Exercise 5: Customizing navigation

Development of Intelligent Systems

2022

In this exercise we will look in further detail some of the previously discussed topics. For some tasks for the mobile robot we need greater precision in the movement of the robot. We can get this precision with customizing the parameters that `move_base` and `amcl` use. For some tasks we have to resort to simple `twist` movement commands. For this exercise you need the `turtlebot_navigation` package and there are no new materials.

1 Customizing `move_base`

The general parameters for `move_base` are set in the `turtlebot_navigation /param /move_base_params.yaml`. Among other things, here you can set:

- The global and local planners which are used.
- Define recovery behaviours. When `move_base` fails to find a valid plan it will perform recovery behaviours in the order they are specified.
- Controller and planner patience. How long the robot will wait for a valid control or a plan before starting recovery behaviours.
- Enable/disable recovery behaviours.
- Enable/disable in place rotation for clearing obstacles.
- Others.

As you are aware, the robot does not always know its precise location. As it moves, if the marking obstacles is enabled, it will sometimes wrongly add obstacles to the map. This can also happen when starting the robot in the wrong location. Note that you can use the service `clear_costmaps` from `move_base` to clear the costmaps if you believe that they have been contaminated.

1.1 Local and global costmaps

As you probably know by now the navigation stack maintains two costmaps. This costmaps are used to plan robot movements by the local and the global planners. When you start `move_base` from the `turtlebot_navigation` package it loads the file `turtlebo_navigation /param/costmap_common_params.yaml` to set some parameters for both the local and global costmaps. You can explore this file to see the parameters that you can set. The actual package that `move_base` uses for maintaining the costmaps is the costmap_2d package. Some common tparameters:

- marking (adding an obstacle in the map) and clearing (clearing an area of obstacles). This can be really useful if you know that the map that you have will not change and there will not be any new obstacles added.

- inflation layer parameters. These determine how close to the obstacles can trajectories be planned.

- robot_radius determines the size of the "forbidden" area around the obstacles. It should be the size of the robot with some safety margins.

- inflation layer -> inflation_radius determines the size of the area near the obstacles where costs are incurred for moving inside it.

- inflation layer -> cost_scaling_factor determines how fast the costs drop, starting from "forbidden" part, to the edge of the inflation layer.

Some additional parameters for the local and global costmaps are set in the `turtlebot_navigation /param /local_costmap_params.yaml` and `turtlebot_navigation /param /global_costmap_params.yaml` parameter files.

1.2 Local planner

A local planner is what connects the robot to the global path plan. The local planner that `move_base` uses by default is the Dynamic Window Approach (DWA) planner which is implemented in the base_local_planner package. Its parameters are set in the `turtlebot_navigation /param /dwa_local_planner_params.yaml`. Some useful parameters that can be set here are:

- Minimum and maximum translational and rotational speeds.

- Minimum and maximum translational and rotational accelerations.

- **Goal tolerance:** how close should the robot be to the goal position and orientation so that the goal is considered reached.

- Various trajectory parameters: weighting how much the robot should follow the global path, how much it should attempt to reach its goal, how much should the controller avoid obstacles, and others.

1.3 Global planner

The global planner which `move_base` uses by default is the `navfn` planner. Its parameters can be set in the `turtlebot_navigation /param /navfn_global_planner_params.yaml`. Optionally you can choose to use the funglobal_planner which is a more general implementation and can be set to behave exactly like `navfn`. The `global_planner` uses Dijktra by default and can also be set to use A*. The parameters for the `global_planner` can be set in funrins_navigation /param /global_planner_params.yaml

2 Customizing `amcl`

The `move_base` node depents on the `amcl` package to provide the location of the robot. The parameters for the amcl node are set in the `turtlebot_navigation/launch/includes/kinect_amcl.launch.xml`. Some parameters that you could change if you are not satisfied with the localization:

- The numbers of particles used in the particle filter.
- Translation and rotational movements needed for performing filter updates.
- Different laser parameter: the maximum and minimum range distances to use when localizing, the number of range reading to use and others.
- The noise in odometry. With the `odom_alpha` parameters you can increase or decrease the noise in odometry which is used in the calculation of the location.
- A lot of other parameters.

Please not that there is a bug in `amcl` which affects the correctness of the location. To fix this bug the `odom_model_type` should be set to `"diff-corrected"`, however, if you do this the you will probably need to adjust the odometry noise parameters (lower `odom_alpha` values). More information is available here and here

3 Homework

No homework!