Robot Planning for Coordinated Evacuation

Samuel Bortolin, Alessandro Grassi

***Abstract*—The aim of the project is to simulate the behavior of robots locked in a room with obstacles that have to find the way out. Such robots will perform the maneuvers to evacuate the map by avoiding obstacles and the other robots. In order to compute the paths the room is divided in cells by using the *vertical cell decomposition* algorithm, inside each cell the centroid is found so a graph can be built on those points, after this the best path is found out of the graph using the *UCS* algorithm and the trajectory is computed using the multi-point Dubins curves to connect those nodes. The entire code is written in C++ on top of the provided simulator, C++ allows to have very good performance in terms of execution speed.**

***Index Terms*—Robot Planning, C++, ROS, UCS, Coordinated Robot Evacuation.**

I. Introduction

The project focuses on solving a robot evacuation problem on top of the provided simulator [1]. The problem is to drive three robots from their initial position to the emergency gate (i.e., target) in minimum time. The requirements are the following:

1. The robots move at a constant speed (so they can execute Dubins maneuvers).
2. The robots have to move without touching the border of the map and the obstacles.
3. The robots must not collide with each other.

Moreover, it is developed upon the following assumptions:

1. The robots are initially deployed at a random position.
2. A ROS component will provide the exact position of the obstacles and of the robots in real-time.

The problem is initially solved using our algorithm and then simulated inside the *RViz* tool [2].

Our implementation [3] has been developed entirely in C++ completing the function *planPath* inside the file *student_interface.cpp*. The solution we are finding is optimal on the roadmap with the assumption that we are in a synchronous environment. Every time that the command *AR_run* is executed the *planPath* is called once, and just one step is executed, meaning that the robots move from just one node to another on the roadmap.

II. Proposed Approach

A. *Assumptions*

For the sake of this project before starting to implement the solution we stated some assumptions. We decided to assume that the robots behave synchronously, which means that the system evolves in a sequence of states and each state is characterized by having the robot located in one of the nodes of the roadmap. In this way, every step of the plan is ended when all robots have reached a location and no robot is allowed to move to the next location before a step completes. This can cause some discrepancy between the planned behavior and the real behavior since some robots might crash into each other in the real one. Despite this, the situation is very unlikely to happen since robots navigate only through nodes that are far enough away from other robots, and since all the agents are choosing the optimal path taking into account the occupancy of the nodes they will never cross paths.

B. *Algorithm*

The algorithms to drive each robot from its starting position to the exit gate can be summarized with the following points:

1. Obstacles inflating
2. Obstacles merging
3. Vertical cell decomposition

4. Finding out the centroids
5. Creating the roadmap
6. Finding out the optimal path (mission planning)
7. Perform the path connecting the optimal nodes using the Dubins curves (motion planning)

The first step was to inflate the obstacles, this is a useful preprocessing step that allows to avoid crashing into obstacles. To do this we took advantage of the Clipper library [4].
To make computations easier and have clearer results we merged overlapping obstacles together and then took the convex hull of the resulting obstacles as final ones. For this last step we also used the Boost library [5].

We divided the environment into feasible cells, such cells have been found using the vertical cell decomposition algorithm [6]. From each vertex of each obstacle we drew a vertical line until the edge of the map or some other obstacle was reached.
In order to find the nodes of the roadmap we found the centroids of each cell and the points that lie in the middle of lines that separate adjacent cells. Then, we connected them to all the others if they did not cross obstacles in order to have all the possible connections and optimize the movements of the robots as much as possible at each time step.

In order to find the optimal path we performed an UCS [7] on the roadmap starting from the target and updating accordingly the costs in order to compute the optimal costs from each node and then simply follow the best improvement at each time step.
After the optimal path is found the nodes are connected optimally using Dubins maneuvers [8]. More on these last two points will be treated in the next section.

III. Solution

A. *Mission planning*

The motion planning phase focuses on deciding the path to choose from the roadmap, so the optimal sequence of nodes that leads to the target coordinating with all the other robots. To perform this task we start from a preprocessing step where the optimal cost from each node to the target is found. This is done executing a *UCS (i.e., Uniform Cost Search)* from the target node. The *UCS* explores the graph prioritizing nodes by its cost, the cost in our case is defined by the euclidean distance. Each time a node is explored its optimal cost to the target is annotated so to be used during the search of the optimal path. In this way we don't have to recompute the exploration of the roadmap for each robot, we just need to compute this preprocessing step once and then search on the found values. With this method we find immediately the nodes that can't reach the target, each node is indeed initialized with optimal cost equal to *-1*, if this value is not modified means that the node was not reached during the exploration of the roadmap from the target.

Once the preprocessing is done, it is finally possible to compute the optimal path, to do so from each robot the neighboring node with the lowest cost value is chosen. Intersection problems are dealt with at this level by checking that the chosen node is not too close to another future robot position, so the policy is to choose the optimal node that is far enough from the other robots, it can also happen that the best option is to not move.

B. *Motion planning*

Now that we have the optimal path on the roadmap we can compute the optimal trajectory to connect the nodes. To do so we used the Dubins curves which are the optimal curve that satisfies the following constraints:

- Initial state (both position and direction)
- Final state (both position and direction)
- Maximum curvature angle

The job of the solver is to find the optimal combination of all possible curves or straight lines that respect the above conditions. The choices that the solver can make are: *curve right (R), curve left (L), go straight (S).*

We compute a different Dubin Curve for each node in the roadmap, but to define the optimal angle at the arriving node we use multi-points Dubins curves from the target node *(i.e., the gate)* back to the starting node. To compute the multi-points the solver uses an

iterative dynamic programming algorithm [8] where it tries to minimize the length of the total path.

