be tested and graded by teachers, in batch mode, in their own computers. Thus, the format specified for their execution is mandatory.
- Each group will make a presentation of its work, consisting of an oral presentation, based on powerpoint or similar, and a short discussion of the work (10 minutes, maximum for presentation and discussion).

7 Deliverables and deadline

- Source code of the developed agent.
  - A single agent can solve all the challenges, or there can be a different agent for each challenge.
  - The code should be in a folder called `agent`, regardless of the programming language used.
  - `agent` folder should contain a script file called `build.sh` that allows to build the source code. Even if the code is developed in `Python`, the script should exist (it may do nothing). If you are using non-standard `Python` modules, please use this script to install them in a virtual environment.
  - `agent` folder should contain a script file called `run.sh` that allows to run all the agents. This script should accept options `-c1`, `-c2` and `-c3` to run the agent of the corresponding challenge. If you set up a virtual environment in `build.sh`, do not forget to activate it in `run.sh`.
  - **Source code must be submitted to the Moodle RMI website by November, 1st.**
- Presentation (in PDF format).

- The presentation should include:
  - Initial slide (names, nmecs, course, assignment, etc.).
  - For each challenge (localization, mapping, planning) you should present your **approach** for solving each (sub)problem, the experiments carried out and the **results** obtained (use the slides you deem appropriate).
  - Final slide with the conclusions of the work
  - Slide with bibliography/sources used in the work (this slide does not need to be presented).
- All slides should be numbered.
- **Presentation must be submitted to the Moodle RMI website by November, 6^{th}.**

8 Bibliography

- "Principles of Robot Motion: Theory, Algorithms, and Implementations", Howie Choset et al., MIT Press, Boston, 2005.
- "Introduction to Autonomous Mobile Robots", Second Edition, Roland Siegwart et al., MIT Press, 2011.
- "Artificial Intelligence: A Modern Approach", 4^{th} edition, Russel and Norvig, Pearson, 2021.