COMS W4733: Computational Aspects of Robotics

Homework 6

Due: May 8, 2019

This assignment consists of two parts. Part 1 should be completed and submitted individually. Part 2 may be completed either by yourself or with one other student. If working with a partner, please be sure to use the "Group submission" option on Gradescope.

1 Reading on Visual Localization (60 points)

Carefully read "Localization Methods for a Mobile Robot in Urban Environments" by A. Georgiev and P. Allen (2004). Please provide responses to the following.

1. (a) Find the state transition matrix $\mathbf{\Phi}_k$ in Eq. (3). You may leave it in the factored form as shown there, but please explicitly find $\mathbf{F}$. The matrix $M(\mathbf{r})$ is given by

$$M(\mathbf{r}) = \begin{pmatrix} 1 & \sin\varphi\tan\theta & \cos\varphi\tan\theta \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi\sec\theta & \cos\varphi\sec\theta \end{pmatrix}.$$

0, 0, 0, 0, 0, pi/4 开始只绕z

   (b) Suppose that the robot starts at $\mathbf{x}(0) = (0, 0, 0, \pi/4, 0, 0)^T$ and that the encoders sample $\Delta\phi_1 = \pi/6$ radians and $\Delta s_1 = 5$ meters between 0 and 2 seconds. Referring to Eqs. (6)-(9), what is the reference trajectory $\tilde{\mathbf{x}}(t)$?

2. For building model selection, Sect. 4B describes a process in which the robot selects the closest model that passes a distance and viewing angle test. Since we don't want models that are either too close or too far, would choosing a model in the middle of this range, rather than the closest, would be a good design choice?

3. (a) The authors' RANSAC implementation uses the minimum number of three matching pairs of segments for pose estimation. Would there be any benefits or drawbacks of using more than three pairs?

   (b) Once a pose is found, the authors evaluate it using a coverage metric on the consensus set of matching edges. Briefly describe (at a high level) what we would expect to happen with the pose estimation if we use the consensus set found only from a single threshold (e.g., the initial "large" value). What if we use the normalization procedure of dividing by the total projected length—would underrating correct poses be an issue?

4. Fig. 18 shows an example on which matching has failed. Find and take a picture of another building facade on campus where this may happen. While you don't have the robot and camera parameters, try to replicate the robot's perspective similar to the other figures (e.g., don't just zoom all the way in on a picture of two identical windows). Briefly explain why your image would also be problematic.

2 Particle Filter Implementation (90 points)

You will be implementing a simple particle filter in Python. The task will be to localize a robot in an environment with a given number of landmarks. There will be no ROS component.

2.1 Provided Files

The `world.txt` file provides the size of the environment on the first line and positions of various landmarks on subsequent lines. You are free to modify these parameters to test your code.

The `robot.py` file provides a basic class to represent a robot as well as its associated particles. There are three pose uncertainty parameters and four parameters for transition model uncertainties as well as observation model uncertainties.

The `visualize.py` file contains the main simulation loop. It should be run with three parameters: the `world.txt` file, number of particles, and number of iterations. The robot and particles are all initialized in random states in the main function; note that we're explicitly setting the robot noise components to be $(0.5, 0.1, 5.0, 1.0)$. Landmarks are all squares with side lengths of 6, while the robot is assumed to be a point particle (it is simply drawn as a circle).

2.2 Tasks

1. (30 points) Implement the `move` function in the `Robot` class. The robot should update its state according to the nonlinear transition function below and return the action input $\mathbf{u}_k$.

$$\begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix} = \begin{pmatrix} x_{k-1} \\ y_{k-1} \\ \theta_{k-1} \end{pmatrix} + \underbrace{\begin{pmatrix} 10\cos\theta_k \\ 10\sin\theta_k \\ \Delta\theta_k \end{pmatrix}}_{\mathbf{u}_k} + \begin{pmatrix} w_x \\ w_y \\ w_\theta \end{pmatrix}$$

   - The transitions in x and y are functions of θ_k, which means that the robot first turns according to $\Delta\theta_k$, followed by a movement of 10 units along its current heading. The transition must be done in this order.
   - w_x, w_y, and w_θ are all zero-mean Gaussian random variables with variance given by `forward_noise` in the first two components and `turn_noise` (radians) in the third.
   - Before actually moving, the robot should assess whether it is likely to pass through a landmark or a wall. You can call the `sense` function to obtain observations and determine whether any landmarks lie in its proposed path. If so, the robot should turn until it determines that it is safe to move forward; the total angle turned will be $\Delta\theta_k$.
   - Your robot should be aware that there is noise in both transitions and observations. In other words, you will probably want to have a buffer between where the robot thinks it's going and where it thinks the obstacles are. How you determine this is up to you, but the robot's noise parameters should offer some guidelines.

2. (20 points) Implement the `sense` function in the `Robot` class. Suppose that the robot has a 360-degree range bearing sensor. It should return measurements for all landmarks visible from its current position; in other words,

$$\mathbf{z}_k = \left(r_1, \phi_1, r_2, \phi_2, \cdots, r_M, \phi_M\right)^T + \left(v_{r1}, v_{\phi 1}, v_{r2}, v_{\phi 2}, \cdots, v_{rM}, v_{\phi M}\right)^T.$$

The Gaussian noise random variables are all zero-mean and have variance given by `range_noise` for the range measurements and `bearing_noise` for the bearing measurements, respectively.

3. (10 points) Implement the `move_particle` function in the `Robot` class. This should be much simpler than moving the robot since the input vector $\mathbf{u}_k$ is already determined by the robot. Otherwise, each particle should move according to the same transition model given above (don't forget additive Gaussian noise).

4. (30 points) Implement the `update` function in `visualize.py`. This function should first have the robot move and take a measurement. Each particle should then move using the robot's input vector $\mathbf{u}_k$ and the `move_particle` function and then get a weight assigned to it.

   First, if a particle has moved into a landmark or outside the environment, it should receive a weight of 0. Otherwise, a particle can use `sense` to get an expected measurement. For simplicity, you can assume that all components of this vector are independently distributed as univariate Gaussians with means equal to the expected measurement components and variances of `range_noise` or `bearing_noise`. Then a particle's weight would be the cumulative product of the likelihoods of each actual measurement component sensed by the robot.

   Finally, all particles should be resampled; a simple way would be to use `numpy.random.choice`, which allows for sampling with replacement from a given set with a provided parameter of normalized importance weights.

2.3 Writeup and Submission

The visualization of the particle filter does not delete any old particles or robot states, but it does show each successive iteration's particles in a darker shade of blue (up to 10 iterations). Try to experiment with the number of particles needed so that you get an obvious convergence of particles to the robot's pose within 10 iterations (if you have a lot of particles, you can modify the draw functions to show a subsample of them). Record one or two videos of different successful instances.

In a separate PDF writeup, please describe any implementation choices or techniques not specified in the previous section. In particular, describe the motion model and how your robot determines how much to turn before it moves. Also describe the observation model if you used any custom computations there, and provide some observations about the effectiveness of the weighting and resampling steps. How many particles on average did you end up using?

Be sure to submit all your code, writeup, and videos on Gradescope.