COMP 7308: Introduction to Unmanned Systems Fall 2021

Assignment 3

:

1 Particle filter (50 points)

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

The following folders are contained in the pf.zip achieved file:

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

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

doc This folder contains the pdf file of the KLD sampling paper.

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. For all the questions below, beside the code, please also draw some figures in the report to illustrate your result.

(a) (5 points) 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. See the motion model lecture for details of implementing the odometry motion model.

(b) (5 points) Complete the function `eval_sensor_model`. This function implements the measurement update step of a particle filter, using a *range-only* sensor, i.e., the sensor only measures the distance of the robot to the object but not the relative angle. It takes as input landmarks positions and landmark observations. It returns a list of weights for the particle set. See the particle filter lecture for the definition of the weight w. Instead of computing a probability, it is sufficient to compute the likelihood $p(z \mid x, l)$. The standard deviation of the Gaussian zero-mean measurement noise is $\sigma_r = 0.2$.

(c) (5 points) 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.

(d) (10 points) Add in the python code the part for taking into account the bearing measurement in the particle filter localization.

(e) (10 points) Change the noise level (i.e. the value of Q and R) and the number of samples used, discuss the performance of particle filter localization using different settings with some figures and numbers, e.g., how the estimation error and computational time change when these parameters vary.

(f) (15 points) Read the KLD-sampling.pdf in the doc folder, which presents the KLD sampling algorithm that we briefly discussed in class. Implement the KLD sampling algorithm and compare its performance with the default re-sampling algorithm in (c).

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]
```

2 (50 points) Fast SLAM 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.

In this task, you need to implement the landmark-based FastSLAM algorithm as presented in the lecture. Assume known feature correspondences.

To support this task, we provide a detailed listing of the algorithm as a PDF file and a Python starter code. The framework contains the following folders:

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

starter_code This folder contains the FastSLAM starter code with stubs for you to complete.

doc This folder contains the detailed listing of the algorithm as a PDF file.

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

(a) (10 points) 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 $\delta_{rot1}, \delta_{trans}, \delta_{rot2}$ and the motion noise. The motion noise parameters are:

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

How is sampling from the motion model different from the standard particle filter for localization that you saw and implemented in Problem 1?

(b) (10 points) 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

$$Q_t = \begin{pmatrix} 1.0 & 0 \\ 0 & 0.1 \end{pmatrix}. \tag{1}$$

How is the evaluation of the sensor model different from the standard particle filter for localization that you saw and implemented in Problem 1?

(c) (10 points) 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.

How does the resampling procedure differ from resampling in the standard particle filter for localization that you saw and implemented Problem 1?

(d) (10 points) Try your implementation with different values for Q_t and $[\alpha_1, \alpha_2, \alpha_3, \alpha_4]$ and report your observations using figures, numbers, or analysis.

(e) (10 points) Try your implementation with different number of samples and report your observations about SLAM quality and computational cost using figures, numbers, or analysis.

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']
  ```