4TM00 Robot Motion Planning and Control
Assignment 2 - Search-Based Path Planning & Safe Path Following

Group 5
Devan Sedmak[1](2234238) and Matteo Petris[2](2234203)

***Abstract*— In this assignment we develop two safety mechanisms for the RoboCyl robot: a teleoperation system that modifies user commands to avoid collisions and a reactive navigation algorithm for obstacle avoidance using ROS2. Both approaches were evaluated in dynamic environments, with limitations and improvements discussed.**

I. Safe Navigation Costmap

A. Introduction

In the first part of the assignment we design a ROS node that receives an occupancy grid map and returns a safe navigation costmap for safe reference path planning. We use the environment map to ensure safety around obstacles.

B. Design

The robot is inside a closed rectangular environment with some obstalces, bounded by walls. The environment is represented by the given occupancy grid map. The grid map discretizes the workspace into a grid where each cell represents a portion of the environment.

The occupancy grid map is initialized as a 2D array with dimensions 300 × 300, with:

- $width$: 6 meters (discretized into 300 cells along the x-axis).
- $height$: 6 meters (discretized into 300 cells along the y-axis).
- $resolution$: Each grid cell has a size of 0.02m × 0.02 m.
- $origin$: The bottom-left corner of the map is set to world coordinates $(-3, -3)$.

To convert between the grid and world coordinate systems, we implement two transformations:

1) World-to-Grid: The transformation from world coordinates (x_1, x_2) to grid indices (i, j):
$$i = \left\lfloor \frac{x_2 - (-3)}{0.02} \right\rfloor, \quad j = \left\lfloor \frac{x_1 - (-3)}{0.02} \right\rfloor$$
2) Grid-to-World: The transformation from grid indices (i, j) to world coordinates (x_1, x_2):
$$x_1 = j \cdot 0.02 - 3 + 0.5 \cdot 0.02, \quad x_2 = i \cdot 0.02 - 3 + 0.5 \cdot 0.02$$

Each cell's value is assigned as follows:

- 0: The cell is free.
- -1: The status of the cell is unknown.
- 100: The cell is occupied.
- A value in $(0, 100)$: The cell is uncertain, and the value is equal to the probability of the cell being occupied.

We now construct the binary occupancy grid map B. This matrix is created from the occupancy grid map definig the cell with value less then a treshold (in our case 1) free (False=0) else occupied (True=1).

Now we construct the map M, that is the complementary map of B:
$$M(i,j) = 1 - B(i,j) = \begin{cases} 0 \text{ (occupied)} & \text{if } B(i,j) = 1 \\ 1 \text{ (free)} & \text{if } B(i,j) = 0 \end{cases}$$
Using M we can define the distance map DM as:
$$DM(i,j) = \min_{\substack{(u,v) \\ M(u,v)=0}} \sqrt{(i-u)^2 + (j-v)^2}.$$
Now we impose the $safety\_margin$ equal to the radius of the robot $\rho = 0.2$ and we scale it with the resolution:
$$safety\_margin\_in\_cells = \frac{safety\_margin}{resolution}$$
We correct the distance map DM by the $safety\_margin\_in\_cells$:
$$DM'(i,j) = \max\big(DM(i,j) - safety\_margin\_in\_cells,\, 0\big).$$
Finally we can construct the costmap, called $cost$. It is computed from the distance map DM':
$$cost(i,j) = (maxcost - mincost)\, f_{\text{decay}}\big(DM'(i,j)\big)\big) + mincost$$
where $f_{\text{decay}} : [0, \infty) \to [0, 1]$ is the exponential decay function defined as:
$$f_{\text{decay}}(x) = e^{-kx}$$
In particular we select: $k = 0.6$, $maxcost = 90$, $mincost = 1$.

Mincost and max cost describe the range in which the cost can vary. These numbers are arbitrary; we left them as found in the assignment initial code. An occupied cell will have the maxcost.

C. Limitations

Computing the costmap is computationally expensive. That is because we need to find for each free cell its minimum distance from an occupied cell. Because of that, we had some issues with the implementation. In particular, the path planner was searching for a path, but the costmap was not ready yet. We solve it by adding an exception handler.

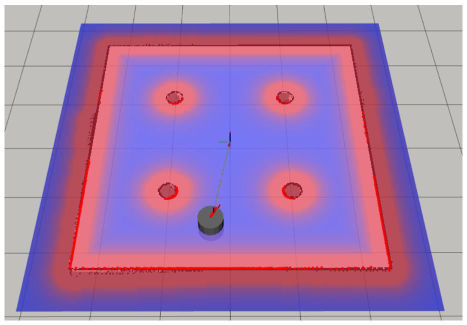


Fig. 1: The cost map

D. *Conclusion*

To improve the costmap, we could integrate the laser scan data to update the costmap if some unknown obstacles are detected, if the environment would have been dynamic.

