Project 1 – Robotic challenge solver using the CiberRato simulation environment

Rui Miguel Oliveira (89216) and Carlos Costa (88755)

Universidade de Aveiro, Campus Universitário de Santiago, 3810-193 Aveiro Portugal
sec@det.ua.pt

1 Introduction

In this assignment our group developed a robot agent to command a simulated mobile robot in order to overcome a set of robotic challenges, involving different navigation skills.
The respective challenges were as it follows:

Control: The objective of this challenge is to control the movement of our robot through an unknown closed circuit as fast as possible and without colliding with the surrounding walls.

Mapping: The objective of this challenge is to explore an unknown maze in order to extract its map.

Planning: The objective of this challenge is to explore an unknown maze in order to locate two target spots and compute a best closed path that allows to visit both target spots starting and ending in the starting spot.

We programmed in Python language and implemented search algorithms to accomplish our goals, and with our work we managed to achieve 3000 points in the Control challenge using the mainRobControl.py file, managed to print all of the map in the Mapping challenge using the mainRobMapping.py file and achieved to create the shortest path possible between the starting point to the beacons and back to the start in the Planning challenge (mainRobPlanning.py).

2 Challenges

In this section we will describe in detail our approach taken in each challenge:

2.1 Control

In the Control challenge, we were given the task of making a robot capable of running 10 laps in a closed circuit at the required time. To achieve this, we implemented four

(“high danger”, “medium danger”, “low danger”, “no danger”) different states, each one for the distance from the sensors to the walls of the map and to stop the robot from turning around during the run.

The robot should deviate to the opposite direction of the nearest wall.

The first state (“high danger”) has the highest priority so that the robot can react to the wall that is close to it first. The closer the robot is to the wall the greater will be the deviation. Furthermore, the front sensor has higher priority than the other sensors.

If the sensors don’t detect the walls at the determined intervals, the robot goes ahead with the maximum velocity (in this case, 0.15 in each motor).

To determine the distances between the robot and the target and consequently make decisions based on that, we use the formulas provided in the project statement. Despite the mathematical conclusions, we changed the values a little, due to noise.

The code is organized into four main conditions as indicated above, in order of priority. In addition, there is a lap counter.

2.2 Mapping

In the Mapping challenge, we were given the task of making a robot capable of exploring an unknown map and drawing it on an output file, “X” for spaces that the robot can move through, “-” and “|” for the walls of the map. To achieve this goal, we programmed the agent with a series of parameters.

First the agent calculates the offset of the GPS to receive the correct coordinates of the robot’s position on the map, then, depending on the position of the map, the agent previews its next move.

The agent has highest priority of moving to spaces around itself that have not been visited, then the priority is to move to the left, then to the front, then to its right, then lastly to its back, always checking its surroundings to see if it can move to each side.

The robot always evaluates each space around itself to check if the coordinates of those spaces have been visited. From each movement the robot makes, it appends the coordinates of its surroundings to a list of nodes that have not been visited. Each time the robot goes to a new node, it updates the list, removing the coordinates of its position that is currently at the list of nodes that have not been visited. Then it adds the same node to the list of positions that the robot has visited.

Every time the robot moves, it also updates the map to the file, using the parameters refereed up above. By the time the robot has traveled all the maze, it should have a drawing of the complete map on the same file and the agent automatic terminates the program.

The code is organized into three main parts.

- The "**going**" functions deal with movement.
  About the movement, there are four situations to consider:

  - Firstly, there is a control of the deviation that may occur due to the existing noise. In this case, it is always compensated for the distance to be covered in the next cell.

  - Secondly, whenever there is a need for the robot to change the direction, it will remain in a state of rotation that will only end when it reaches the predicted direction in the compass.

  - Thirdly, there is the manipulation of the decision of the next cell to be visited, as explained in our exploration strategy explained above.

  - Finally, there is the case where the robot will be able to use its engines in order to reach the next cell.
    In this forward movement there is another type of control of deviation in the robot's movement. This time this control is done between cells, that is, when the robot is not in the center of a cell. This way, whenever the robot deviates from the direction established by the compass, there will be a small deviation to return to the predefined direction.

- The "**evaluate**" functions deal with evaluating the regions around the robot and manipulating the lists of visited and unvisited nodes.

- The "**draw**" functions are responsible for writing the output of the map traveled so far in the output file.

The three parts described above are replicated in the four different directions (north, south, west and east)

2.3 Planning

In the Planning challenge, we were given the task of making an agent capable of searching the maze for two beacons with the objective of calculating the shortest path between the starting point and the two beacons back-to-back.

The type of movement that the agent makes during the realization of this challenge is the same as the second challenge (mapping challenge) does, with a few key differences.

Given the fact that the robot does not need to visit the whole maze, the priorities of the robot have been changed very slightly. This way the execution time is shorter.

Each time the robot passes through a beacon, it calculates the shortest path possible from the current beacon to the previous passed beacon with the help of an A* algorithm, creating a map that the robot has traversed with 1s and 0s, “1” being coordinates that the agent cannot go through and “0” being the opposite.

When all the beacons have been passed through, it appends all paths into a single path that describes the movement that the agent made and prints it in the output file.

3 Conclusion

During the realization of this project, we acquired knowledge of how to work with autonomous systems with sensors to resolve these challenges proposed to us and we gained a rough idea of how programming robotic intelligence is for future reference.

Workload:
Rui Miguel Oliveira: 55%
Carlos Costa: 45%