Capstone Project Thiago Lobato

Machine Learning Engineer Nanodegree December,18th,2016

**Plot and Navigate a Virtual Maze**

1. Definition

1.1 Project overview

This project was based in the Micromouse competition, and the goal is to guide a robot in a maze from a start position to the goal ( the center of the maze). The robot is allowed to run the maze twice, the first one freely to just map the maze and the second one to go to the center as fast as possible. This project handles the programming of such a robot with mapping and shortest path skills in a deterministic environment, where the robot has a orientation and receives sensor data of wall distances as it walks through the maze, the robot is allowed to move in only four directions, up, left, down and right.

1.2 Problem Statement

The problems in this project could be divided in three areas, mapping, exploring and obtain the fastest path. The goal is to make a robot that can identify its location and orientation, handle and map the maze walls/obstacles, explore the maze and, finally, be able to find the best path to the goal.

The solution approach will be to map the robot position based in its past position plus actual movement and, at the same time, map the maze walls with the sensor data, to start to build an image of the problem while exploring.

The exploration should be enough to find, if not the best, at least a good path to the goal without too many exploration steps. The way that this will be approached is trying to explore key points in the maze that would allow the discovery of a good path without waiting too much time in exploration.

The best way will then be achieved with a best path algorithm, which will use the already explored maze data to find the best direction in function of the actual robot position.

1.3 Metrics

The metric of this project is a combination of the first and second run of the robot. It is defined as the number of steps in the second run plus 1/30 of the number in the first one and it is automatic calculated in the tester.py script.

1. Analysis

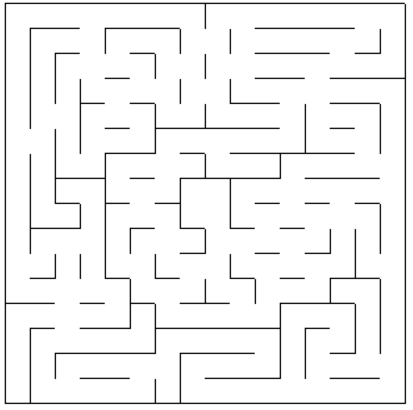
2.1 Data Exploration

The maze in which the robot will be in is considered a discrete grid maze. The robot itself is considered to always be in the center of each grid and has some orientation ('up', 'down', 'left', 'right') as well as some position, which correspond to his actual grid. He has sensors that tell how many empty squares there are in a respectively direction based on his actual orientation (his left, front and right). The robot moves in time steps, and in each one he is allowed to change his orientation in +- 90 degree and to move a number of steps ranging from -3 (opposite direction) to 3, inclusive. If it finds a wall, it will stop and will not go through.

One of the three mazes provided (most specific the number 3) has many ways of going to the center. It is closed in most direct path to the center and needs one to go around up to the end of the maze to turn back and find the solution, it has many redundant paths, as example in the top left and top right which can cost many time steps. The solutions differ a lot in length, range from more or less 50 squares going in the right way from up, right, down and avoiding useless obstacles going to more than 60 if ones chooses to use redundant ways.

2.2 Exploratory Visualization

The grid described before can be seen below:



2.3 Algorithms and Techniques

In this problem, all the sensor data is deterministic, which means we don't have to deal with uncertainty. Therefore one can use a approach that maps the position and orientation of the robot based in its previous position and movement. As for example, if the actual position is [2,4] and orientation 'up', after a 2 steps to the right movement and change of orientation of 90 degree clockwise, the new actual position would be [4,4] with orientation 'right'.

In the mapping, it was important to identify the wall positions (which would prevent movement) to clearly represent the maze. That was accomplish dividing the walls in: barriers from up ,left ,down and right. So each one would have its own matrix of mostly 0's where the 1's would indicate a wall in that position, which means that, in that square, the robot is not allowed to go in the specific direction.

The exploring was done as following: the agent would do a 'swing' exploring, going to each corner and then back to the center, and, in the end, would go back to the start, in that way the agent is able to get a great big picture of the maze and find mostly of the ways to the center(and hopefully also the best) with an efficiently exploring.

To find the fastest path after the exploration the algorithm chosen was A\*, which uses a heuristic function based in the actual distance from the center. Each grid has its own heuristic, which is the Euclidian distance from the goal, that distance plus is taken in consideration when choosing which path to follow. The barriers in each matrix were used in its respectively movements to model the allowed movement.

