Mobile Robot Control 2024 - Global Navigation Assignment

The goal of this assignment is to:

- Apply the A* algorithm to find the shortest path from the start to the exit of a maze where the node list and connections are provided,
- Create your own Probabilistic Road Map to generate the node list and connections for a given map. This serves as input for the A* algorithm to plan a global path, and
- Connect the global planner to your local planner from the previous assignment.

A framework for the implementation has been provided, in which several parts of the code have been left out. The assignment is to complete the implementation by writing these parts.

Setup

The code for this assignment can be found on the GitLab of your group.

Besides the source code (to be completed by you), it provides the files containing information about the maps that will be used to test your implementation. For each map, these files are:

- PNG file of the map,
- YAML file with metadata for the simulator (e.g., map PNG filename and resolution) ,
- JSON file with configuration settings for the simulator (e.g., the robot's initial pose),
- JSON file with parameters of the map (e.g., node grid locations, connections, and the start and goal locations). Note that the structure of this file is different for the two different planning methods that you will work on in this assignment.

These files can be found in the subfolders `config` and `maps`. Below you can find a description of the assignments, followed by instructions on how to test your code, questions to be answered as a part of this assignment and submission requirements.

Assignment

Divide your group into two subgroups; one subgroup works on the A* implementation, the other on Probabilistic Road Maps (PRM). At last, we want you to combine your **local and global planner**, so that the local planner uses the global path as its guide.

A* implementation

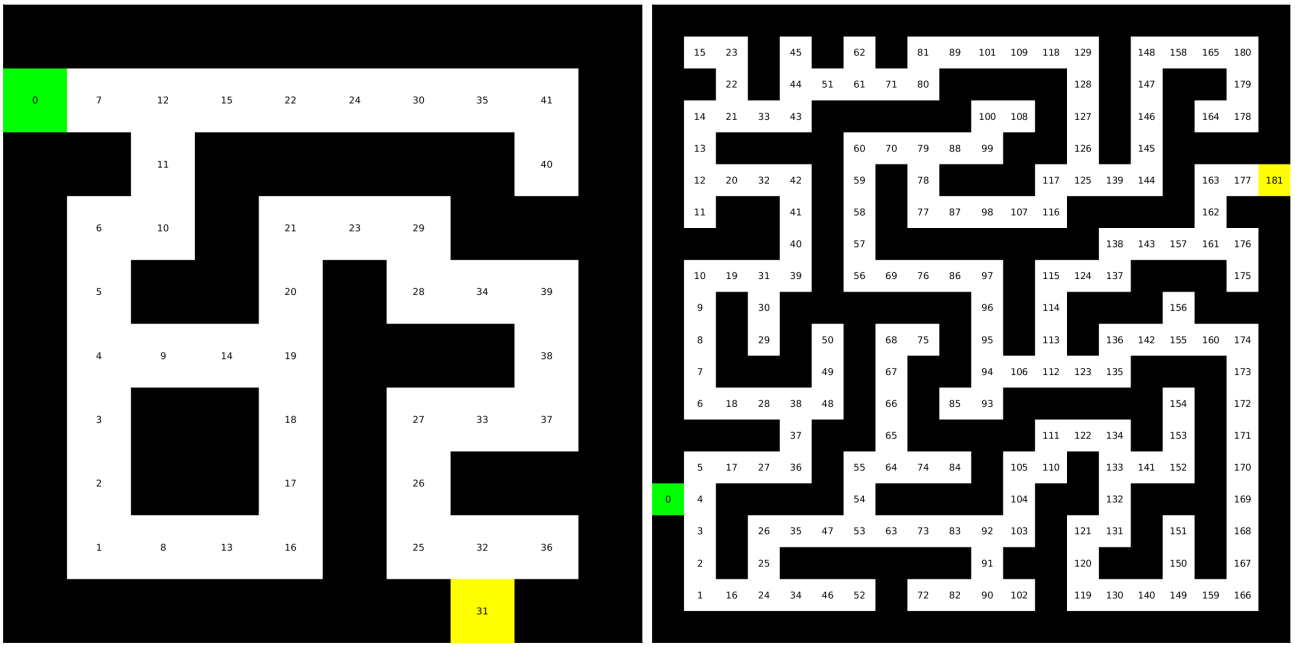
Your job is to write the code in the function `planner::planPath()` in `<path-to-repository>/src/Planner/planner.cpp`. Within this function, to goal is to find the

sequence of node IDs that form the shortest path through the maze from start to exit by applying the A* algorithm. The initialization step and the code's main structure have already been provided. The nodes' coordinates are given in `_nodelist`. The connections per node are given in `_connections`, i.e., `_connections[i]` gives the indices of the nodes that are connected to the node with index `i` (which corresponds to `_nodelist[i]`). Furthermore, `_start_nodeID` and `_goal_nodeID` are the indices of the start and goal nodes.

