CS4048 Robotics: Project Report

Marcell Veiner: 51767902

January 2021

1 Introduction & Overview

This report is to accompany my robotics project. In it I plan to give an overview at the tasks attempted, and describe the files submitted. Sections 3-6 correspond to Tasks 1-4, whereas Sections 7 and 8 correspond to Task 6. This will be done per task, with some exceptions for the sake of structure. Please also note that I will try to critically evaluate my methods, and explain the reasoning behind them. To see what the robot can do, please refer to the submitted video file. As an overview, I present the RQT-Graph for my robot in Figure 1.

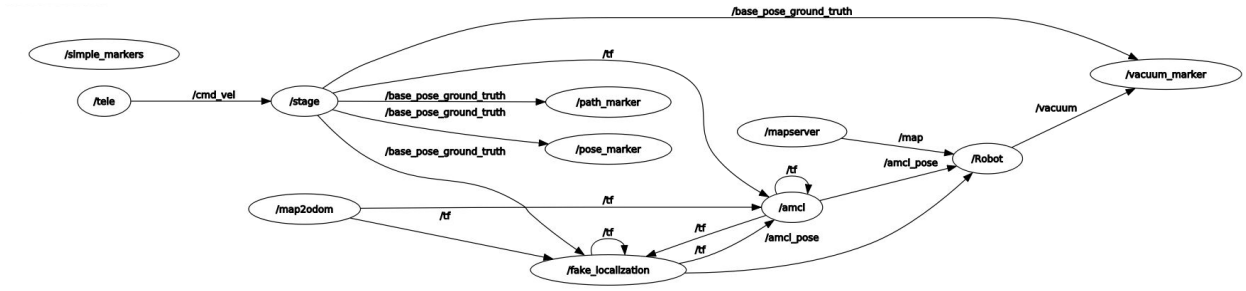

Figure 1: RQT-Grapht

2 Dependencies and Launching

This ros noetic project was created on Ubuntu 20.04 with Python 3.8.5 using Docker Desktop 20.10.0. As it can be observed in the RQT-Graph, my robot uses amcl for localisation. Therefore please install amcl and fake_localization (an API for amcl) according to the relevant documentation in order to launch the project. It also needs numpy and scipy installed, having done these, in a terminal simply write

```
roslaunch assessment assessment.launch
```

This will load stage, and rviz, the latter with an already saved configuration. Therefore no addition of markers is required. (In the unfortunate case of path issues, please load the config.rviz file.

Additionally the marker nodes, and localisation will be launched. We will discuss these next. Note that amcl will cause warnings to pop up in the terminal constantly. This is a recent and known issue, with seemingly no clear solution [1]. According to the developers this may be ignored. Unfortunately I have found no way to silence it, despite looking thoroughly. At this point let me list the parameters that can be specified when launching the project.

- debug = 0/1: This will enable debug mode, speeding up some calculations for speedy launch
- optimal_path = 0/1: Calculate optimal order of rooms or use heuristic
- draw_vacuum = 0/1: Launch separate window for visualising vacuuming progress
- no_vacuum = 0/1: Skip vacuuming all together
- battery = 0/1: Enable battery

[1] https://github.com/ros/geometry2/issues/467

In all cases the default is 0. That said we may launch the robot itself by opening a separate terminal and typing:

```
rosrun assessment robot.py
```

3 Markers and Visualisation

Upon launching rviz we may see that it is already populated with markers, although not all are shown already. Here is a exhaustive list of these markers going from top to bottom (see Figure 2).

- Grid
- LaserScan: The particle cloud obtained from the laser sensor
- Marker (1): Believed pose from odom, green arrow
- Marker (2): True pose, blue arrow
- Path (1): Believed path, green
- Path (2): True path, blue
- Map
- Markerarray (1): With blue spheres the goals and yellow cubes the charging points
- Path (3): The total planned path, i.e. the order of rooms planned, green
- Path (4): Current path, i.e. vacuuming plan / route to charging point / to room, red
- PoseWithCovariance: Pose from amcl, red arrow
- PoseArray: Possible poses from amcl, red arrow cloud
- Markerarray (2): Vacuuming indicators in rviz, coloured cylinders

Please note that when switching to the kidnapped robot, I get strange oddities in rviz, i.e. flickering of some markers.

