University of Technology Nuremberg
Lecture: Introduction to Mobile Robotics
Held at the Technical University of Munich
Winter term 2022/23

Department of Engineering

Prof. Dr. Wolfram Burgard
Michael Krawez
Reihaneh Mirjalili

FastSLAM Project Pt. 1

FastSLAM Implementation

FastSLAM is a Rao-Blackwellized particle filter for simultaneous localization and mapping. The pose of the robot in the environment is represented by a particle filter. Furthermore, each particle carries a map of the environment, which it uses for localization. In the case of landmark-based FastSLAM, the map is represented by a Kalman Filter, estimating the mean position and covariance of landmarks.

Implement the landmark-based FastSLAM algorithm as presented in the lecture and using the Webots simulator. Assume known feature correspondences.

To support this task, we provide a detailed listing of the algorithm as a PDF file and a Webots framework in the archive `fast_slam.zip`. The archive, among others, contains the following folders:

worlds contains the Webots world file `fast_slam.wbt`.

controllers contains the corresponding controller file `fast_slam_controller.py`. That file already contains a FastSLAM framework where you need to fill in the blanks in the code.

(a) Familirize yourself with the Webots simulator. First, download and install the version for your system from https://cyberbotics.com (available for Windows, Mac, and Linux). Then, follow the official tutorial at

https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots

After that, open `fast_slam.wbt` in the simulator. The world contains a Pioneer robot and several objects that are considered as landmarks. You can control the robot using the arrow keys. After moving the robot, a plot will appear that shows the map of the environment and the estimated robot trajectory. However, the algorithm will only work correctly after you complete the code in `fast_slam_controller.py` as described below.

(b) Complete the code blank in the `sample_motion_model` function by implementing the odometry motion model and sampling from it. The function updates the poses of the particles based on the old poses, the odometry measurements δ_{rot1}, δ_{trans} and δ_{rot2} and the motion noise. The motion noise parameters are:

$$[\alpha_1, \alpha_2, \alpha_3, \alpha_4] = [0.1, 0.1, 0.05, 0.05] \tag{1}$$

(c) Complete the code blanks in the `eval_sensor_model` function. The function implements the measurement update of the Rao-Blackwellized particle filter, using range and bearing measurements. It takes the particles and landmark observations and updates the map of each particle and calculates its weight w. The noise of the sensor readings is given by a diagonal matrix

$$Q_t = \begin{bmatrix} 1.0 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{2}$$

(d) Complete the function `resample_particles` by implementing stochastic universal sampling. The function takes as an input a set of particles which carry their weights, and returns a sampled set of particles.