University of Trento
Department of Information Engineering and Computer Science

UNIVERSITY OF TRENTO

Robot Planning and its Application

Coordinated evacuation

Professor:
Prof. Luigi Palopoli

Students:

Nicola Farina	nicola.farina@studenti.unitn.it	229296
Luca Zardini	luca.zardini@studenti.unitn.it	229366

The code is available at:
`https://github.com/nicola-farina/roboplan`

Academic Year 2022-2023

0.1 Introduction

The objective of the project is to design and develop a strategy for a coordinated evacuation of three robots within an environment containing obstacles. More in detail, the goal is for the three robots to reach the exit gate in the shortest possible time, ensuring there are no collisions between the robots and the obstacles along the way. In order to reach that goal, we focus heavily on *mission planning*, which involves environment perception, environment modeling and shortest path generation; and *motion planning*, which takes care of finding a strategy to move the robots towards their destination while avoiding collisions. In Section 0.2 the environment used is presented; Section 0.3 introduces the approach adopted; in Section 0.4 the mission planning is defined, while in Section 0.5 the motion planning is presented. The results are shown and commented in Section 0.6, and Section 0.7 presents some final conclusions and future improvements.

0.2 Environment

The project was developed within a custom *ROS2* environment[1]. The main topics that we used are the following: */map_borders*, */obstacles*, and */gate_position*, which provide information concerning map layout, obstacle placement and gate positions respectively. The robots are actuated thanks to the */follow_path* action, which takes a path as input and automatically takes care of actuating the robot in order to follow it.

0.3 Approach

The approach adopted to solve the project consists in the development of a *ROS2* node that subscribes to the topics cited in Section 0.2 in order to retrieve the environment information. When all the data has been gathered correctly, the obstacles are enlarged in order to treat the robots as a point while avoiding obstacles (0.4). After that, the roadmap is generated using a *visibility graph* (0.4), which is used to find the shortest path to the exit gate for each robot (0.4). Then, each path in the graph is converted into a path that the robots can follow using *multipoint Markov Dubins*. At this point, a coordination algorithm takes care of delaying the actuation of each robot in order to avoid collisions between themselves, if needed (0.5). Finally, each path is sent to the robots.

0.4 Mission planning

The objective of mission planning is to compute the most efficient path for each individual robot. This objective entails the execution of the following steps:

1. Gather information encompassing the map layout, the robot's initial position, and the positioning of obstacles;
2. Extend the boundaries of obstacles to accommodate the robot's dimensions and avoid collisions;
3. Create a graph representing the map layout, often referred to as the roadmap;
4. Systematically traverse the graph to identify the shortest route for each robot.

Obstacle offset

In order to expand the obstacles, we used the *clipper* library[2], which provides several utilities to perform geometric operations on 2D polygons. The main functions that we used take care of offsetting and joining.

[1] https://github.com/pla10/Shelfino_ROS2
[2] https://sourceforge.net/projects/polyclipping/

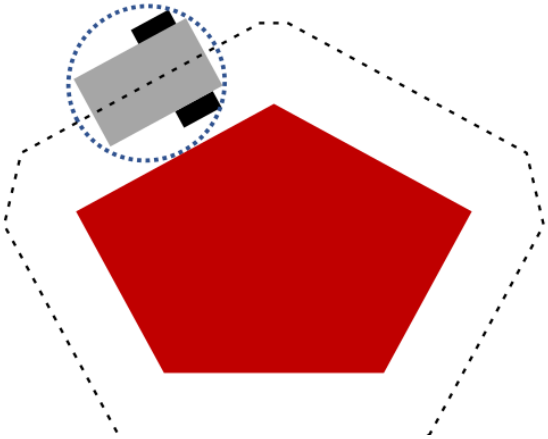

Figure 1: Example of polygon offsetting in order to take into account the robot size.

