Homework Assignment 4: *Localization*

Overview

In this assignment, you will implement the “Monte Carlo Localization” technique (particle filter for localization) as described in class.

Homework submission notes

Please mark your team number in the filenames of the files you submit – at least in the filename of the .zip file (if submitting a single .zip) or the .pdf write-up file (if submitting individual files).

Coding style requirements

Each function must be documented with its purpose, arguments, and outputs, like this:

```
def get_motor_speeds(lin_vel,ang_vel):
     “””Calculates Sparki motor speeds from linear and
angular velocities.
     Arguments:
          lin_vel: linear velocity in cm/s
          ang_vel: angular velocity in rad/s
     Returns:
          Left motor speed, left motor direction, right
motor speed, and right motor direction
     “””
     # calculations go here
     return left_motor_speed, left_motor_dir, ...
```

Variables should have understandable names and comments documenting their purpose. Classes and significant pieces of code should be commented as well.

Include a README file which explains how to run your code.

Assignment components

1. Add a function to draw the particles. (10 points)

I have provided a starting point for the ParticleFilter class. It stores the particles as an n x 2 matrix (where n is the number of particles).

In this implementation we will not estimate θ with the particle filter; we will simply use the "true" value of θ from the simulator.

Each particle contains a hypothesis for (c_x, c_y). The class also stores a weight value for each particle.

i. Create a draw() function which draws the particles. At minimum, draw each particle as a small circle centered at (c_x,c_y).

   Optional: You can also draw a line to indicate θ or even draw a "ghost" robot in gray.

2. Create a function to sample from the motion model. (20 points)

In this part, you will add a function sample_motion_model() which samples from the velocity motion model discussed in class. The function should take as input the timestep (time_delta). The function will return the T_motion transformation matrix corresponding to the sampled motion.

Here is the algorithm for the function. I used 0.5 for all of the α_i coefficients.

i. Sample a linear velocity by adding Gaussian noise to the robot's linear velocity. To sample Gaussian noise use numpy.random.normal. The standard deviation for the linear velocity noise should be:

   $$\sqrt{\alpha_1 v^2 + \alpha_2 \omega^2}$$

   where v is the input linear velocity and ω is the input angular velocity.

   Sample an angular velocity by adding Gaussian noise to the robot's angular velocity. The standard deviation for the angular velocity noise should be:

   $$\sqrt{\alpha_3 v^2 + \alpha_4 \omega^2}$$

ii. Calculate T_motion using the sampled linear and angular velocities. You can use the same calculation that is provided in Robot.py.

3. Create a generate() function which randomly moves each particle by sampling from the motion model (20 points).

The generate() function will update each particle's position by sampling a motion from the motion model and applying the motion to the particle.

The function should take as input the timestep (time_delta).

For each particle:

i. Calculate T_robot_map using the particle's (c_x,c_y) and the robot's θ value.

ii. Sample T_motion using the sample_motion_model() function, right-multiply T_robot_map by T_motion, and then store the resulting (c_x,c_y) value in the particle.

4. Create an update() function which calculates the weight for each particle according to the rangefinder measurement and map. (20 points)

The update() function will update each particle's weight by comparing the rangefinder reading with the expected rangefinder reading according to the particle's position.

For each particle:

i. Calculate T_sonar_map using the particle's (c_x,c_y), the robot's θ value, and T_sonar_robot.

Get the expected rangefinder reading using omap.get_first_hit().

Calculate the particle's new weight value as:

$$e^{\frac{-(z-d)^2}{\sigma^2}}$$

where **z** is the rangefinder reading and **d** is the expected rangefinder reading. I used $\sigma^2=10$ in my code.

ii. Normalize the weights by dividing each weight by the sum of all the weights.

5. Evaluate the particle filter localization algorithm. (30 points)

Submit the write-up as a Word doc or PDF.

i. Test the algorithm by running localization.py several times, driving around the map and seeing if the particles converge on the robot. Do they converge or not? Describe the behavior you see.

ii. Test both increasing and decreasing the spread of the initial set of particles. How does this affect the algorithm?

iii. Test increasing and decreasing the noise parameters $\alpha_1,\ldots,\alpha_4$. How does this affect the algorithm?