AIMS CDT Week - Localisation Exercise

1 Overview

In this exercise, you will be implementing some of the key functionality of a particle filter for Monte-Carlo localisation. More information about how a particle filter operates can be found in Chapters 7-8 of [1].

2 The Code

The code for this exercise can be found in the `pf_localisation` ROS package. Below is a summary of the important files in this package:

- `scripts/localisation.py`: The main ROS node for running the particle filter.
- `src/viz.py`: Utility functions to produce RViz markers.
- `src/pf_localisation/pf.py`: The file you will be writing in!
- `src/pf_localisation/pf_base.py`: An abstract base class for the particle filter.
- `src/pf_localisation/sensor_model.py`: The robot's sensor model.
- `src/pf_localisation/util.py`: Utility functions for quaternions.

3 The Task

In this exercise, your task is to fill in the empty functions in `pf.py` to complete the particle filter. These functions are:

1. `generate_random_pose`: This function should randomly generate and return an unoccupied pose in the map.
2. `initialise_particle_cloud`: Given an initial pose (you can provide this in RViz) generate an initial particle cloud. There are two ways to implement this function:
   (a) Generate a set of uniform random poses across the map.
   (b) Use the initial pose as the mean of a Gaussian and generate the particles from that. The Von Mises distribution can be used to compute the orientation of the particles.

I'd recommend trying both and comparing the two in different scenarios. For example, what if you give the robot a poor initial pose?

3. `update_particle_cloud`: This function updates the particle cloud given a laser scan. Odometry predictions are dealt with elsewhere so in this function, a new set of particles must be generated by resampling based on the particle likelihoods using the sensor model.

4. `estimate_pose`: This function should take the current particle cloud and generate the estimated pose of the robot. This can be done in a number of ways (see Section 5). A simple solution is to take the average of the particles, but be creative!

More detailed descriptions can be found in the docstrings of the source code.

4 Running the Particle Filter

To run the particle filter, run the `./start_jackal.tmux` script. If you get errors related to `laser_trace`, run the `compile.sh` script in the `src/laser_trace` directory of the `pf_localisation` package. If everything starts successfully, give the robot an initial pose in RViz, and then teleop the robot with the PS4 controller. You should then see the particle cloud being updated in RViz.

5 Extensions

If you finish the main task, you could improve your particle filter in the following ways:

1. Consider improving the method for resampling particles in `update_particle_cloud` by using the KLD sampling method outlined in [2].

2. Consider better methods for estimating the robot pose, e.g. the DBScan clustering algorithm introduced in [3].

3. Adapt your particle filter to solve the 'kidnapped robot problem'. This can be done by adding a small number of randomly generated particles to the particle cloud when resampling in `update_particle_cloud`.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press Cambridge, 2000, vol. 1.

[2] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[3] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, 1996, pp. 226–231.