Robotic challenge solver using the CiberRato simulation environment

José Moreira[1][79671] and Bruno Aguiar[1][80177]

Universidade de Aveiro, Portugal

Abstract. This report describes the CiberRato simulation challenge, as well as the approach taken to beat it, the methods used and the results obtained.

Keywords: CiberRato · Robot · Localization · Mapping · SLAM

1 Introduction

The proposed challenge consisted in programming a robot to navigate in an unknown maze, with just the help of sensors, a compass and the movement model. The sensors available that were used were the front, left, right, back, and ground sensors. All these sensors, as well as the compass were noisy, meaning we could not trust them completely, and they were there as a complement to the movement model. The proposed solution has a really good success rate at the challenge, as most of the times it can be concluded.

2 Localization

To locate the robot in the maze, we used the movement model. Considering the information given in the assignment description, we can start by saying what was used, and what was not. We did not use the model for the rotation. Even though the compass had noise, it was still more suitable, at least for the testing that was done, to use it to rotate. The rotation was implemented in almost the same way it was when there was no noise, just a degree wider range that allowed to make up for the noise. With this said, the model was only used to move forward. To estimate the position on the (x,y) axis, or the translation movement, we start by calculation the left and right power output: $out_t = (in_i + out_t)/2$. After that, we can calculate the lin value, with $lin = (out_l + out_r)/2$ that is the value what will be added to the current position, x or y, depending on the current value of theta: $x_t = x_{t-1} + lin * cos(theta_{t-1})$ and $y_t = x_{t-1} + lin * sen(theta_{t-1})$. Regarding the values of theta, the used values were static, depending on the direction the robot faced, and assigned when each rotation was performed. However, this position predictions have an error, even if small, after some time accumulating it can miss a position entirely. To fix this, our strategy was to calibrate, eliminating said error with the front sensor. The robot is programmed to walk from cell to cell, always aiming to 'stop' at the center of each cell. When this happens, we can verify the value of the front sensor, to see if there is a wall. It is always the front sensor because the robot always moves forward, and not backwards. if the sensor detects a wall, at the distance that indicates that the robot is in the center of the cell, we assume that it is true, and change the current position that the robot assumes he is to the actual position, meaning, the center of the cell. Regarding location, our solution works most of the times, only failing when there is a collusion with a wall that was not avoided, because it causes a huge discrepancy between the theoretical speed and the actual speed.

3 Maze Discovering

Regarding the maze discovering, we kept a list of visited cells and a list of known cells that were not visited. With the a* algorithm we always went to the closest known not visited cell. The closest cell is calculated with: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, returning an Euclidean distance, as we were not able to calculate the Manhattan distance. This is a clear disadvantage, as sometimes the closest cell in distance takes a longer path. Although, for the A* heuristic, the Euclidean distance is the best choice as it doesn't overestimate the real distance to our goal. When a position had multiple cells at the same distance, the tiebreaker was to prioritize right direction, then front direction, then left direction.

4 Results

After completing the programming process, the testing process begins. We ran our robot 10 times on C4(for the purpose of this report) and it concluded the challenge 9 times, with an average time remaining close to 200. Of the 9 successful attempts, only one had collisions, small ones that allows the robot to not loose track of its position. The one it failed happened because a big collision happened. By big collision, we refer to a collision where the movement of the robot is fundamentally changed, maybe even stopped. This happens when a rotation occurs, and the robot is not fully centered in the cell. As the robot moves forward, a wall appears not in front, not on the side (90^{o}) but somewhere in between, that goes undetected. This however happened only a few times, as the results show. By successful, we mean the robot finishes in the starting position, after mapping the whole maze, the mapping file is created and written, as well as the pathing file.

5 Conclusion

During the process of finishing this challenge one of our biggest issues was the time it took for our searching algorithm to calculate a path. After trying a lot of different possible solutions, we discovered the issue was the optimization of the connections that was poorly done, making it took thousands of iterations just to calculate a path. This lead to conclusion number one, even regarding assignment one, that one of our main problems was the searching algorithm and the rest was mainly reusable. After solving these optimization issues, our shortest-path algorithm was able to take only a few iterations (with a maximum of around 25 iterations), and we were able to conclude the challenge without any other real issue. Overall, we are satisfied with the results we got, as the robot can complete the challenge 4 most of the times, however it is not perfect. Some future work would be to develop a more accurate way to make sure the robot is always on the center of the cells, that would remove the cases where it hits a wall making it loose track of its positioning, and it would also be possible to make it move faster on straights, if it was needed.