Intelligent Mobile Robotics

Robotic challenge solver using the CiberRato simulation environment

Rui Miguel Oliveira (89216) and Carlos Costa (88755)
Universidade de Aveiro
sec@det.ua.pt

1. Introduction

1.1 Problem context

In this project it was proposed to develop a robotic agent to command a simulated mobile robot in order to implement a set of robotic tasks, involving different navigation skills.

The list of tasks is the following:

1. **Localization:** The agent needs to navigate and localize itself in an unknown maze, using the movement model, the distance to obstacles and the compass. GPS is not available. Motors, distance sensors and compass are **noisy**. Collisions with walls will be penalized.

2. **Mapping:** The agent needs to explore an unknown maze in order to completely map its navigable cells. At the same time, the agent needs to localize target spots placed in the maze. The number of target spots may change between mazes. After completing the mapping task, the agent should return to the starting spot. Collisions with the walls will be penalized.

3. **Planning:** The agent needs to compute a closed path with minimal cost that allows to visit all target spots, starting and ending in the starting point.

1.2 Proposed solution

To solve the proposed challenge, we chose to:

1. **Localization:** Use **movement model formulas** along with the **robot sensors** to predict the GPS coordinates and the angle of the compass.

2. **Mapping:** Use an **A* algorithm[1]** to determine the closest navigable cell to the robot's current position and thus return the shortest path. From there, store the coordinates of the positions already visited and those yet to be visited.

3. **Planning:** After the full exploration of the map and the return to the initial position, the **traveling salesman problem** was used to return the shortest path among all the beacons.

(All programming was done in Python)

2. Methodology

2.1 Localization

To predict the coordinates and orientation of the robot, two mechanisms were used:

2.1.1 Movement model formulas

In this assignment we were given the robot without the help of GPS, so we had to calculate where the robot is so that we can navigate it through the whole map. To achieve that we used some **movement model equations** provided by the teachers and adapted to our movement code. To get the optimal localization for the robot, we need to preview the movement that the robot will make in the next cycle. To calculate that we need the translation of the robot position considering its current orientation followed by a change of orientation of the robot. To get the orientation for the next cycle of movement, we used this equation:

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

where θ_t is the orientation of the robot at that precise cycle, θ_{t-1} is the rotation of the robot on the previous cycle and *rot* is the rotation of the agent. To know the value of the rotation of the agent, we used the corresponding equation:

$$\text{rot} = \frac{outr - outl}{D}$$

where *outr* is the power of the right motor at that cycle, *outl* is the power of the left motor at that current cycle and *D* is the diameter of the agent. Since the diameter of the agent is 1, we can ignore this part of the equation.
To achieve the optimal value of the translation, we first need the value of Lin, which we can know by applying this equation:

$$\text{lin} = \frac{outl - outr}{2}$$

Now that we know both the value of Lin, we can get the values of X and Y for the translation with the help from both equations:

$$x_t = x_{t-1} + \text{lin} * \cos(\theta_{t-1})$$
$$y_t = y_{t-1} + \text{lin} * \sin(\theta_{t-1})$$

where x_t and y_t are the values of the coordinates X and Y respectively at the current cycle, while x_{t-1} and y_{t-1} are the values of the same coordinates during the previous cycle.

To determine the **compass** value, the theta of the movement model was averaged with the noisy compass. From there, the normalization of degrees was applied to obtain values between 0 to 180 and -180.

2.1.2 "Sensor's correction"

Considering that there is Gaussian noise with mean 1 and standard deviation σ, it was decided to use the sensors (all pointed at 90 degrees) when the distance to a wall was greater than 0.33 (diameters). Thus, as soon as the distance between the agent and a wall is greater than 0.33 diameters, we will no longer use the movement model formulas to predict the coordinates. At these times we will use the coordinate (x or y or both) of the wall closest to the robot, the thickness of the wall (0.1), the distance between the robot and the wall and the robot's radius (0.5) to predict de current coordinates.

Example of a **front** correction when the robot is facing **North**:

```
closest_x_wall_distance = 1/measures.irSensor[0]
closest_x_wall_coordinate = round(movement_model_x + 1)
sensors_x = closest_x_wall_coordinate - wall_diameter - closest_x_wall_distance - robot_radius
```

In the above case the sensors_x value contains the current value of x.

2.2 Mapping

The mapping algorithm has suffered some changes from the previous assignment. Before if all the nodes surrounding the agent have been traversed, it would go to a random node that hadn't been visited before. Now we implement an A* pathfinder algorithm to calculate which of the nodes that have not been visited is the closest, so that the robot could go there next if the conditions mentioned before happens. Using this new method, we can save time while exploring the whole map.