The exercises/parts to be completed are indicated in the code:

1. The first exercise is to find, in every iteration, the node that should be expanded next.
2. The second exercise is to explore its neighbors and update them if necessary.
3. The last exercise is to trace back the optimal path.
   It should not be necessary to make changes to other files or functions, but you are allowed to do so.

Visual representations of the two maze maps including the node IDs are shown below. All horizontally or vertically adjacent nodes are connected. You can use these for debugging your A* algorithm.



PRM implementation

The task of this subgroup is to finish the code in `<path-to-repository>/src/PRM/prm.cpp`. In this file, the goal is to generate a graph that represents a Probabilistic Road Map (PRM) for a given map. The graph consists of

- `vertices`, where each vertex is a `(x,y)` coordinate, and
- `edges`, where each edge contains two id's of vertices that are connected to each other. This can also be seen in the header file `prm.h`. This graph is used by the A* algorithm to create a list of vertices to visit to get from a start to a goal position.

In the code, three exercises need to be completed:

1. Firstly, the occupied areas of the map (i.e., the walls) need to be inflated in order to account for the size of the robot,
2. The second exercise is to write code that checks whether `new_vertex`, which is a random `(x,y)` coordinate sampled for the free space of the map, is a certain distance away from the existing vertices in the graph.
3. The last exercise is to determine whether an edge between two vertices is valid, i.e.:
   - the two vertices are within a certain distance from each other, and
   - the edge does not go through occupied space of the map.
   Again, it should not be necessary to make changes to other files or functions, but you are allowed to do so.

Combining local and global planning

Have a look at the `main` function from `<path-to-repository>/tests/test_assignment.cpp` to see how the planner is being constructed dependent on A *with a gridmap or A* with PRM, and that it calls `planner.planPath`. This will provide you with an idea how to integrate the global planner into your code of the local planning assignment.

To test the integration of the two, build a map for the simulator and on the field (you can use the `map_compare` (see below). You can determine your current position using the odometry data. During testing, ensure that you put a obstacle (or multiple obstacles) on the global path to enforce the local path to evade the obstacle.

Testing

To run a test for one of the maps, first build your code:

- `cd ‘path-to-repository’/build`
- `cmake ..`
- `make`

If this is successful, run the test:

- Start the simulator with the YAML and JSON files corresponding to the desired map, e.g.: `mrc-sim --map ‘path-to-repository’/maps/metadata_maze_small.yaml --config ‘path-to-repository’/config/config_maze_small.json`
- In a new terminal, open the visualization: `sim-rviz`
- In another terminal: `cd ‘path-to-repository’/bin`
- Run the test for one of the map (use the JSON parameter file as an argument), e.g.: `./assignment ../config/params_maze_small.json`
- If you do not want to show the PRM, add `HIDE_PRM` to the previous command, i.e.: `./assignment ../config/params_maze_small.json HIDE_PRM`

Note that when you want to test PRM code, you can only use the `map_compare`. For A*, you can use `maze_small`, `maze_large` and `map_compare`. In this way, a comparison can be made between their solutions. For `map_compare`, each algorithm has its dedicated `params_map_compare_Y.json` file, where `Y` is `grid` or `PRM`.

Questions

1. How could finding the shortest path through the maze using the A* algorithm be made more efficient by placing the nodes differently? Sketch the small maze with the proposed nodes and the connections between them. Why would this be more efficient?
2. Would implementing PRM in a map like the maze be efficient?
3. What would change if the map suddenly changes (e.g. the map gets updated)?
4. How did you connect the local and global planner?
5. Test the combination of your local and global planner for a longer period of time on the real robot. What do you see that happens in terms of the calculated position of the robot? What is a way to solve this?
6. Run the A* algorithm using the gridmap (with the provided nodelist) and using the PRM. What do you observe? Comment on the advantage of using PRM in an open space.

Submission

Upload your code with comments/documentation on your group's GitLab page on the main branch, meaning that both solutions should work. Additionally, on your group's wiki page you should:

1. Answer the questions above,
2. Upload screen recordings of the simulation results and comment on the behavior of the robot. If the robot does not make it to the end, describe what is going wrong and how you would solve it (if time would allow), and
3. Upload video's of the robot's performance in real life (same as simulation videos), and comment the seen behavior (similar to the previous question).