IV. Results

We were able to construct an evacuation plan for the robots in many different maps. We have obtained pretty good performance that are measured by the following KPIs:

1. Time to construct the roadmap: it is less than 5ms on simple maps, ~20ms on average on more complex maps.
2. Time to execute the task (i.e., the interval from the start of the mission to the time the last robot evacuates through the exit): less than 1 minute (considering the fact that we executed it on a virtual machine with limited hardware resources) using a curvilinear abscissa of the movement of the robot from one point to the following one in reaching the next node (ds) set to 0.03 and maximum curvature set to 15, but it is very map-dependent and depends also on how much the robot has to travel and the number of steps to reach the gate. In simple maps we are able to complete the task in less than 30s.
3. (Negative) number of times the robots touch the walls or an obstacle: most of the collisions with walls and obstacles are avoided by the fact that we inflated the obstacles and also using the Dubins maneuvers we should not bump into them.
4. (Negative) number of collisions between the robots: most of the possible collisions with other robots are avoided by the fact that we consider in advance the next positions of the robots and we will avoid these to be too close at the same step, but some situations where the robots paths cross one with the other may still rarely happen.

The following images report a testing case that we computed in different maps. We reported the initial configuration of robots and obstacles, the inflated obstacles, the cells generated by the VCD algorithm, the initial roadmap and how robot positions and roadmap evolve with the steps.

