CMPE 565 AUTONOMOUS ROBOTS
Assignment 8: RRT Path Planning

Deadline: 6th April 2018

In this assignment, you will implement RRT(Rapidly Exploring Random Tree) algorithm. Please update your repository by "git pull". You are going to use myrrt package as a template.

1 Mapping

You are going to use the map from the assignment 7. You also need amcl package for localization on the map. Therefore, make sure that you have completed the assignment 7. You are going to use same simulation scene.

2 Path Planning Implementation

Path planning algorithm supposed to generate way points. Those way points will describe the path of the robot. `goto_pose` package will listen for way points and it will move the robot to that point. Note that the communication between `goto_pose` package and `RRTWrapper` code is already handled.

You will implement your own RRT algorithm by using the `RRTWrapper` code. `RRTWrapper` will provide current position and target position and it expects way point vector from your RRT implementation. You can use any graph library, but we expect your own RRT implementation. You will need a method to check whether there is an obstacle at a certain point on the map. This will also be provided in `RRTWrapper`.

3 Implementation Directives

`goto_pose` package is simply a way point follower package which you will not be concerned about. You will implement your own RRT algorithm in `myrrt` package. You are free to play with the codes, but it is easy to change `MyRRTSolver.cpp` and `MyRRTSolver.h` files. This class is called from the `MyRRTWrapper.cpp` code. As you will discover, `MyRRTWrapper` provides the current position and target position to the `MyRRTSolver` and gets a vector of poses to follow to reach the target position. You are free to use any graph library, but you can also find a very simple graph classes under `src/graph` directory. Therefore, some methods in `MyRRTWrapper.cpp`

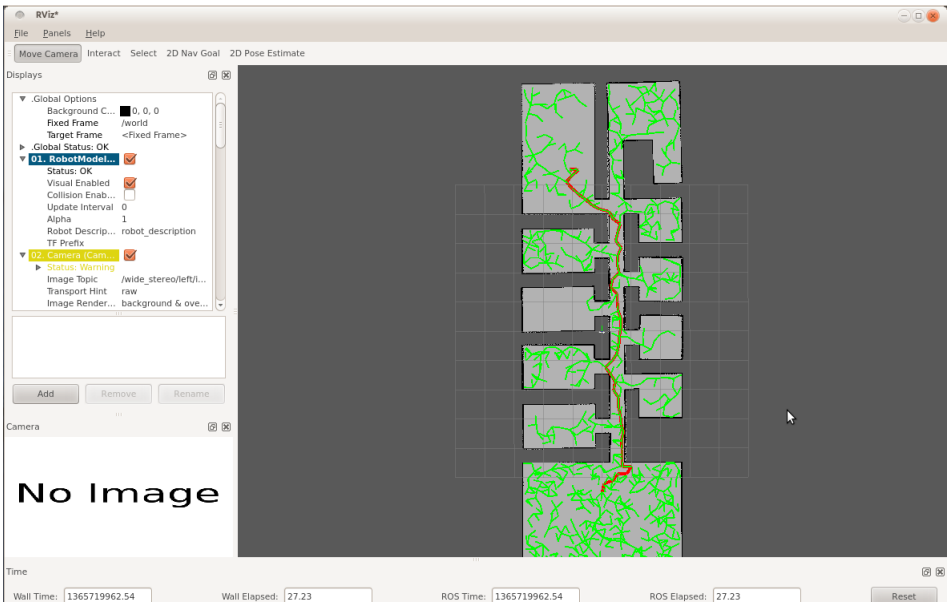

Figure 1: An example RRT visualization

and in `MyRRTSolver.cpp` depends on those classes. Since they are simple data structures, you can write your own classes and modify the related visualization methods.

`MapHelper.cpp` class provides methods to check whether a given point is valid on the map. There is also another method to check whether the line between two points is occupied or not, but **you are going to implement this method**.

Before running launch files, check that you provide correct file for map file that will be used by `map_server` package. By default all launch files expects map.yaml file under the map directory of myrrt package.

`launch/rrttest.launch` file helps you debug your RRT implementation without simulator. It simply reads initial position and target position and generates a path and finally visualizes on the rviz. You should see a picture similar to the Figure 1. Make sure that you run rviz, add map topic, and markerarray topic. You can run launch files with the following command: `roslaunch myrrt rrttest.launch`. You will see similar output. If you have troubles with the visualization, you can ask to the group via e-mail. Some considerations about visualization:

- Before running the code, make sure that you run rviz, add map, and markerarray messages. You should change markerarray topic to ”mygraph”. Also, change fixed frame to odom.
- When you run your algorithm, it builds the RRT tree and calculates a path. Then, it creates the visualization message and send them. If at the time of message sending, you did not run the rviz, you will not see your tree.
- If you run your algorithm, and visualize the markers, the markers will be on the rviz for 10 minutes. If you want to reset marker visualization, click the check box on the markerarray to remove current markers and re-enable to be able to visualize new ones.

`launch/rrtplanning.launch` file runs whole system. Before running this launch files:

- run `roscore`
- run vrep simulator and open assignment8.ttt file
- run the simulation

After these steps we can run `rrtplanning.launch` file. It will run all required packages such as `map_server` and `amcl`. It will also run `myrrtwrapper` node which will generate path and communicate with `goto_pose` package to move robot from one position to another. You can run launch files with the following command: `roslaunch myrrt rrtplanning.launch`

Some general notes:

- For now `goto_pose` package moves robot very slowly. You can change the `goto_pose` package to move the robot faster and safer.
- `MapHelper` class uses original map, but due to motion noise and localization errors, we should add a safety thickness to our map. It can be done by changing maphelper code to treat walls as if they are thicker.
- `goto_pose` package uses the localization information provided by `amcl` package. If you cannot achieve to move the robot safely (due to localization errors), you can try odom frame in `goto_pose` package because it is provided by the simulator as the position of the robot.

4 RRT as Global Path Planner

If you complete the assignment by using `myrrt` and `goto_pose` package, you will get **80** points. If you integrate your rrt algorithm implementation to `move_base` as a global path planner, you will get **100** points. If you integrate your path planning algorithm to `move_base`, you are going to replace your navigation task by giving navigation goals on the rviz. However, if you use `goto_pose` package, you will give navigation goals by the parameters of the launch files.

5 Deliverables

- Comment on the parameters of the RRT algorithm by trying to answer the following questions. How did you determine the step size? What is your measure to end the tree building phase? How to did you thicken walls to safely move the robot?
- Comment on the narrow passages. How your robot navigates on the passages? Do you use any path smoothing algorithm or any other method to move the robot in narrow passages?
- Provide an image shows the built RRT and calculated path on rviz.
- Provide a video that shows the robot following the generated path. (Note that: this part can be done using both `goto_pose` and `move_base` package.)

6 Resources

- https://en.wikipedia.org/wiki/Rapidly_exploring_random_tree
- http://wiki.ros.org/navigation/Tutorials/Writing%20A%20Global%20Path%20Planner%20As%20Plugin%20in%20ROS