2.4 Benchmark

The benchmark of that problem could take many considerations. The first one being an agent that could not map its position and environment and would make just random moves. That agent would inconsistently reach the goal and with an undefined number of steps, taking the maximum of 1000 steps in consideration it would use 500 in the exploring and 500 in the second run (equal because each run would be independently from another and should have the maximum number of steps possible in order to maximize the chance of getting in the goal) having the minimum possible score.

The other option would be a robot able to map all the environment but not necessarily find the fastest path. But then which would be the best way to map all maze? With just random moves it would take way more than the actual number of grids, so it would be an optimization of its own. I would say that this problem has a difficult definition of benchmark, since there's no reasonable trivial solution and all other ones would be an algorithm comparison, not exactly a trivial benchmark like "test all functions values and find the minimum".

1. Methodology

3.1 Data Preprocessing

In this project, all the data was deterministic and provided through the model with the sensor data, location and mazes environment. Therefore no data preprocessing was needed.

3.2 Implementation

The implementation is not so complex, but it surely wasn't trivial. As commented before, the maze was a discrete grid, so all things related to positions were modeled with a matrix having each grid a row and column. The robot would consider its position and movements based in X and Y position, the matrix index in python, however, don't work that way. In a grid n x n, for example, robot position [0,0] would mean matrix position [n,0]. In order to consider all this it was created a vector that would map the y position in the right index.

The mapping was discussed in the algorithms and techniques part, where the sensor data was using to model different matrix from obstacles in the calculation of the fastest path.

A variable 'goal' was used to specify the goal to which the path algorithm should take. During the exploration run that goal changes between the center of maze and corners to try to explore the key points of the maze.

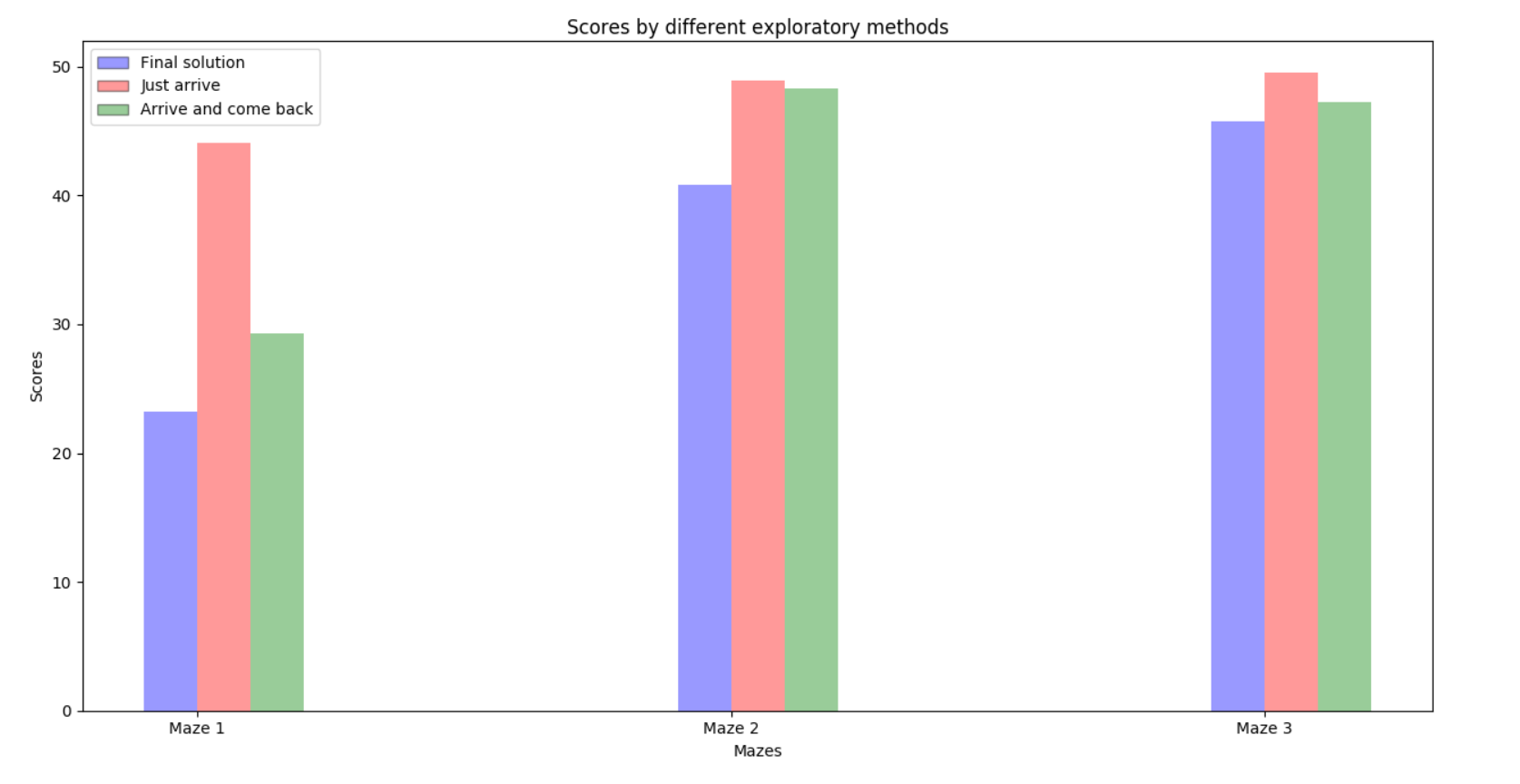
Also in the initialization it was created the heuristic function for the A\* method. It is created in two loops, taking in consideration the sum of each contribution for X and Y for the heuristic for an arbitrary maze size and goal. That heuristic is changed every time that the robot change the goal to explore.