II. Search-Based Path Planning

A. *Introduction*

In the second part of the assignment, we will design a ROS node that receives a goal position and generates a safe navigation path over an occupancy grid map. The path should guide the robot toward the goal while avoiding obstacles and minimizing navigation cost. We will be using the robot's pose, costmap and goal position. It is important to note that it may not always be possible to reach all given goal positions in a complex environment, and executing reference paths that are too close to obstacles can be challenging. Therefore, the objective is to find a reference path with adequate clearance from obstacles; otherwise we will indicate that no safe navigation path exists.

B. *Design*

Firstly we need to define what is a path. A path is a representation of movement strategy. Here we define a path as a sequence of adjacent free grid cells.

To plan the path, we must decide on the connectivity of adjacent grid cells first. In this assignment, we use 8-connected neighbors. This means that each cell has 8 neighbors, including diagonal directions. This allows to more flexible movement across the grid.

Now we construct a connected graph $G = (V, E)$. V is the set of nodes corresponding to the grid cells. E is the set of edges between these nodes. An edge (A, B) exists only if the cost of the cell A or B is lower than the selected $maxcost'$.

This $maxcost'$ is a threshold. If $maxcost' \in [mincost, maxcost)$ then a cell with a cost $\in [maxcost', maxcost]$ will not be connected to the graph.

If $maxcost' > maxcost$ then every cell with will be connected in the graph, even an occupied cell. Using an appropriate $maxcost'$ will ensure that, when planning the path, only safe edges are considered. If the edge is safe, then we assign a weight to it equal to the travel cost.

The travel cost between two grid cells determines the cost of moving from one cell to another. For adjacent cells, the cost is calculated as:

$$\text{TravelCost}(A, B) = \text{dist}(A, B) \times \frac{\text{cost}(A) + \text{cost}(B)}{2}$$

where:

- $\text{dist}(A, B)$: Distance between two cells, computed as Euclidean. This means that the distance between 2 cells in the vertical or horizontal direction is 1. Meanwhile the distance between 2 cells in the diagonal direction is $\sqrt{2}$,
- $\text{cost}(A)$: Cost of cell A calculated as in first part of the assignment.

To compute the optimal path from the starting position to the goal while minimizing the total navigation cost, we implement Dijkstra's algorithm. This algorithm explores the graph and computes the travel cost iteratively until the optimal path is found. We propose an overview of the algorithm:

1) Initialization:
   - Set the travel cost of all nodes to infinity except the start node, which is initialized to zero.
   - Add the start node to the frontier set (nodes to explore).
2) Iterative Expansion:
   - Select the node with the lowest travel cost from the frontier set.
   - For each neighbor of the selected node:
     – Compute the estimated travel cost via the current node.
     – Update the cost if a lower travel cost is found.
     – Add the neighbor to the frontier set if it hasn't been explored.
3) Termination:
   - The algorithm terminates when the goal node is reached, and the optimal path is reconstructed by backtracking.

So if a path exists, the algorithm outputs the sequence of grid cells leading to the goal with adequate clearance from obstacles. This sequence of cells is then converted into the world cordinate system using the grid-to-world transformation.

So we obtain a discrete path:

$$p : \{1, \ldots, m\} \to \mathbb{R}^2$$

where m is the number of points in the path.

A point in the path is now defined as $p(\tau)$ where τ is the index of the point in the path array.

If no path is found, then algorithm indicates that no safe navigation path exists, like shown in the figure 4.

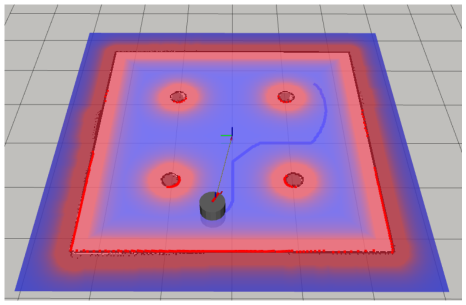


Fig. 2: The path p found by Dijkstra's algorithm from the starting position to the goal

