Comp 150 Probabilistic Robotics
Homework 2: Image-Based Particle Filter for Drone Localization

In this problem, your task is to implement an image-based particle filter for localization against a known aerial map. The agent is a flying drone whose camera points down and can take small images, while the map is an image of the entire area.

1 Simulation Environment

Your simulation environment should enable the agent to randomly move in the x and y directions, simulate taking sensor measurements before each movement, as well as generate a reference image for a particular position on the map. You can assume that the drone has a very accurate on-board compass and always orients itself in the North direction before taking an image (in other words, you do not need to worry about the drone's orientation, only its x and y coordinates). Here are some preliminaries/specifications:

- The origin of the map, (0,0), is at the center of the image and 1 unit of distance = 50 pixels. Given an input image as the map (three images are included in this homework), your code should automatically calculate the range of the map in the x and y directions (note that for most image processing APIs, the image origin 0,0 is at the top left corner of the image).
- The drone's starting x, y position is randomly generated according to the uniform distribution. Thus, the state vector $\mathbf{x} = [x, y]^T$.
- Your simulator should be able to simulate an RGB image reading, $\mathbf{z} \in \mathcal{R}^{m \times m \times 3}$, given the drone's true position (for extra credit, also add some noise in the measurement), as well as generate the reference image for any particular x, y position. Both the observation image and the reference image should be of size $m \times m$ (you can experiment with the value of m, and perhaps start with a value of 25).
- At each time step, your simulator should generate a random movement vector in the x and y direction, described as $[dx, dy]^T$ such that $dx^2 + dy^2 = 1.0$ (you are welcome to experiment with faster or slower moving drones). When randomly generating movement vectors, you should reject any that

move the agent off of the map. This movement vector will be known to the agent using the particle filter to localize against the map.

- The actual position x_{t+1} should then be set to $x_t + dx + \mathcal{N}(0, \sigma^2_{movement})$. Similarly, $y_{t+1} = y_t + dy + \mathcal{N}(0, \sigma^2_{movement})$. The constant $\sigma^2_{movement}$ represents the uncertainty in the robot's movement (e.g., due to wind, etc.). The agent implementing the particle filter does NOT have access to the actual displacement, only the movement vector (i.e., intended displacement) before noise is applied.

- At each time step, your simulator should draw a circle on the image so that the user can see the true position of the drone. The user should press enter in between time steps to advance the simulation.

2 Particle Filter Implementation

Once you have a simulation environment going, it is time to implement the two main steps of the particle filter: 1) sensing and re-sampling particles according to the likelihood of seeing the observation; and 2) moving the particles according to the known movement vector (with some noise added).

Following, are the guidelines:

- Initially, generate a set $\mathcal{P}$ of N particles, uniformly distributed across the map.

- Implement a function approximator for $P(\mathbf{z}|\mathbf{x})$ which you will use to set the weight for each particle.

- Implement weighted importance sampling with replacement to generate the re-sampled set $\mathcal{P}'$.

- Finally, move each particle according to the known movement vector $[dx, dy]^T$, then add some position noise.

- At each stage, draw the particles on top of the image as circles; when computing the weights, you can re-draw each particle with circle diameter proportional to its weight.

3 Experiments and Evaluation

As part of the homework, you are in charge of defining a metric for how well your particle filter works. One simple metric, though probably not the best one, is the probability of the largest cluster of particles being centered around the true position after some fixed number of time steps, K.

After defining your metric(s), it is now time to run an experiment in which you compare (at least) two different conditions. For example, you can look at what happens when the size of the observation image becomes larger. Or

perhaps you have two different implementations of the function $P(\mathbf{z}|\mathbf{x})$. You could also vary the number of particles, N. Feel free to be creative.

4 What to Turn In

You should turn in your code and a README file with some basic instructions on how to run it. A PDF report should contain a brief description of your algorithm and experiments. Results should be shown with graphs or tables. The report should also contain an illustrative example showing the particles, along with the true position at $t = 0$ and then showing how they move and cluster over some number of time steps.

5 Extra Credit

- Implement a GPS sensor in your simulation with relatively high variance. Decide how to set the particles' weights according to both the probability of observation and the GPS sensor reading. Compare the results with and without the GPS sensor.
- Implement a function for $P(\mathbf{z}|\mathbf{x})$ that does something else other than the simple distance-based measure discussed in class. For example, you could compute the color histograms of the observation and the reference image and use those to estimate a similarity.
- When simulating the drone's measurement image given its true position, add some white noise to the resulting image. Do an experiment to detect how much noise needs to be added for performance to suffer according to your metric.