Albert-Ludwigs-Universität Freiburg
Lecture: Introduction to Mobile Robotics
Summer term 2017

Institut für Informatik

Prof. Dr. Wolfram Burgard
Dr. Daniel Büscher
Marina Kollmitz
Oier Mees

mobilerobotics@informatik.uni-freiburg.de

Sheet 7

Topic: Discrete Filter, Particle Filter

Due date: 16.06.2017

Exercise 1: Discrete Filter

In this exercise you will be implementing a discrete Bayes filter accounting for the motion of a robot on a 1-D constrained world.

Assume that the robot lives in a world with 20 cells and is positioned on the 10th cell. The world is bounded, so the robot cannot move to outside of the specified area. Assume further that at each time step the robot can execute either a *move forward or a move backward* command. Unfortunately, the motion of the robot is subject to error, so if the robot executes an action it will sometimes fail. When the robot moves forward we know that the following might happen:

1. With a 25% chance the robot will not move
2. With a 50% chance the robot will move to the next cell
3. With a 25% chance the robot will move two cells forward
4. There is a 0% chance of the robot either moving in the wrong direction or more than two cells forwards

Assume the same model also when moving backward, just in the opposite direction.
Since the robot is living on a bounded world it is constrained by its limits, this changes the motion probabilities on the boundary cells, namely:

1. If the robot is located at the last cell and tries to move forward, it will stay at the same cell with a chance of 100%
2. If the robot is located at the second to last cell and tries to move forward, it will stay at the same cell with a chance of 25%, while it will move to the next cell with a chance of 75%

Again, assume the same model when moving backward, just in the opposite direction.
Implement in Python a discrete Bayes filter and estimate the final belief on the position of the robot after having executed 9 consecutive *move forward* commands and 3 consecutive *move backward* commands. Plot the resulting belief on the position of the robot.

Hints: Start from an initial belief of:

$$bel = numpy.hstack\left((numpy.zeros(10), 1, numpy.zeros(9))\right)$$

You can check your implementation by noting that the belief needs to sum to one (within a very small error, due to the limited precision of the computer). Be careful about the bounds in the world, those need to be handled ad-hoc.

Exercise 2: Particle Filter Implementation

In the following you will implement a complete particle filter. A code skeleton with the particle filter work flow is provided for you. A visualization of the particle filter state is also provided by the framework.

The following folders are contained in the pf_framework.tar.gz tarball:

data This folder contains files representing the world definition and sensor readings used by the filter.

code This folder contains the particle filter framework with stubs for you to complete.

You can run the particle filter in the terminal: `python particle_filter.py`. It will only work properly once you filled in the blanks in the code.

(a) Complete the code blank in the `sample_motion_model` function by implementing the odometry motion model and sampling from it. The function samples new particle positions based on the old positions, the odometry measurements δ_{rot1}, δ_{trans} and δ_{rot2} and the motion noise. The motion noise parameters are:

$[\alpha_1, \alpha_2, \alpha_3, \alpha_4] = [0.1, 0.1, 0.05, 0.05]$

The function returns the new set of parameters, after the motion update.

(b) Complete the function `eval_sensor_model`. This function implements the measurement update step of a particle filter, using a *range-only* sensor. It takes as input landmarks positions and landmark observations. It returns a list of weights for the particle set. See slide 15 of the particle filter lecture for the definition of the weight w. Instead of computing a probability, it is sufficient to compute the likelihood $p(z|x, l)$. The standard deviation of the Gaussian zero-mean measurement noise is $\sigma_r = 0.2$.

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

Some implementation tips:

- To read in the sensor and landmark data, we have used dictionaries. Dictionaries provide an easier way to access data structs based on single or multiple keys. The functions `read_sensor_data` and `read_world_data` in the `read_data.py` file read in the data from the files and build a dictionary for each of them with time stamps as the primary keys.

  To access the sensor data from the `sensor_readings` dictionary, you can use

  ```
  sensor_readings[timestamp,'sensor']['id']
  sensor_readings[timestamp,'sensor']['range']
  sensor_readings[timestamp,'sensor']['bearing']
  ```

  and for odometry you can access the dictionary as

  ```
  sensor_readings[timestamp,'odometry']['r1']
  sensor_readings[timestamp,'odometry']['t']
  sensor_readings[timestamp,'odometry']['r2']
  ```

  To access the positions of the landmarks from `landmarks` dictionary , you can use

  ```
  position_x = landmarks[id][0]
  position_y = landmarks[id][1]
  ```