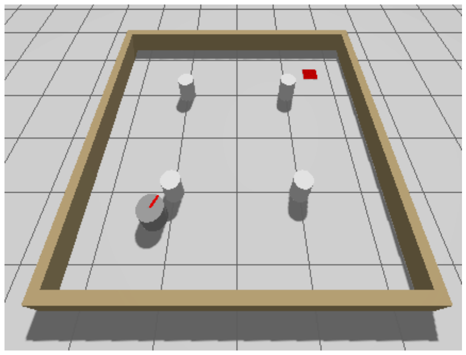
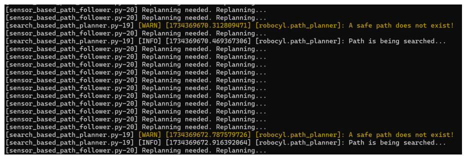


Fig. 3: Example of using a $maxcost' < maxcost$ to create a case when is impossible to find a path


```
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[search_based_path_planner.py-19] [WARN] [1734369670.312809471] [robocyl.path_planner]: A safe path does not exist!
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[search_based_path_planner.py-19] [INFO] [1734369670.469367306] [robocyl.path_planner]: Path is being searched...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
[search_based_path_planner.py-19] [WARN] [1734369672.787579726] [robocyl.path_planner]: A safe path does not exist!
[search_based_path_planner.py-19] [INFO] [1734369672.916392064] [robocyl.path_planner]: Path is being searched...
[sensor_based_path_follower.py-20] Replanning needed. Replanning...
```
Fig. 4: The orange lines show that a safe path does not exist

C. *Limitations*

Our design is computationally expensive, since it takes a lot of time to plan a path. We could improve the speed of computation by using the A* algorithm. It leverages a lower-bound estimate of the travel cost, calculated as Heuristic$(A, B) = \text{dist}(A, B)$.

D. *Conclusion*

By combining the costmap and Dijkstra's algorithm, we ensure that the navigation path is both safe and optimal. The path is considered safe because any occupied cell or a cell within a distance less than or equal to the safety radius from an obstacle is marked as unreachable. The path is optimal because it is found using Dijkstra's algorithm, that finds an optimal solution.

III. Sensor-Based Path Following

A. *Introduction*

In the third part of the assignment, we will design a ROS node that receives a reference path toward a global goal position and generates command velocities to safely follow it while avoiding sensed obstacles. We will be using the robot's scan measurements, pose, goal pose and the reference path. It is important to note that it may not be possible to safely follow all reference paths from start to end in complex environments. Therefore, the objective is to follow the path as much as possible. If the path cannot be followed safely, we will indicate that replanning is needed and we will find a new path.

B. *Design*

The robot in use is RoboCyl. It is a fully-actuated velocity-controlled robot with a circular body shape of radius $\rho = 0.2\,\text{m}$. The model is described are as follows:

- State Variable

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{(2D Position)}$$

- Equation of Motion

$$\dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt} = \mathbf{u} \quad \text{(Velocity Control)}$$

The robot can only detect some points on the surface of these obstacles. These points will be called point obstacles. The point obstacles and the distance to a point obstacle with respect to the robot's position x are defined as follows:

- Point obstacles $\quad$ ($q_i \in \mathbb{R}^2$: Point obstacle position)

$$\mathbf{Q} = (q_1, \ldots, q_l) \in \mathbb{R}^{m \times 2}$$

where l is the number of detected points by the scan.
- Distance to point obstacle q_i

$$d_i(\mathbf{x}) = \|\mathbf{x} - q_i\|$$

We use the defined point obstacles and distance to point obstacle to construct a the Local Safe Corridor. We use the Local Nearest Points N_Q around the robot's position as we did in the first assignment.

After that we need to define the Local Free Space:

$$LF_Q(x) = \left\{ \mathbf{y} \in \mathbb{R}^2 \mid B(\mathbf{y}, \rho) \subseteq SC_Q(x) \right\}$$

and we construct it as in the first assignment.

Now we can find the last point of the path that is inside $LF_Q(x)$:

$$\tau^* = \max_{\substack{\tau \in [0,m] \\ s.t. \\ p(\tau) \in LF_Q(x)}} \tau$$

We have the following cases:

- If τ^* doesn't exist then this implies that there are no points of the path inside the Local Free Space.
- If $\tau^* = m$ then $x_p = p(\tau^*)$ and this correspondes with the goal.
- If $\tau^* < m$ then x_p will be found in the following way

$$d_s = d(p(\tau^*), \partial LF_Q) = \min_{y \in \partial LF_Q} \|p(\tau^*) - y\|$$

$$d_e = d(p(\tau^* + 1), \partial LF_Q) = \min_{y \in \partial LF_Q} \|p(\tau^* + 1) - y\|$$

$$\alpha = -d_e/(d_s - d_e)$$

$$x_p = \alpha * p(\tau^*) + (1 - \alpha) * p(\tau^* + 1)$$

where τ^* is the maximum index of the path within the Local Free Space.