At the start of each step the sensor data is used to start mapping the maze, this is done putting a wall in the matrix position corresponding to the actual robot position/orientation plus the empty steps in that sensor direction. Then the next direction to the actual goal is obtained with the A\* algorithm. That direction then goes to a function that converts the wished direction to a robot movement, based in his actual orientation.

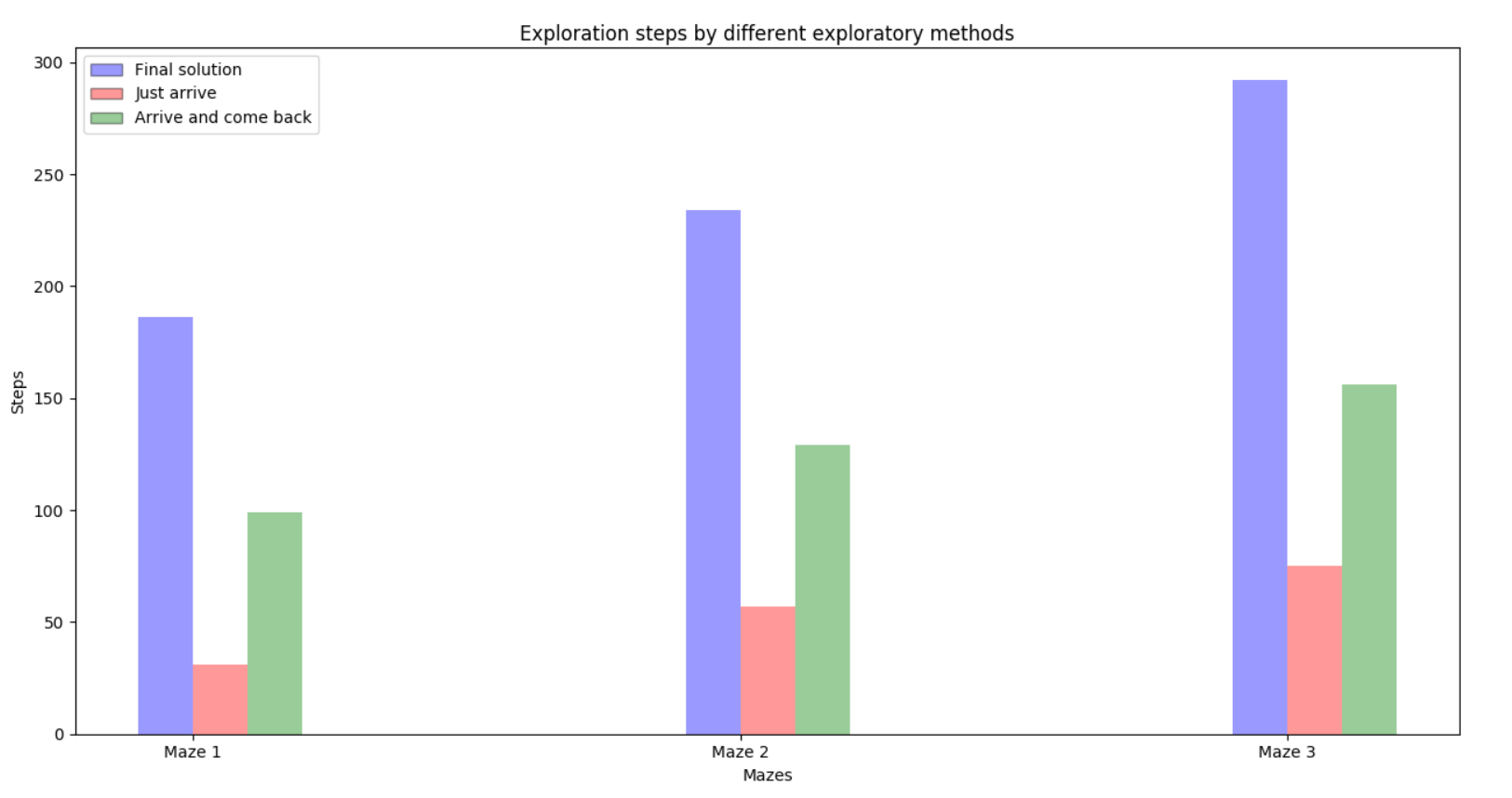
The new position and orientation are saved and the output is given. After it finds the first goal a new goal appears for exploration purposes until it comes back to the origin and start the second run using the fastest founded way.