In this way the robot can be in three different states.

1. **In rotation**

   In this case, the robot will remain in this state until it reaches the value of the previously proposed compass.

2. **At the center of a cell**

   In this state the program makes a significant sequence of actions:

   - If the current coordinate is a beacon, its coordinate and identifier are added to a list of beacons.
   - The current state of visited positions is drawn in an output file.
   - The positions around the robot are evaluated in visited or to be visited
   - The A* map is updated (All map positions are initialized to 1 and whenever a navigable position is discovered the map is updated with 0 in these respective positions)
   - From the A* algorithm and the set of visitable positions, the nearest point and the respective path (set of navigable positions) from the robot to it is calculated. In case the distance between two points is the same, priority is given to the robot to change direction to the left or right and only then go forwards and backwards. If all positions have been visited, the robot returns to the initial position and the program ends.

3. **Between cells**

If the robot is crossing between cells, the following instructions are carried out:

- If the agent has not yet reached the next position (current coordinate plus 2 diameters) it advances.
- If for some reason it exceeds the predicted coordinate, it slowly retreats.

As there is a considerable amount of noise, sometimes the robot does not move perfectly. There is control over the distances between walls. In other words, whenever the robot deviates from the center of the cell or is too close to a wall in front, there is an adjustment of its movement.

Thus, the greater the distance between the agent and a wall, the greater the deviation to return to the center of the cell.

2.3 Planning

In order to determine the closed path with minimal cost that allows to visit all target spots we apply **the traveling salesman problem**. Thus, after exploring the entire map and returning to the initial position, the program simulates all **permutations** of positions between beacons and chooses the smallest one that starts at the initial position.

To determine the distances between beacons, we use the A* algorithm again.

2.4 Handling exceptions and collisions

The agent is somewhat prepared to handle collisions with walls and exceptions thrown at runtime.
In both cases, in order to preserve the current state of the map, since it is essential for generating the final path, the program leaves its natural run, calculates the final path and ends.
In this way, we will be left with a map that, although not fully explored, remains a valid map.

2.5 End of simulation time

If at some point the simulation time ends, the final path is calculated with the map explored so far and the program ends.

3. Results

On average our agent finishes the given map with a time remaining of 575 ticks.

Two output files are returned. One with the entire map scan and beacons position and the other with the optimal path between beacons in a closed circuit.

3.1 Map output file

```
 - - - - - - - - - - - - - -
|XXXXXXX|XXXXXXXXX|XXXXXXXXX|
 - - X - X - - - X X - - - X
|XX0XXXXXX|XXXXX|X|XXXXXXXXX|
 X - - - X X - - X - X - - X
|XXX|XXX|XXX|XXX|XXX|XXX|3XX|
 - X X - X - X X - X - X - X
  |X|XXXXX|XXX|X|1|XXXXX|XXX|
 - X - - - X - X X - X X - X
|XXX|XXXXXXX|X|X|XXXXX|XXXXX|
 X - X - - - X X X - - X - X
|X|2|X|XXXXXXX|XXX|XXXXXXXXX|
 X X X X - - X - - X - - - X
|XXXXXXXXXXX|XXXXXXXXXXXXXXX|
 - - - - - - - - - - - - - -
```

3.2 Path output file

```
0 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
18 -6
16 -6
14 -6
14 -4 #1
14 -6
16 -6
18 -6
18 -4
20 -4
20 -6
22 -6
24 -6
24 -4
24 -2
22 -2 #3
```

```
24 -2
24 -4
24 -6
22 -6
20 -6
20 -8
18 -8
16 -8
16 -10
14 -10
12 -10
10 -10
10 -8
8 -8
6 -8
4 -8
4 -10
2 -10
0 -10
0 -8 #2
0 -10
-2 -10
-2 -8
-2 -6
0 -6
0 -4
0 -2
-2 -2
-2 0
0 0 #0
```

4. Discussion

In conclusion, we reinforce that with few tools and even with noise it was possible to create a very interesting project where it could progressively increase in complexity if all the previously developed tools were sufficiently precise.

We believe that our agent will be able to adapt to different maps and apply its features to them.

Workload:
Rui Miguel Oliveira: 75%
Carlos Costa: 25%

5. Bibliography

- [1] A* Algorithm: https://github.com/BaijayantaRoy/MediumArticle/blob/master/A_Star.ipynb?fbclid=IwAR0caLz9Z7JVTpnJdKc200_DAxtHy94aIrZd0ZHKRwwzP05kypoI2B3ynX4