The obstacles use a positive offset for enlargement, while the map uses a negative offset. Furthermore, we use two different offsets: a first offset, equal to half the size of the robots, is used for collision detection purposes, as illustrated in Figure 1; and a second, slightly larger offset is used to accomodate potential deviations in the robot's trajectory through narrow passages, increasing safety at the cost of diminishing the available free space within the environment. In the scope of this project, the offsetting function emerged as a key player, serving to magnify the obstacles. Additionally, the joining function is used to merge obstacles that happen to touch each other (either before or after offsetting) into a single obstacle.

Roadmap generation

The roadmap is generated by making use of a *visibility graph*. This graph captures essential locations within the environment as vertices, which, in the context of this project, include obstacle and map vertices, robot positions, and gate location. Edges are established between these vertices based on unobstructed lines of sight, indicating that no obstacles impede the direct path between the paired vertices. This concept is visually depicted in Figure 2. By previously enlarging the obstacles, we ensure that the roadmap points already account for the robots' dimensions, effectively preventing collisions. During the roadmap generation process, the presence of concave obstacles was addressed through the utilization of Convex Hulls. A Convex Hull represents the smallest convex polygon that can enclose a set of points (the concave obstacles). The Convex Hull for each obstacle has been calculated, and, if the number of vertices in the Convex Hull did not align with the original obstacle, the additional edges have been inserted into the graph. This approach ensured that the roadmap considers also these edges.

The algorithm that we use to compute the visibility graph is the naive version and has a complexity of $O(n^3)$, where n is the number of nodes within the graph. This complexity arises due to the need to examine each pair of nodes ($O(n^2)$ and verify the presence of obstacle edges intersecting the line segment ($O(n)$). While there exists a possibility to optimize the algorithm's complexity to O(n^2), this optimization has not been incorporated within the scope of this project.

The selection of this algorithm over other combinatorial methods, such as *vertical cell decomposition*, *triangular cell decomposition*, *maximum clearance roadmap*, or sampling-based techniques like *Rapidly Exploring Random Tree (RTT*)*, is driven by its ability to identify obstacles and naturally identify the shortest path. This algorithm stands apart by not necessitating the partitioning of the map into predefined polygons or adopting a probabilistic approach, which can be challenging to replicate consistently over time.

A notable disadvantage of our approach is that it is reliant on robot, obstacles and gate positions. This means that any alteration of them entails the recomputation of the entire graph, which poses a computational challenge. This is not a problem, for example, for the cell decomposition approach. As such, the same

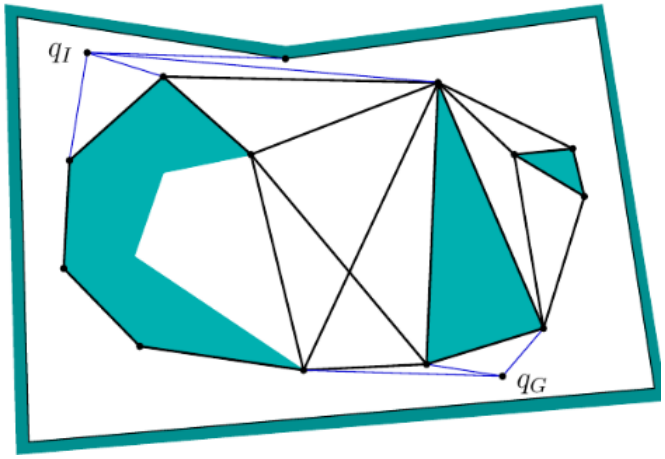

Figure 2: Visibility graph example.

roadmap can be repeatedly applied whenever there are changes to exits or robot positions, as long as the environment remains unaltered. Conversely, the visibility graph approach encompasses robots and gate positions as nodes.

In this project's context, given the assumption that the environment is static and closed, we deemed the visibility graph a good approach.

Shortest path

The classic Dijkstra algorithm has been applied to determine the shortest path. This algorithm allows graph exploration and, when provided with a starting point and a destination, identifies the optimal route with a time complexity of $O(E\,logV)$, where E is the number of edges and V the number of vertices in the graph. The weight associated to each arc is the euclidean distance.