3.3 Refinement

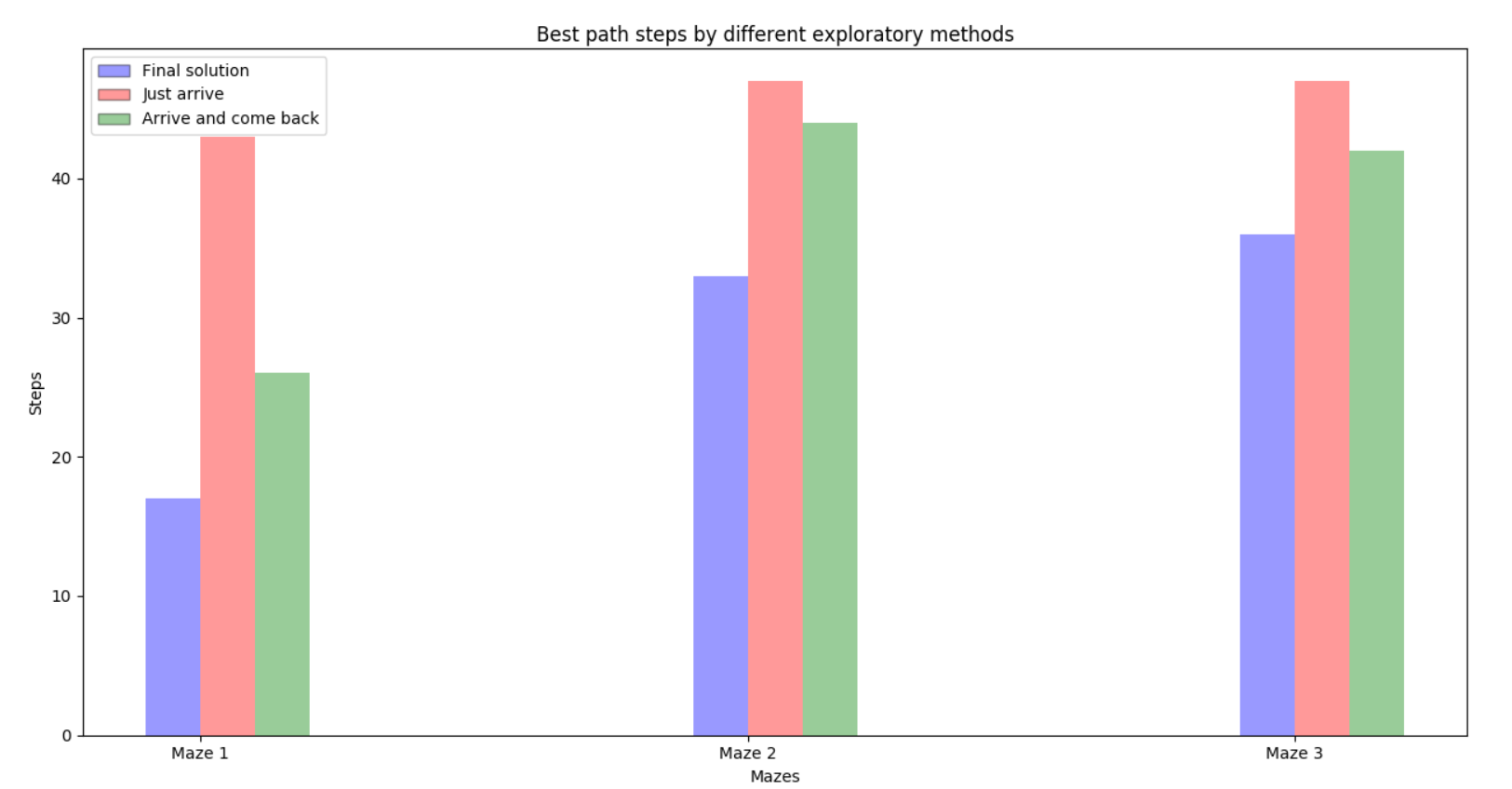
The first thing that was tried was to just arrive at the center with A\* and then start the second run. The number of training steps was very low, but the general result wasn't so high in comparison to the other methods. Than it was tried a more explanatory approach in which the agent would return to the start after the goal was founded. The results improved, but could still be better. The last one was the most exploratory of all three, which was discussed before, where the agent would do a swing search with center and corners to find most of the ways that would lead to it. In the figure below one can see a comparison of scores between the 3 explorations.