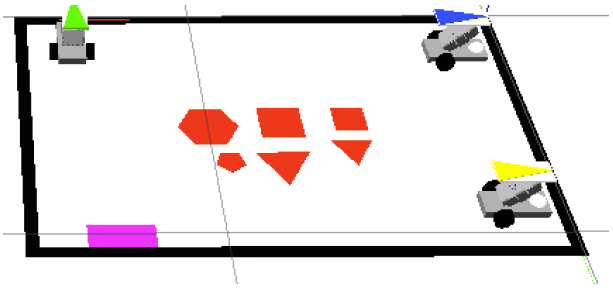

Fig. 1. Initial map configuration

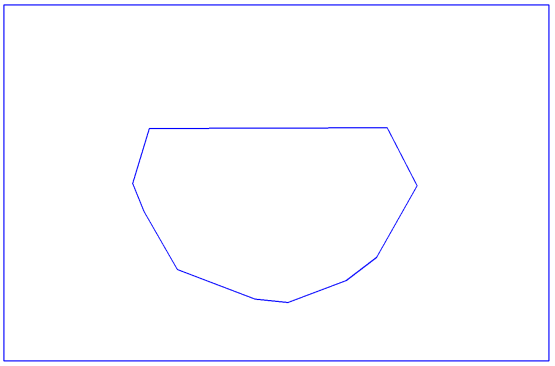

Fig. 2. Inflated obstacles

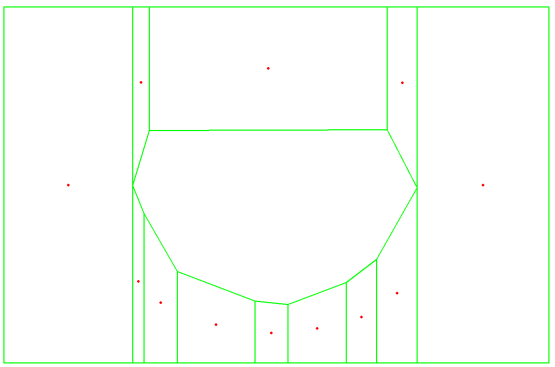

Fig. 3. Resulting cells generated by the VCD algorithm

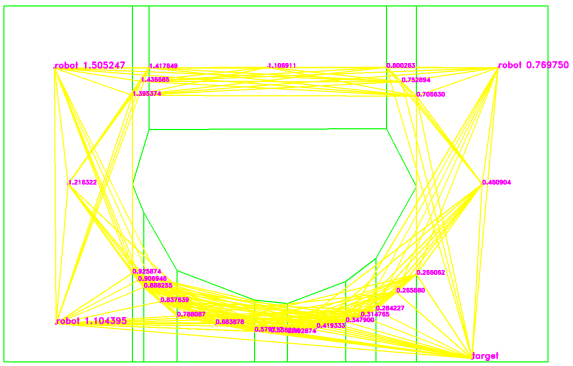

Fig. 4. Initial roadmap with links and cost for each node

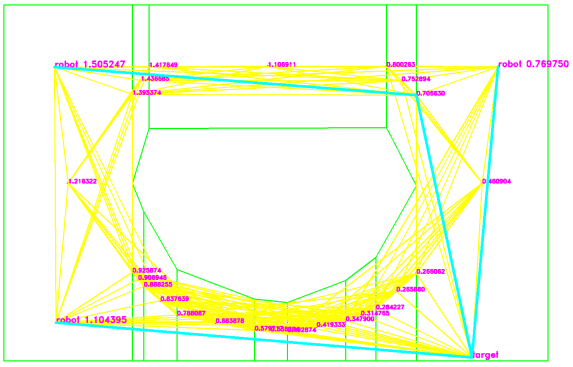

Fig. 5. Initial roadmap with highlighted the best path for each robot

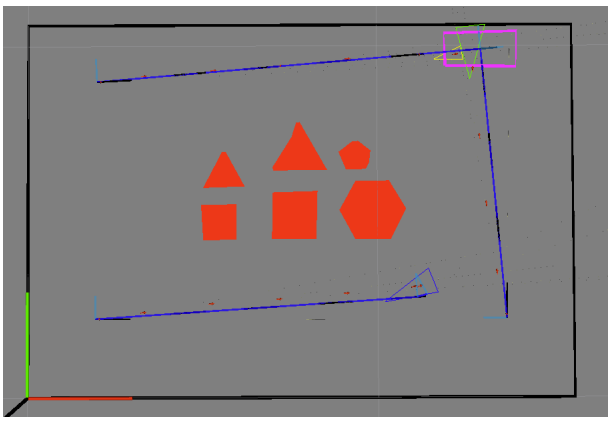

Fig. 6. Evolution of the robot positions after the first step

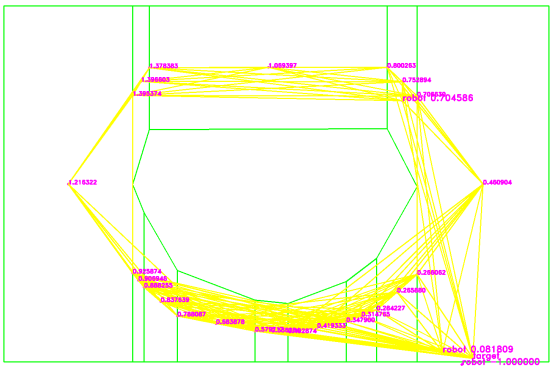

Fig. 7. Evolution of the roadmap after the first step

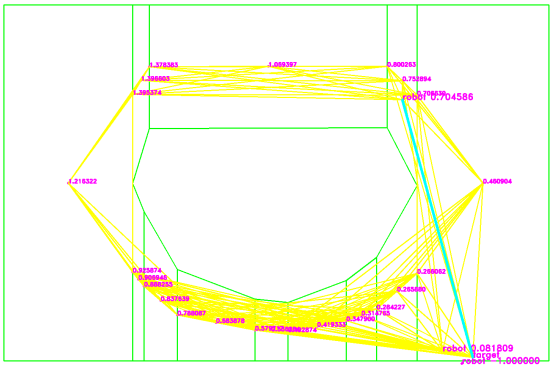

Fig. 8. Roadmap with highlighted the best path for the remaining robot that has not reached the gate after the first step

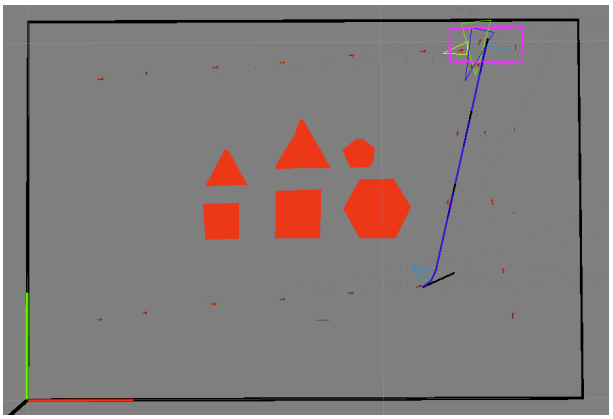

Fig. 9. Evolution of the robot positions after the second step

In the Appendix some other images for other testing cases are reported and analyzed.

V. Conclusions

In this project an escape scenario has been approached by inflating obstacles, dividing in cells the map via vertical cells decomposition, finding the centroids of each cell, creating the roadmap using the centroids and the points that lie in the middle of lines that separate adjacent cells as nodes, finding the optimal path on the roadmap using an UCS-based strategy and finally follow this optimal path using Dubins curves to reach the gate in minimum time.

The found solution has good performance in terms of computational time, to perform the whole process it takes less than 300 ms on the virtual machine. Even the correctness of the solution is good even if there is still room for improvement. Out of our tests the robots never crash on the obstacles and are always able to compute the optimal path to the exit gate, the only catch is that sometimes robots crash with each other. This was expected since we are assuming a synchronous environment without counting on the intersection or proximity of trajectories.

A future work is for sure to compute the optimal path taking into account the trajectory that the robots will follow so as to avoid having them crashing into each other. Another improvement might come by having a better policy for the cell decomposition since sometimes our approach is not completely able to fully decompose cells correctly.

References

[1] Fork of the provided simulator repository with some minor changes: https://github.com/samuelbortolin/AppliedRoboticsEnvironment

[2] http://wiki.ros.org/rviz

[3] Fork of the provided student interface repository with the implementation: https://github.com/samuelbortolin/AppliedRoboticsStudentInterface

[4] https://sourceforge.net/projects/polyclipping/

[5] https://www.boost.org/

[6] Implementation of the vertical cell decomposition algorithm from which we took inspiration in developing our code https://github.com/Shikherneo2/path-planning/blob/master/vertical_cell_decomposition.py

[7] B. J. H. Verwer, P. W. Verbeek and S. T. Dekker, "An efficient uniform cost algorithm applied to distance transforms," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, no. 4, pp. 425-429, April 1989, doi: 10.1109/34.19041.

[8] M. Frego, P. Bevilacqua, E. Saccon, L. Palopoli and D. Fontanelli, "An Iterative Dynamic Programming Approach to the Multipoint Markov-Dubins Problem," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 2483-2490, April 2020, doi: 10.1109/LRA.2020.2972787.

The authors have the following addresses: {samuel.bortolin, alessandro.grassi}@studenti.unitn.it.

This report is the final document for the course of “Robot Planning and its application”.

Appendix

Here are reported some other examples to show the algorithm in action on different maps.

