Documentation

TAS Project

Hannah Baumann

Parameters:

To improve the collision avoidance of the car I read about the DWA local planner and the parameters which can be adjusted (local planner parameters, local costmap parameters, global costmap parameters, common costmap parameter). We tried different values for different parameters to find the optimal setting.
Unfortunately the changes often did not lead to an improvement (or even changes) of the behavior of the car even though it theoretically seemed to be most likely. But we could still achieve an improvement.

The changed parameters are commented in the yaml files in the folder 'parameters'.

Simple Navigation Goal:

This node is basically the already existing node 'simple_navigation_goal'. We changed some of it in order to adjust it to our purpose.
New about the code is the following:

- It is reading in the waypoints out of the file 'Waypoints.txt' which is generated by our program 'Waypoints'.
- It is sending these waypoints to the move_base one by one in a loop whenever the previous waypoint (goal) has been reached reached.

Waypoints:

This program is generating the waypoints for our task in a high automated way and saving them in a text file 'Waypoints.txt' which is then used by the 'simple navigation goal' node.
This is done by finding an optimal way through the lab by using A* search.
The needed input is the map of the lab, the coordinates of the corners of the lab and the orientations of the corridors.

Main.cpp

- Building the graph by setting the walkable variable for each node (pixel of the map) and saving the connection to neighbored nodes
- Create instance from pathfinder and launch the search four times for every corridor (one corner to another corner).
- Get every x-th node (e.g. a node every 2m) of the found path and save them in a text file with their orientation.

PathFinder.cpp, PathFinder.h

- Creating the costmap by using the map of the lap. Changing free spaces into black and occupied (not walkable) spaces into white. Using a blur filter afterwards to create higher costs around obstacles.
- Implementing the A* search from start to goal, using the euclidian distance and the costs from the costmap as a heuristic. Returning the optimal path.

PriorityQueue.h

- Customized stl::priorityqueue()
- Allows the contained nodes to change (higher computational effort, O(n)), because this functionality is required for the A* search (nodes in the queue have to be updated).
- Functions:
    - Push (insert element with specified position (cost))
    - Pop (return element with lowest position (cost))
    - Find (true if element exists in queue)
    - FindPos (true if element exists with higher position (cost) in queue)