The increase in the exploration steps can be seen below:



The number of steps in the second run, however, decreases a lot with more exploration, which is what is expected, as seen in the figure below.



1. Results

4.1 Model Evaluation and Validation

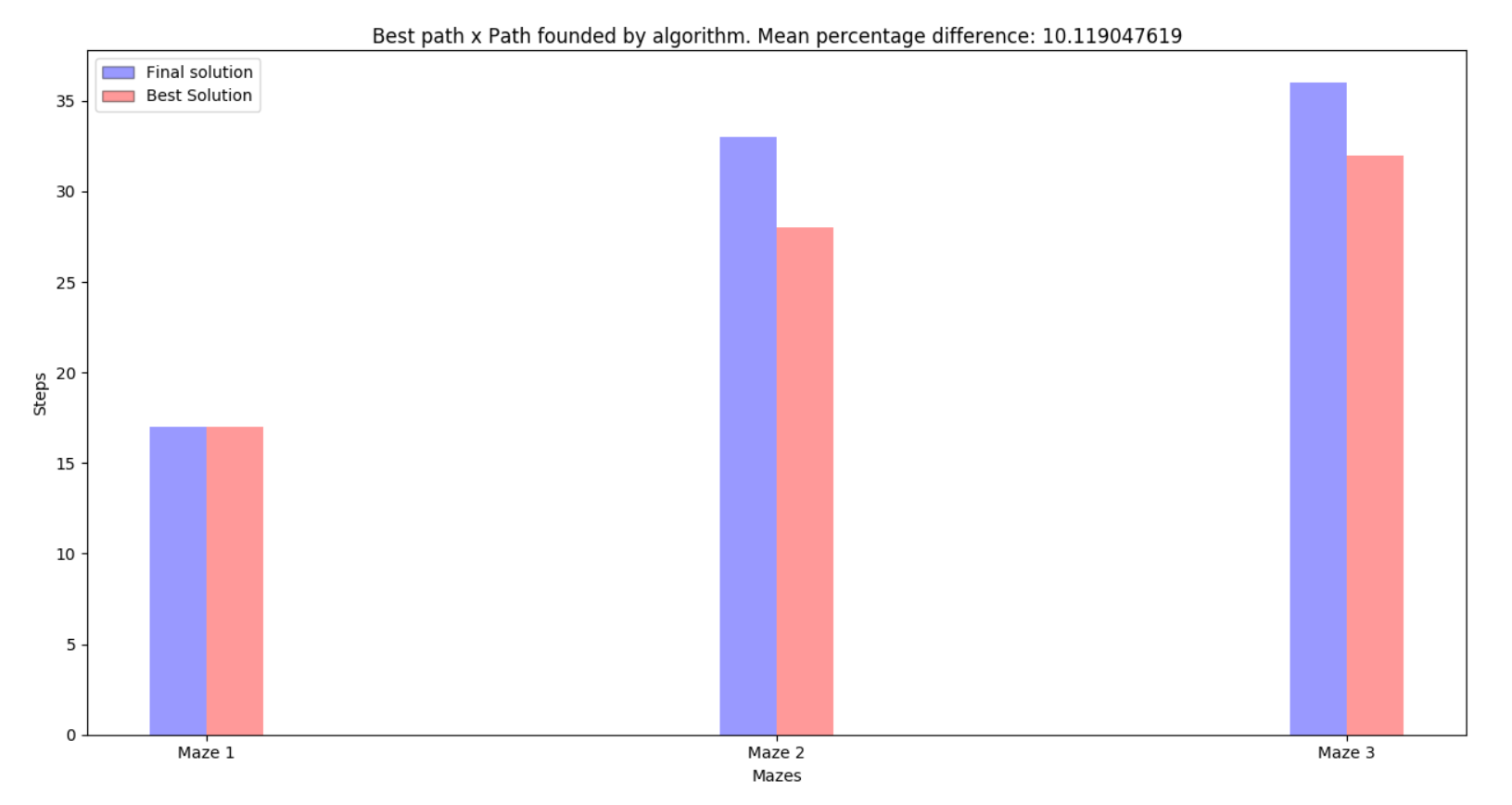
The developed algorithm can find a good path in all three provided mazes, and, as seen in the refinement section, is better than the other solutions.