A. *Test2*

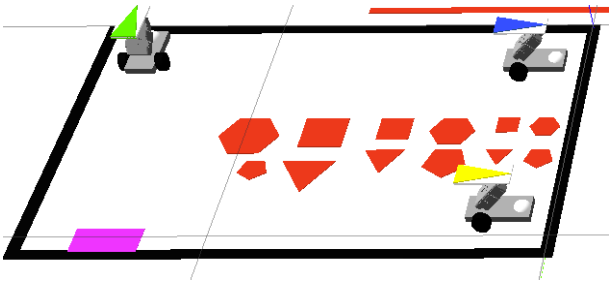

Fig. 10. Initial map configuration

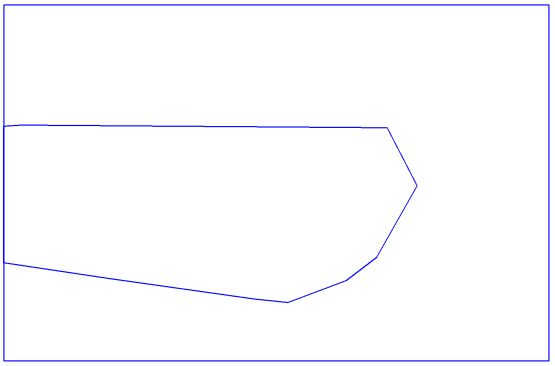

Fig. 11. Inflated obstacles

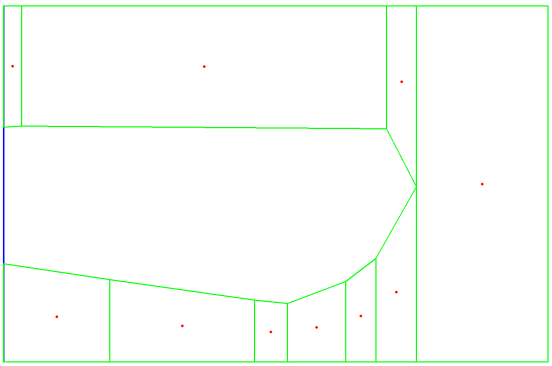

Fig. 12. Resulting cells generated by the VCD algorithm

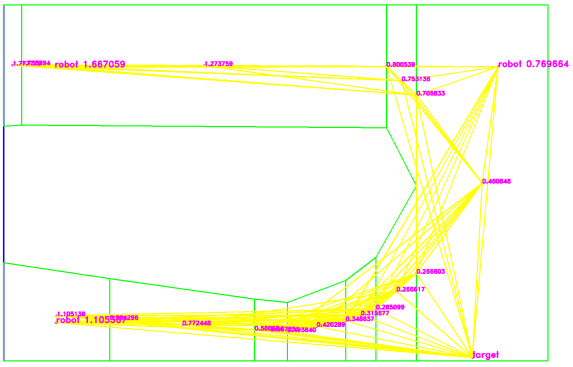

Fig. 13. Initial roadmap with links and cost for each node

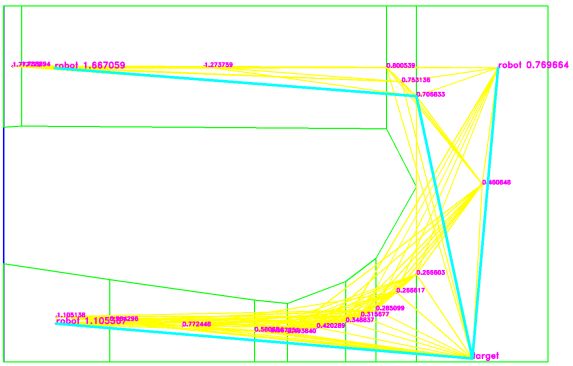

Fig. 14. Initial roadmap with highlighted the best path for each robot

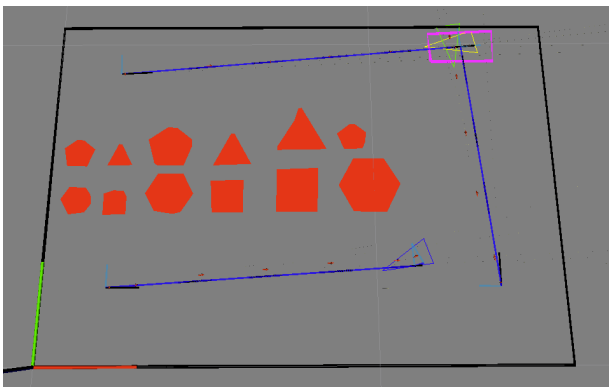

Fig. 15. Evolution of the robot positions after the first step

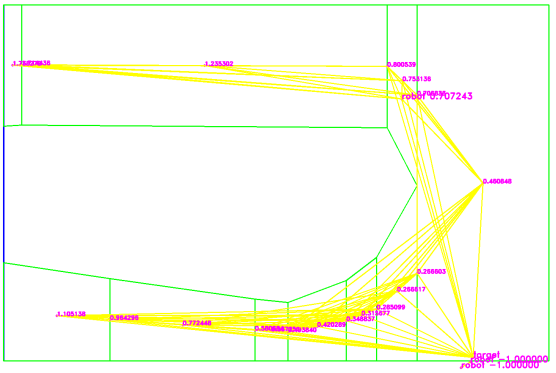

Fig. 16. Evolution of the roadmap after the first step

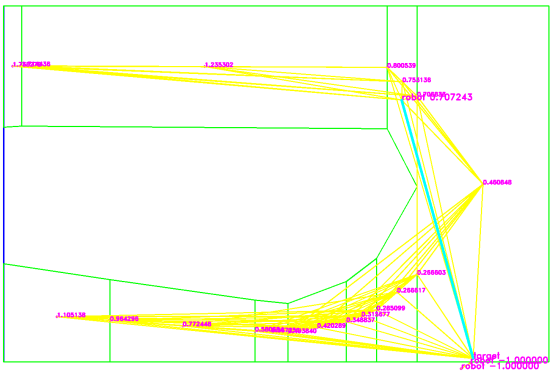

Fig. 17. Roadmap with highlighted the best path for the remaining robot that has not reached the gate after the first step