Now, we can design a path-following strategy based on potential fields. The key idea is to treat the projected path goal x_p as the center of an attractive field, where the potential reaches its minimum (equal to 0). At the same time, obstacles are modeled as centers of repulsive fields, where their influence diminishes to zero at their respective centers. The attractive field is constructed using the gradient of the attractive potential, which is defined by the Squared Euclidean Distance. This leads to the following formulation:

$$V_{\text{att}}(x) = ||x - x_p||^2$$

$$\nabla V_{\text{att}}(x) = 2(x - x_p)$$

The repulsive field is constructed using the gradient of the Bounded repulsive potential. The repulsive potential and repulsive gradient are respectevely:

$$V_{\text{rep}}(x) = \frac{1}{\prod_{i=1}^{m} s_{k_i}(d_i(x))}$$

$$\nabla V_{\text{rep}}(x) = -V_{\text{rep}}(x) \sum_{i=1}^{|N_Q|} \frac{s'_{k_i}(d_i(x))}{s_{k_i}(d_i(x))} \nabla d_i(x)$$

where $s_k(x)$ is the exponential sigmoid, so we have:

$$s_k(x) = tanh(kx)$$

$$s_k(x)' = k(1 - tanh(kx)^2)$$

After some experimentation we choose the threshold decay rate $k = 3$. We combine the repulsive and attractive potencial using the Multiplicative Navigation Potencial:

$$V(x) = V_{\text{att}}(x) * V_{\text{rep}}(x)$$

$$\nabla V(x) = \nabla V_{\text{att}}(x) * V_{\text{rep}}(x) + V_{\text{att}}(x) * \nabla V_{\text{rep}}(x)$$

Now we need to transform ∇V in $\nabla V'$, referring to the orientation of the robot in the environment, as we did in the first assignment. Finally, we must choose an appropriate control gain g and set the control input u equal to:

$$u = -g \nabla V'$$

After experimentation, we selected $g = 0.3$.

If the robot comes to a position where there is no intersection between the path and the Local Free Space, then it stops ($u = 0$) and it replans a new path as seen in the Fig. 6.

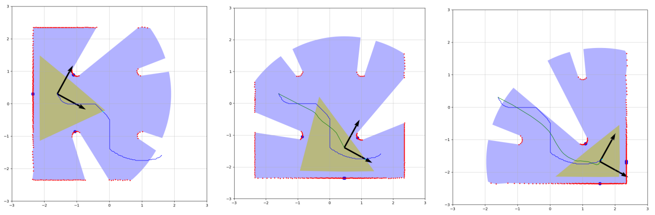


Fig. 5: The movement of the robot from its initial position to the goal, following the designated path

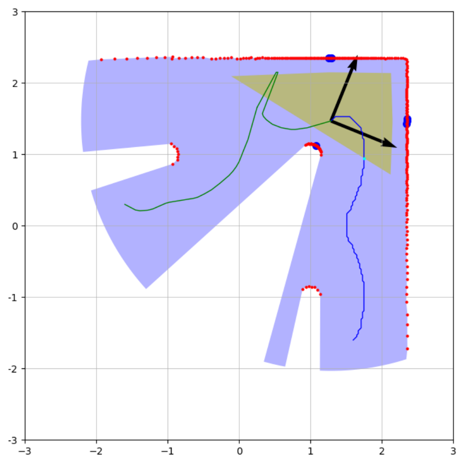


Fig. 6: In this case the robot doesn't find an intersection of the local free space with the path, so it replans a new one

C. Limitations

If we set $maxcost' < maxcost$ then it could happen, that the robot doesn't find a safe path to follow. In this case it will just wait for a goal change or an user input. This limitation could be overcome by either moving the robot a little bit away from the nearest obstacle or projecting the goal to the nearest safe cell.
Another limitation is that the path is constantly being planned, even if we don't need a new one. This is because the Search-Based Path Planner node computes a new path at each rate, even if we don't use it. We could solve this issue by adding a communication topic from the Sensor-Based Path Follower node to the Search-Based Path Planner node, that requests a new path only when it is needed. Or compute a new path only when there is no intersection with the Local Free Space.

D. Conclusion

The robot successfully follows the path and reaches the goal in a short time. However, as we said, there is room for optimization in the computation of the cost map and the path. Specifically, the path computation could be further optimized by performing it only when necessary.

References

[1] Ö. Arslan, Robot Motion Planning and Control (4TM00) - Course material, Eindhoven University of Technology, 2024.