0.5 Motion planning

The path generated through the visibility graph is designed to be independent of any specific robot, having been constructed based on graph points. To operationalize this path for robot movement within the environment, validation is essential, and a dedicated robot path needs to be established.

To identify the shortest curves between origin and destination points, the Markov-Dubins path, as detailed in section 0.5, has been employed. After the generation of Dubins curves for each robot, a strategic assessment becomes necessary to avoid collisions. This assessment focuses on identifying instances where multiple robots may simultaneously occupy the same position within the map. Such scenarios should be preemptively addressed in order to prevent collisions among the robots.

Multipoint Dubins

The Dubins curve represents an optimal path that a robot at position (x_i, y_i, th_i) can follow to arrive at position (x_f, y_f, th_f), with a bound on the maximum path curvature k_{max} and with the assumption that it can only move forward. A Dubins curve is always composed of three arcs, offering the options of turning left, right, or proceeding straight. The configurations available for the Dubins curve are LSL, LSR, RSL, RSR, LRL, and RLR (such configurations can include arcs with length zero).

The Multipoint Markov-Dubins challenge involves determining the optimal sequence of Dubins curves for a given list of points. This intricate task has been tackled through an iterative dynamic programming method, proposed by Frego et al. [**?**]. In this approach, a predefined set of possible angles is employed to identify the most suitable curves. The method then progresses by selecting the best angle and corresponding

curve for each intermediate point. The key constraints are the initial and final angles, while the remaining angles are selected from the predefined set. It's important to note that increasing the number of angles raises the computational complexity of the algorithm.

This dynamic programming approach has a time complexity of $O(nk^2)$, where n is the number of points and k is the number of predefined angles.

Avoiding robot collisions

After deriving the optimal set of Dubins curves for each robot, the next step involves ensuring that these robots do not collide with each other during their movement. The algorithm implemented works under the following assumptions: a robot is no longer considered to be within the environment once it reaches the exit; each robot moves with uniform speed at the same velocity. Given that the robots in the environment are represented as points, the list of points of possible collision detection are identified by examining the intersections of circles centered on each robot point and with radii corresponding to the robot size. The initial step of the implemented algorithm involves interpolating the paths of the robots into a finely detailed list of points that each robot passes through. Subsequently, this list is used to calculate all potential intersections between the three Dubins paths, as determined by the interpolated points.

Following this, based on the predefined assumption of constant velocity, it is possible to compute, for each robot, at which time it occupies a specific location. The algorithm identifies firstly the collisions occurring before the first and second robots complete their respective paths. Whenever a collision is detected, the second robot is assigned a waiting period of one second (the approximate time it takes for one robot to travel a distance equal to its size), that is taken into account to check the other collisions. Every time a new collision is identified, the complete collision path between two robots is recomputed, with new one-second waiting periods assigned to the second robot if needed. Once no collisions are detected between the first and second robots, the same algorithm is applied to the first and third robots, as well as to the second and third robots. Moreover, when a new collision is found, the paths of the other robots must be re-evaluated, as the waiting period of one robot can potentially lead to a new collision with another. When, across all combinations of the robots, no collisions are identified, the algorithm calculates the amount of time each robot must wait before commencing its movement.

In this algorithm, the first robot starts its task immediately without any delays, while the second and third robots might experience some waiting periods. The most challenging situation occurs when the second robot has to wait for the first one to start and the third robot has to wait for the second one to commence.

Finally, to actually start the simulation, the path of each robot has been published on the *action-server* after the robot waiting time is computed. The simulation concludes once all robots successfully reach their designated destinations.

0.6 Results

In this section, three scenarios are presented. In the first one, the map is not convex and it is actively used for the robots in order to reach the gate; in the second one, a concave obstacle occupies a relevant part of the map and in the third a lot of convex obstacles are present in the map, forcing the robots to avoid them to reach the gate.