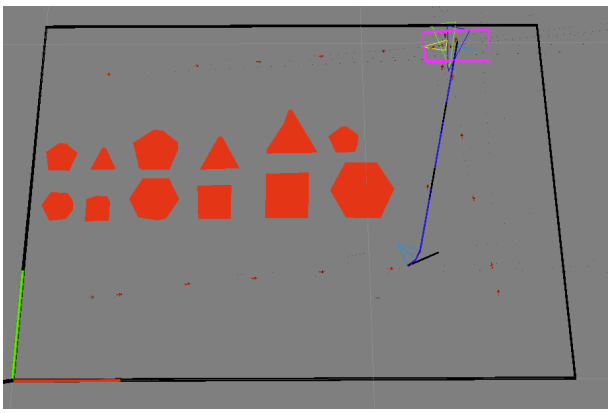

Fig. 18. Evolution of the robot positions after the second step

B. *Test3*

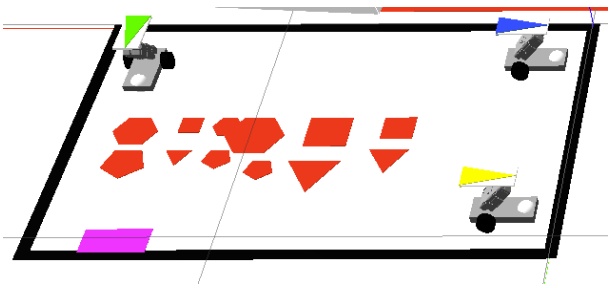

Fig. 19. Initial map configuration

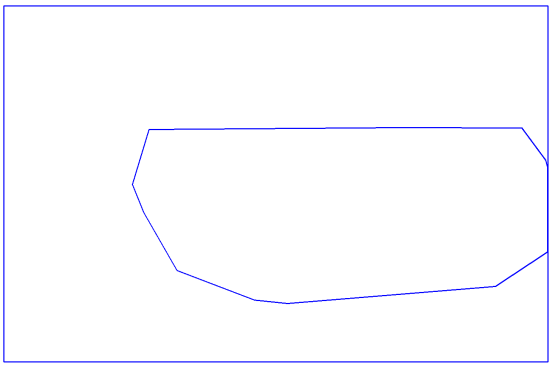

Fig. 20. Inflated obstacles

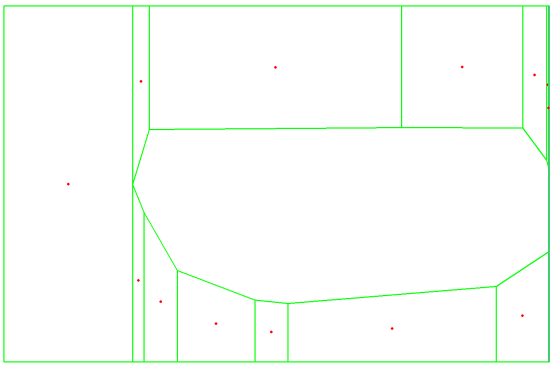

Fig. 21. Resulting cells generated by the VCD algorithm

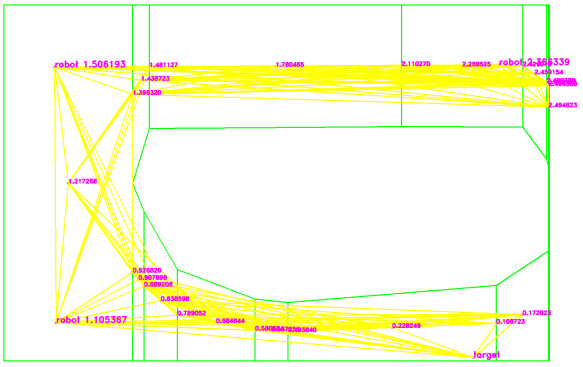

Fig. 22. Initial roadmap with links and cost for each node

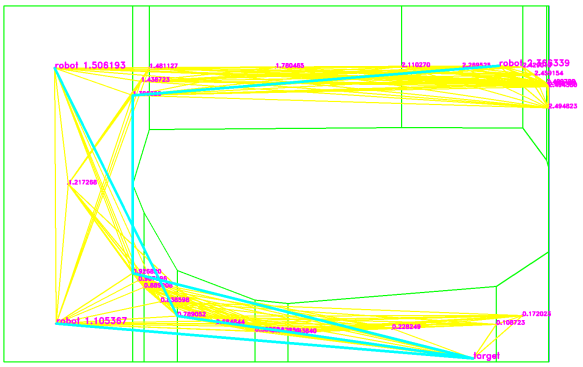

Fig. 23. Initial roadmap with highlighted the best path for each robot

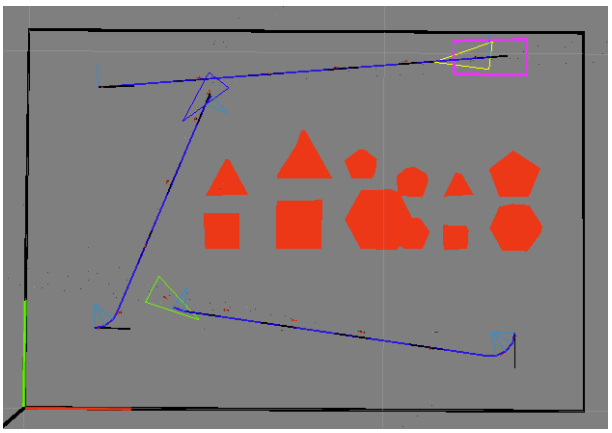

Fig. 24. Evolution of the robot positions after the first step