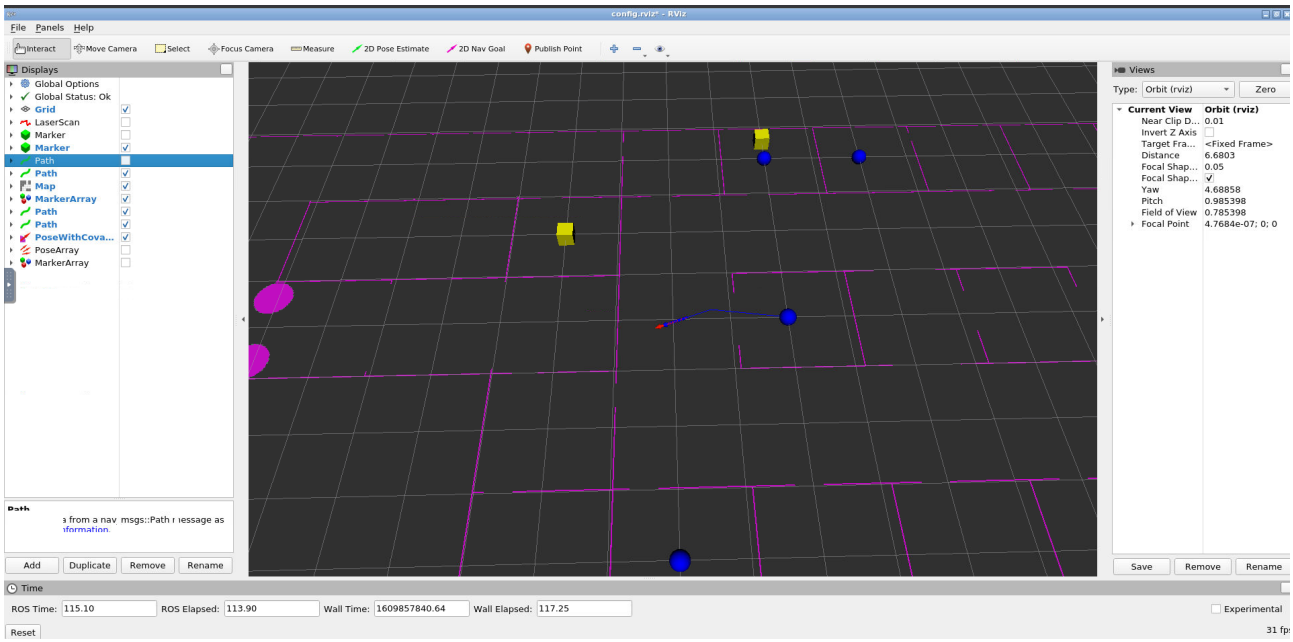

Figure 2: Rviz: Markers

4 Path Planning

4.1 Mapping

This task is solved by the path_planner. It starts by subscribing to the /map topic, and creating the grid representation of the received map on callback. This mostly involves decoding the row-major form, but also I inflate occupied points by the robot radius in grid coordinates, otherwise the robot would try to plot routes it has no ways of completing. Note that this inflation is minimal as it is no more than the robot's width, but it is also maximal as we have so narrow doors, anything more than the current value closes some.

Next we create a costmap, which demotivates the robot for going close to walls on the map. This is done by applying a blur convolution filter to the grid. This also helps smoothing the plotted path.

4.2 Trajectory Planning

Next we have to determine the order of the rooms. The optimal_path parameter as described in Section 2 determines which case we are in. If it is 0, then for all permutations of the rooms heuristic distances (i.e. their distance on the map disregarding walls) are computed, and the minimum selected. Otherwise, we first create all pairs of combinations, removing duplicates by noting that the path from A to B is the same backwards. We then compute the path and cost for all newly encountered paths and select the minimal one. As we keep the best path in memory, we do not need to recompute this either. All in all, this computation was optimised, but still it takes considerable amount of time.

One reason behind that is the use of costmap, this means that at each point we need to evaluate if we may cross into the costly zone or not, which means checking more grid cells. The parameter debug in 2 disables the costmap, to speed up launching, but this will ultimately lead to the robot getting stuck, especially at the top room in the middle, with notoriously narrow entrance.

The A* algorithm uses a Priority Queue, another thing to note is that when obtaining the neighbours of a grid cell, I permute these, which lets me have a smoother path.

This script also includes checking which charging point is closest and then planning a route to it, and then back where we left off.

5 Driving

The controller.py script involves a basic controller for the robot, which is used by the main node, robot.py. The controller simply checks if the point of interest is further than some epsilon, then turns towards it (I enable forward movement if this angle is small) then move proportionally towards it. To get a smooth velocity I use the sigmoid function:

$$\frac{1}{a + e^{b(x-c)}}$$

where a controls the maximum velocity, b the rate of change and c the shift in the x component. This values have been determined by trial and error. The turning velocity is somewhat simpler, it is simply checked that it is contained in the normal usual period of $[-2\pi, 2\pi]$.