As shown in Fig 3, the two upper robots have no red obstacles that occlude their path. Without adding the map borders in the visibility graph, the robots are forced to pass near an object edge if a straight line is not found between the robot and the gate. In this case, it is possible to notice that the first and the two robots pass only near the map vertex, while the third robot passes near the obstacle and the map vertices in order to reach the gate.

Fig 4 shows how the robots behave when a concave obstacle is present in the map. In this case, it is necessary to consider the possibility that a free edge (i.e., an edge without intersection with other obstacles

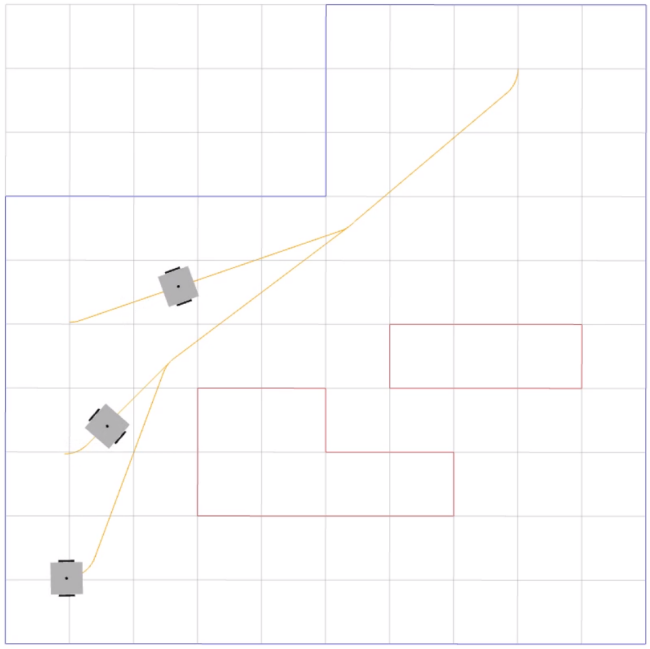

Figure 3: Example of robots simulation. The border contours are colored in blue, the obstacles are colored in red and the robots plans are colored in orange.

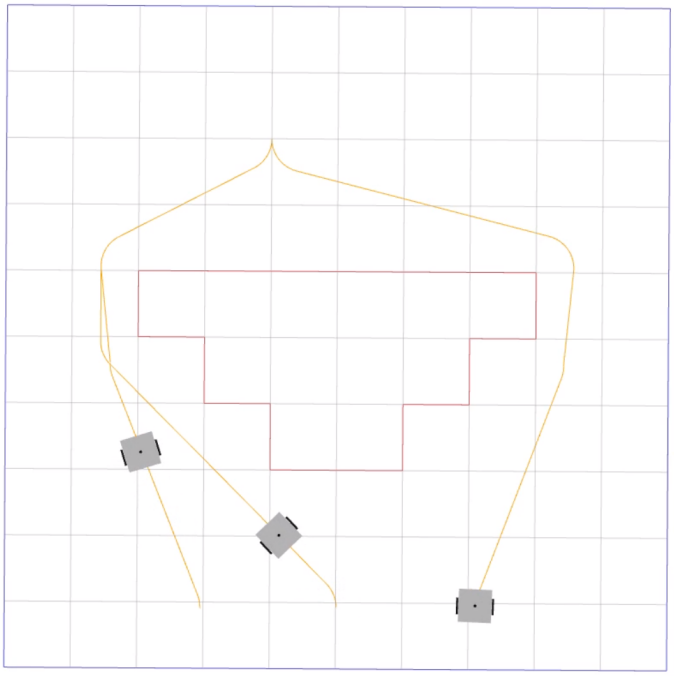

Figure 4: Example of robot simulation. The border contours are colored in blue, the obstacles are colored in red and the robots plans are colored in orange.

edges) can be found also among vertices in the same obstacle. It is possible to see that the first and the third robots do not need to traverse these edges, since they are well positioned and only two vertices are necessary, while the second robot takes advantage of these edges among the same obstacles that allow it to reach the gate earlier.

