CS3027/5059 Robotics

Assessment I

2016
version 1.0

1 Overview

This assessment will evaluate your ability to perform path planning and localisation. It aims to evaluate the following two learning outcomes:

- Familiarity with Path planning algorithms.
- Ability to implement robot localisation.
- Integrate with RViz to visualise your robot in the world.
- Good ROS programming practices.

Your robot will operate within the stage simulator, which can be run using the command

```
rosrun stage_ros stageros ‘rospack find stage_ros‘/world/willow-erratic.world
```

to use the willow-erratic map. A different map is provided below.

2 Practicalities

The due date for this assessment is 5pm on Friday the 28th of October. Handing in up to 24 hours late will incur a 10% penalty, while handing in up to 7 days late will incur a 25% penalty. After that, you will receive 0 for the assessment.

Please be aware of the University’s policy on Plagiarism, ensure that all work is your own, and that any sources you use are acknowledged. Additionally, please make sure that all source code you hand in follows programming best practices and is well documented.

This assessment will count for 55% of your overall course mark. You may work in teams of 2, or individually. If you work in a team, please include a short text file describing who did what in the project. You may also tell me if you feel your partner is not pulling their weight. If you work individually, you will gain an automatic 10% bonus to your final mark.

3 Preliminaries

1. You will need to install the map server (`sudo apt-get install ros-indigo-map-server` for indigo, or `ros-kinetic-map-server` for kinetic).
2. Please download the zip file from the assessment page on myAberdeen which contains
    - A `.world` file (for stage)
    - A map image file (`map.png`)

- A `.yaml` file for the map server
- A python node (`addGaussianNoise.py`). This node publishes a topic, `/noisy_base_scan` which you will need to use instead of `/base_scan` in your code.
- A launch file (`r.launch`).

3. The r.launch file will start up ROS, the map server a static transform and stage using the appropriate map (assuming you put everything in the /home/viki directory, otherwise, please edit the yaml file).

4. The static transform creates a `/map` frame which can be used in RViz. If you prefer, you can edit the launch file to use `fake_localization` as per practical 3.

5. Start up RViz (just type `rviz`) and play around with it. For example, try to add the map to the display, as well as the various frames of reference. To do this, choose "Add" on the left hand part of the window and choose the appropriate item. You may need to type in the correct frame/topic to make this work.

4 Tasks

4.1 Task 1

Implement a ROS node that subscribes to `/base_pose_ground_truth` and broadcasts a transform as a child of the `/map` frame called `/real_robot_pose`. Then use this information to create a marker on the RViz window, displaying the robot's true location and orientation. You should also display the robot's pose, as obtained from odometry information, within RViz, enabling me to see the difference between where the robot is, and where it believes it is.

4.2 Task 2

The main aim of this assessment is to have your robot visit various points in the world. The only problem is that it's going to have been kidnapped at the start...

The aim of this task is to perform robot localisation. You may implement either a EKF or particle filter based localisation algorithm to address this task, or utilise an existing library. The task will be deemed successful when the robot goes to a start point, as given by the parameter `/start_pos` (passed as a list $[x, y]$). You should create a marker when the robot reaches the start, as we will need to measure the distance between the actual start and where it thinks it is.

Alternatively, you can choose to solve a non-kidnapped version of the problem, in which case you should read the parameter `robot_start` which will be passed as a $[x, y, \theta]$ triple.

4.3 Task 3

Your robot will be passed a sequence of 6 points from the parameter server, `p1`, `p6`, again passed as a list. From the start position, your robot must visit these 6 points (in any order you choose). Additionally, you must display these 6 points and the path you choose to follow on RViz, and also display the path your robot *actually* followed (using what you implemented in Task 1). Markers need to be dropped at each location by your robot.

You may also prefer to do this task with no odometric noise (i.e., if you haven't had much luck with task 2).

As an extra challenge, you may wish to implement a local obstacle avoidance algorithm to avoid obstacles detected but not shown on the map you've been given.

5 Marking

Marking of this assessment will take place in the practicals of Monday the 7th and Tuesday the 8th of November. You will receive 20 marks for achieving Task 1.

Achieving Task 2 will give you 20 marks if you utilise existing ROS stacks, or 30 marks if you implement the localisation yourself. Achieving this task with a known initial starting pose will reduce the marks you receive by 10.

Task 3 is worth 50 marks. Utilising existing ROS stacks will reduce this to 30 marks, and doing this task with no sensor or odometric noise will reduce your possible maximal mark by another 10 points. Note that I expect you to utilise path planning here in the first instance - solving this problem purely with a local obstacle avoidance algorithm is possible, but will reduce marks by another 10 points. On the other hand, combining path planning with a local obstacle avoidance algorithm (and assuming that you implemented both yourself) will add 20 points to your mark.

Note that marks will be deducted if your robot wanders too far from the optimal (or planned) path, believes it is at a goal when it is far away, etc (these will be judged somewhat subjectively, but you can assume that a 20cm error will cost you one mark, 40cm two marks, etc, for each part of the mark, i.e., moving far away from the optimal line could cost several marks, as could being far away from the goal).

6 Hints

- Both sensor and odometer noise will be present, take a look at the files to see how much noise exists.
- Don't forget your robot has a physical size.
- There might be a faster way to reach the points than visiting them in order.
- **N.B.** I may change some of the parameters (e.g., noise, penalties for time/distance) during the assessment. Parameters like noise will be stabilised quickly, but you should make sure you can change them easily in your code.