This script also includes basic logic for obstacle and wall avoidance. The laser scan is incorporated, as it when close to walls it disables driving. The laser range is divided into 5 arbitrarily defined regions left, front left, front, front right and right. Depending on the minimum distance perceived in these regions the robot behaves slightly differently.

If it was to go straight into an obstacle, then it tries to back up, and turn the other direction. If it is close to a wall on one side, it tries to turn parallel to it (if it is not already), then move forward.

There is also a simple function for turning into a specified orientation, which is used by the vacuuming logic, discuss next.

6 Vacuuming

Vacuuming is mostly handled by the controller.py as for driving. The logic on when to switch to vacuuming is handled by robot.py.

6.1 Vacuum Markers

I used the script given for vacuuming, i.e. launching the project with the parameter draw_vacuum set to 1. Regardless, as it was specified in the exercise I plot the progress in rviz as well. The radius of the markers in

rviz is the same as the vacuuming radius, yet I see difference to the figure using the parameter and in rviz. In particular, there is sometimes a small gap (unvacuumed) in rviz, whereas this does not show in the other figure. I have tuned vacuuming using the script provided, please note that my implementation lets me to simply make the robot more thorough in order to do get rid of these discrepancies.

When the robot start vacuuming, it plots a maze-like path on the floor of the current room (also the layout is visible in rviz). I decided this is a clear indication of which room is being vacuumed, and also the markers left in both rviz and by the other script are of different colour to the background indicating clear progress. Therefore I did not see the reason for cluttering rviz with more markers, which would just simply highlight the rooms floor more.

6.2 Vacuum Navigation

The robot starts by scanning the room and determining the corners of it, or rather than the point of safe distance from these corners. Then it creates a maze-like path across it and follows it. See Figure 3. Note that due to the way of planning the path, this guarantees good coverage. There is another algorithm in controller.py for vacuuming, which spirals outwards from the centre, not used. This was the first version, inspired by how actual vacuum cleaners move.

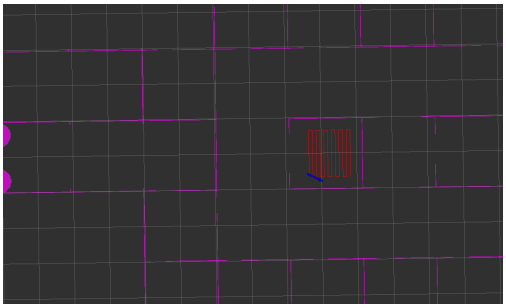

Figure 3: Path Plotted for Vacuuming

7 Battery

The logic for battery is rather simple. The robot keeps checking whether the battery is above a critical threshold, and once it falls below, it plans a route to the closest charging point. There is also functionality to go back and resume what where the robot was. Although this could definitely be further improved, for some reason the robot did not charge for me when next to / directly on a charger, and so development on this functionality was halted.

I am unsure that it is possible to reach all goals with one charge of battery also. Without vacuuming enabled and visiting only the closest room drains my battery down to 60%.

8 Localisation

The robot uses amcl for localisation, and as such does not require to know the starting position. This however causes some glitches in rviz, and given the tight corridors, the robot may get stuck due to the small discrepancies.

9 Reflective Analysis

This project proved to be incredibly difficult due to numerous reasons. I have spent more than the estimated time on it, to be exact 72 hours. I am really grateful for the extension, as this almost doubled the time spent on it. ROS has a steep learning curve (and its documentation, tutorials and community are almost nonexistent), so the initial steps were quite challenging. Driving wasted at least 20 hours of this project. I have written many dead end algorithms, such as trying my own pure pursuit, or writing a wall-follower. It was difficult to continue from here, as more challenging tasks needed stable driving.

I think this was achieved only to some extent. This was a learning process and a first encounter so it was not easy to write stable code like that. Most of this project would benefit greatly from a refactor, but more so from a second start. Yet to stop wasting time on tuning parameters, I had to accept it and move on to the other tasks.

Some tasks were also difficult to assess. I have been using Docker Desktop for running ros, following the initial advice, and although it was fine for the most part, sometimes it could get laggy. This made it rather hard to tell if my robot is starting and stopping quickly, or if it is just the speed of my computer. I really hope that effort will have some weight on the marking.