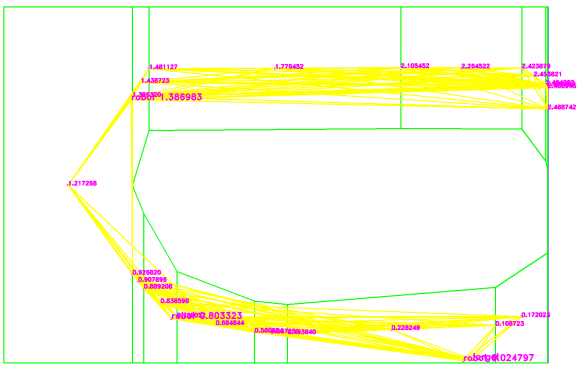

Fig. 25. Evolution of the roadmap after the first step

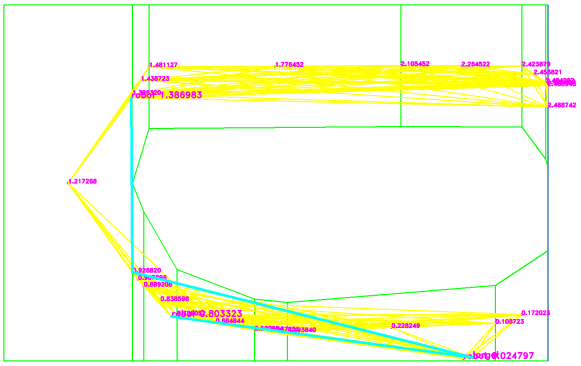

Fig. 26. Roadmap with highlighted the best path for the remaining robots that have not reached the gate after the first step

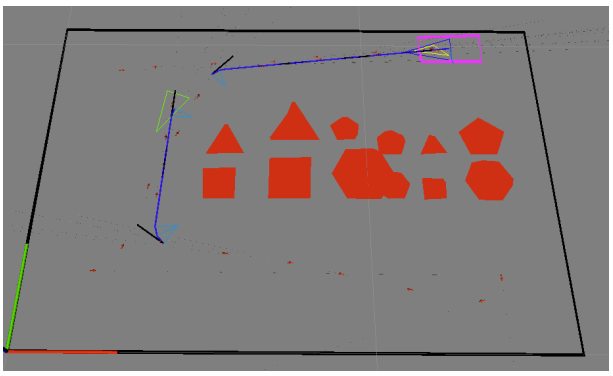

Fig. 27. Evolution of the robot positions after the second step

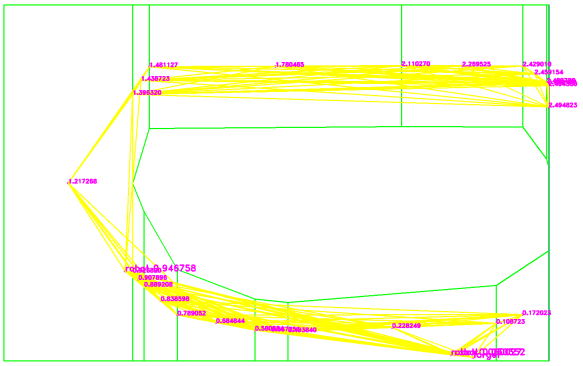

Fig. 28. Evolution of the roadmap after the second step

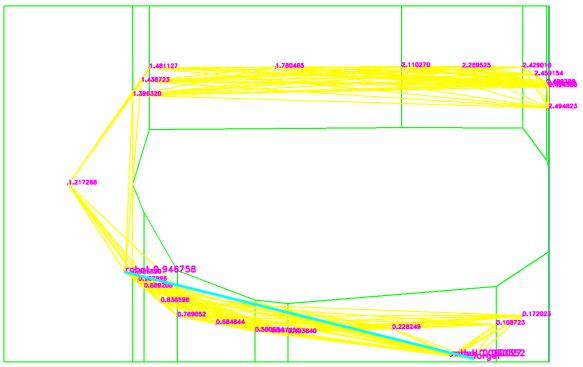

Fig. 29. Roadmap with highlighted the best path for the remaining robot that has not reached the gate after the second step