The developed robot does exactly what expected in the beginning introductory part, it can robustly map its environment, not hitting the wall any single time and being able to map any wall that it sees. It can well explore the maze using a swing search in between the corners and the center and with A\* it can find the best path of the mapped maze.

The algorithm was robust to all mazes and also to the created one that will be discussed in the next section, it works well even with new perturbations (obstacles) and, since it does find the goal every time, the result model can be trusted.

4.2 Justification

Since the benchmark were not so strictly, it is better to compare the results with the best possible outcome. If we consider the best possible path for each one (this wasn't so easy to check, so I would give a 2/3 steps of uncertainty) we will have a mean difference of 10.11 %, which is a pretty reasonable result considering that the robot didn't explore all maze. The result can be seen in the figure below:

Results

5.1 Free-Form Visualization

5.2 Reflection

The problem was adequately solved, the robot is able to map the environment using its sensor data as way of store wall positions, to explore the map with some predetermined paths that would try to maximize exploration results and and find the best path from its mapped map with the A\* algorithm. The result is better than the benchmark and future improvements were discussed.

Specially difficult was the mapping part with the A\*, the idea to use four matrix to map the barriers and then to implement it in the propagation from A\* was quite challenging for me. I'm not sure it is the best implementation but I was proud of it.

Specially interesting was that whole idea of mapping, it is amazing that we can teach a robot to map himself good enough to go in unknown environments with just a few lines of code. Before I even started studying machine learning I thought those kind of things would be out of my world.

5.3 Improvement

The problem solved here was in a discrete world, which is not the case in many real applications. Depending of the problem we can, however, discretize the problem, if or robot would have a radius of 0.4 units, the walls being 0.1 units thick, it is still possible to discretize the domain working for the robot to not hit the wall based in some sensor metric. Like implementing in code a continuous sensor value to model both a discrete grid position and a maximum distance from the wall metric, allowing to still be able to use grids but taking care about the added complexity. The most difficult case I can think of would be some very curve labyrinth, which would ask for a more discretized grid to work properly.

Having a continuous domain would allow one to also use a smooth path transition. In the grid world we consider just very 'hard' movements. This is not always the case in a car, for example. Thus being able to smooth movements would be also a great improvement.

Another point that wasn't considered here was the uncertainty of sensors. We worked in a deterministic world, if we were to handle uncertainties, some algorithm like Kalman filters and particle filters could be used to have a nice position distribution which would be able to work even in noisy environments.