Fig 5 shows the path found for each robot in presence of three obstacles. In this case, the three robots are forced to avoid the first obstacles and adjust their curvature in order to correctly reach the gate.

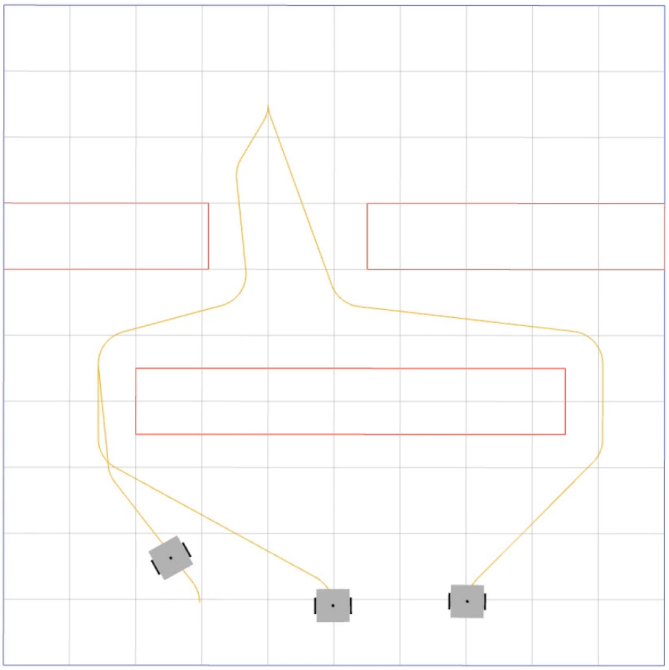

Figure 5: Example of robot simulation. The border contours are colored in blue, the obstacles are colored in red and the robots plans are colored in orange.

0.7 Conclusion

This project proposes an approach to solve the problem of coordinated evacuation of three robots. This approach operates under a set of key assumptions: each robot travels at a uniform velocity, and that the environment is static and closed.

The initial phase involves the collection of information pertaining to the map layout, robot locations, and obstacle positions. Subsequently, the obstacles are expanded in size to account for the dimensions of the robots. These modified obstacle positions, along with the vertices of the map, the robot positions, and the location of the exit gate, are used to build a visibility graph.

Employing a Dijkstra algorithm for exploration, the shortest path for each robot is computed. This path then serves as the basis for generating a sequence of Dubins curves.

Finally, the algorithm also mitigates potential collisions between robots by identifying intersections in their paths and subsequently introducing a delay in the departure of one robot. This collision avoidance strategy ensures a safe execution of the evacuation plan.

0.8 Possible improvements

A more robust and safety-oriented strategy, capable to reduce the possibility of collisions among robots in presence of external forces or different robot velocities, involves the application of the developed collision avoidance algorithm to every pair of points within the visibility graph. This approach entails waiting until all the robots have reached a node within the visibility graph. This synchronous behavior enhances the management of robot safety, ensuring that they progress with minimized risk of collision. However, it's worth noting that this approach may introduce some delays in the process of evacuating the environment.

The visibility graph implemented has a complexity of $O(n^3)$. Although there are readily available implementations of more efficient algorithms with $O(n^2)$ complexity online, they have intentionally not been utilized for a specific purpose. The decision was made to understand its challenges. Moreover, this approach allowed us to examine the algorithm's limitations and the development of effective strategies to overcome them.

While our approach is applicable for simulations, it might not be as safe for real-time scenarios, where external disturbances to the robots are frequent (e.g. bumps in terrain, inaccurate actuation). For such scenarios, a reactive approach is preferable (it could involve re-computing the plan at several steps while reaching the goal).

References

[1] Marco Frego, Paolo Bevilacqua, Enrico Saccon, Luigi Palopoli, and Daniele Fontanelli. An iterative dynamic programming approach to the multipoint markov-dubins problem. *IEEE Robotics and Automation Letters*, 5(2):2483–2490, 2020. doi: 10.1109/LRA.2020.2972787.