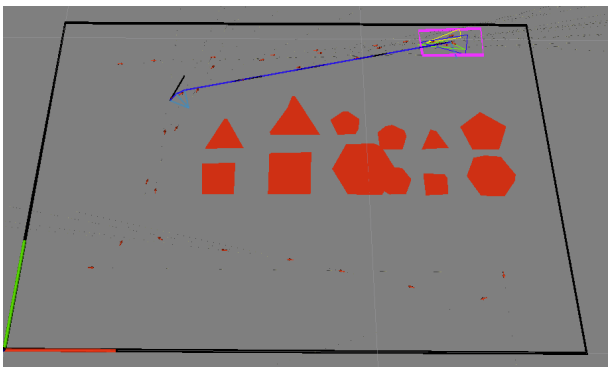

Fig. 30. Evolution of the robot positions after the third step

C. *Test4*

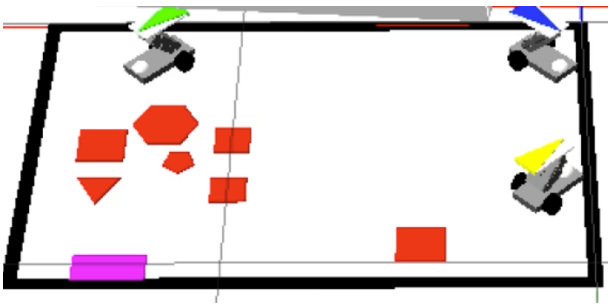

Fig. 31. Initial map configuration

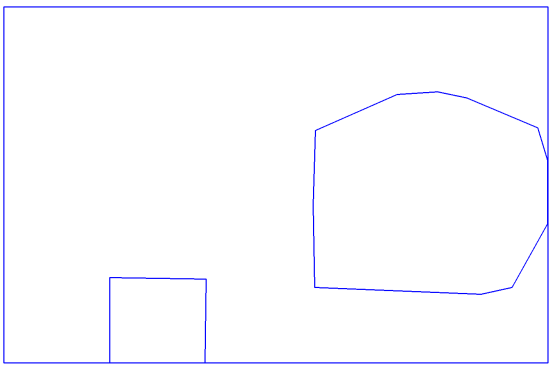

Fig. 32. Inflated obstacles

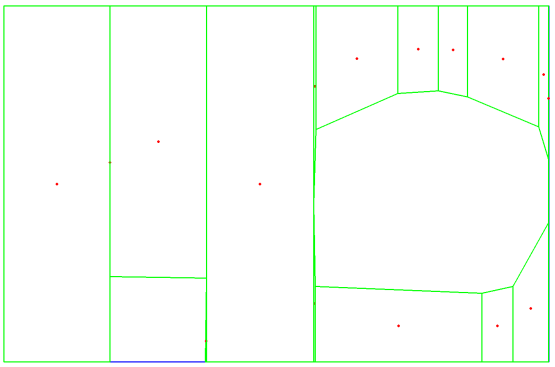

Fig. 33. Resulting cells generated by the VCD algorithm

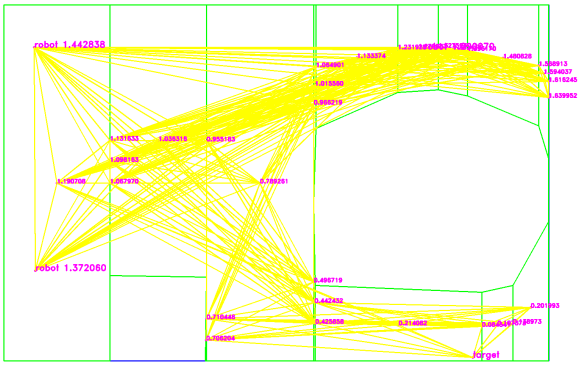

Fig. 34. Initial roadmap with links and cost for each node

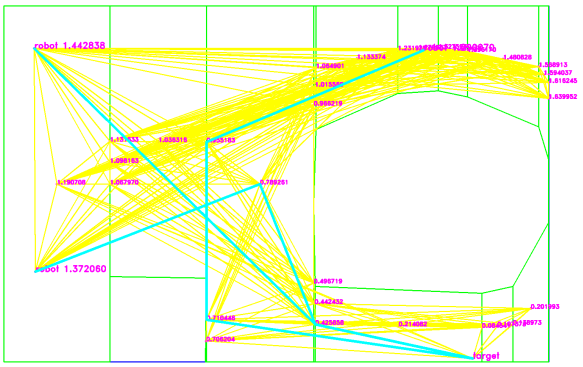

Fig. 35. Initial roadmap with highlighted the best path for each robot

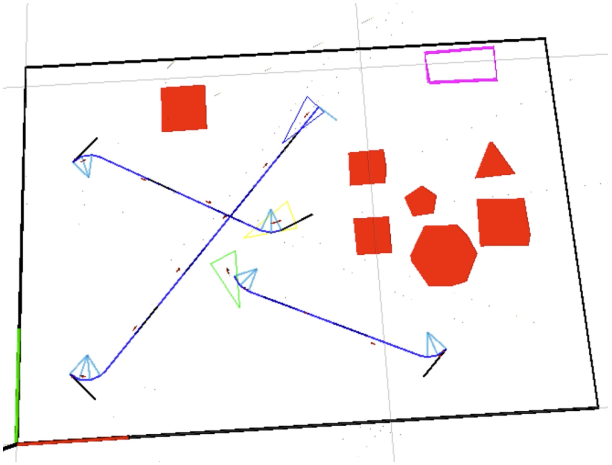

Fig. 36. Evolution of the robot positions after the first step

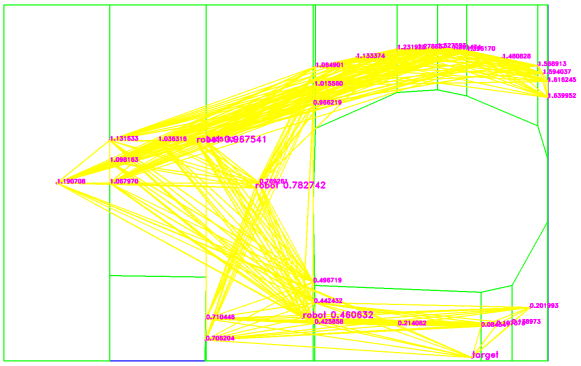

Fig. 37. Evolution of the roadmap after the first step

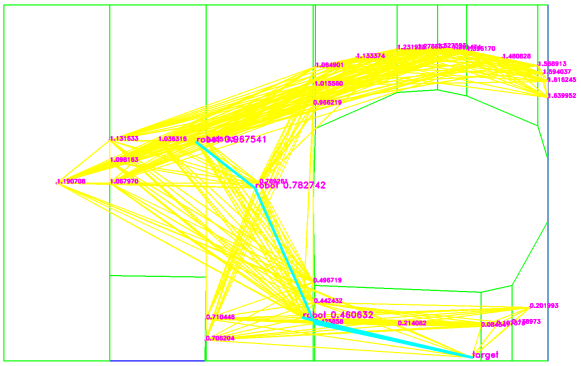

Fig. 38. Roadmap with highlighted the best path for each robot

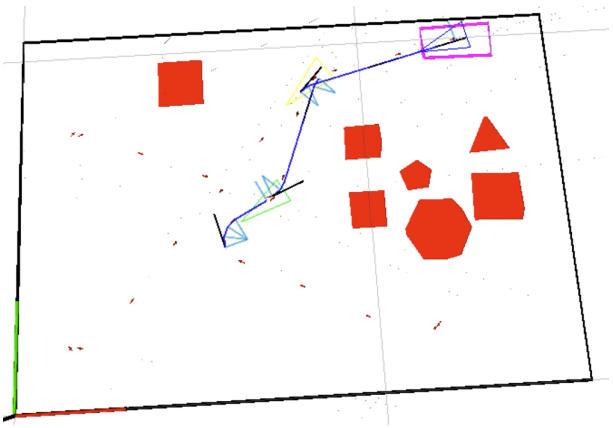

Fig. 39. Evolution of the robot positions after the second step

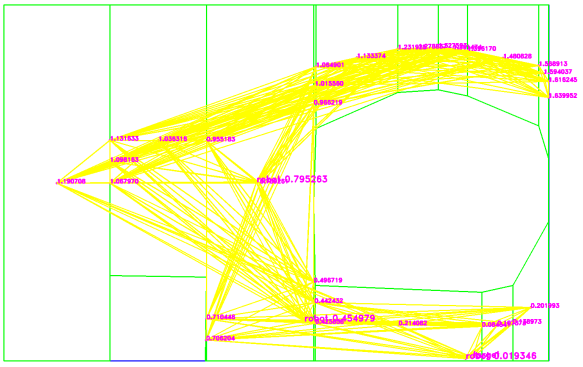

Fig. 40. Evolution of the roadmap after the second step

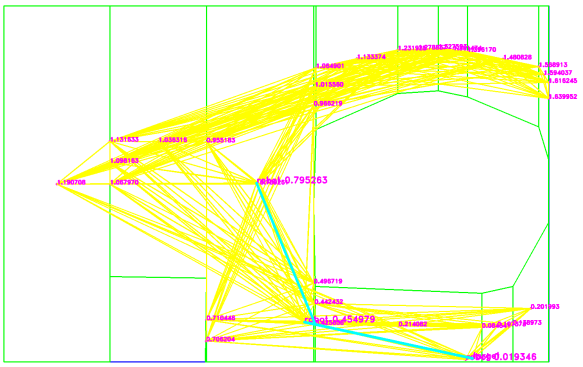

Fig. 41. Roadmap with highlighted the best path for the remaining robots that have not reached the gate after the second step

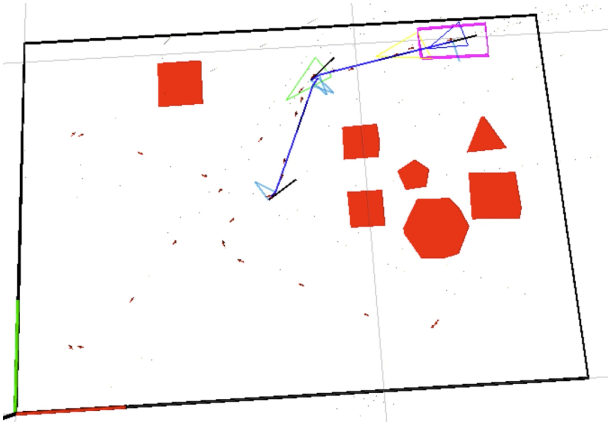

Fig. 42. Evolution of the robot positions after the third step

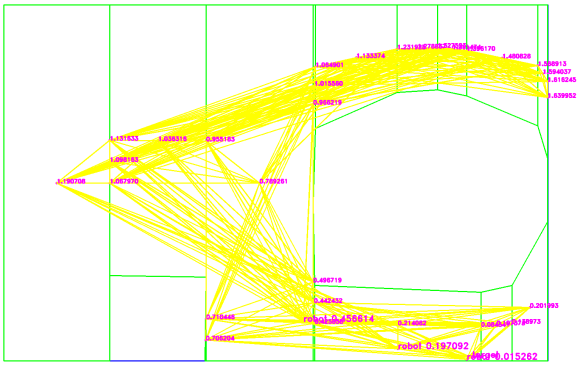

Fig. 43. Evolution of the roadmap after the third step

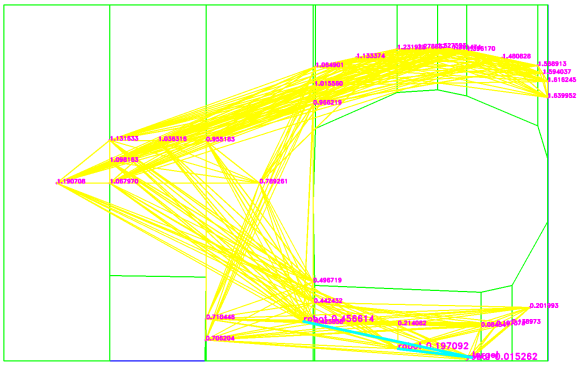

Fig. 44. Roadmap with highlighted the best path for the remaining robot that has not reached the gate after the third step

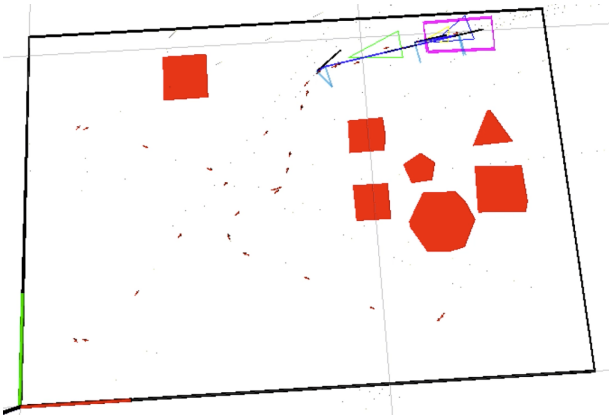

Fig. 45. Evolution of the robot positions after the fourth step