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

Milan Dahal

This is a description of implementation 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

The simulation environment enables 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. We have assumed that the drone has a very accurate on-board compass and always orients itself in the North direction before taking an image. The description of the simulation environment is given below:

- 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, the code automatically calculates the range of the map in the x and y directions. Figure-1 shows the left, right, top and bottom boundaries in world coordinate units.
- The drone's starting x, y position is randomly generated according to the uniform distribution. Thus, the state vector $\mathbf{x} = [x, y]^T$ in randomly generated on each iteration.
- At each time step, the simulator draws a circle of radius 10 pixels along with text "Drone" overlaid on the image so that the user can see the true position of the drone. The user can press any key on the keyboard to advance the simulation.
- The simulator simulates an RGB image reading, $\mathbf{z} \in \mathcal{R}^{m \times m \times 3}$, given the drone's true position. The simulator adds some noise to the observed image. The observation image is of size $m \times m$ which can be selected by the user. Several tests were done with m in range of 25 and 100. In this example, we chose aperture size m = 40.
- At each time step, the simulator generates a random movement vector in the x and y direction, described as $[dx, dy]^T$ such that $dx^2 + dy^2 = 1.0$.

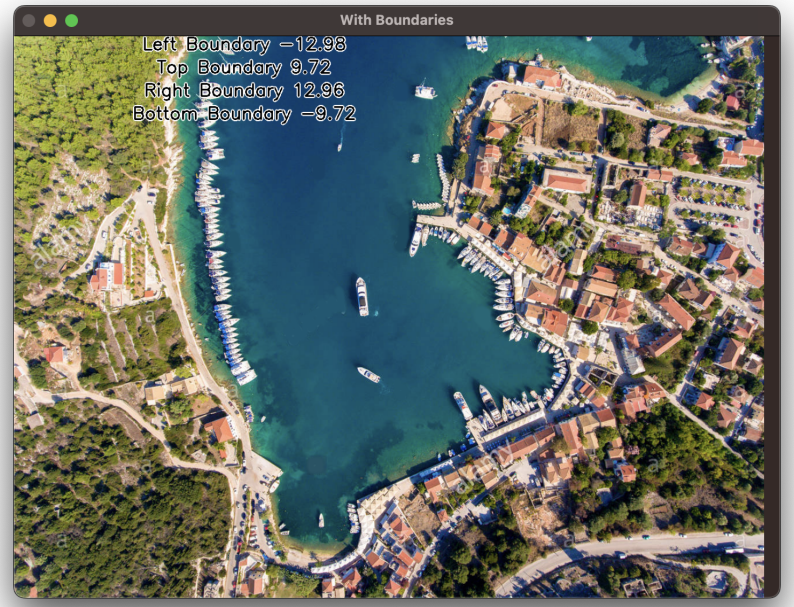


Figure 1: Boundary values overlayed on the map

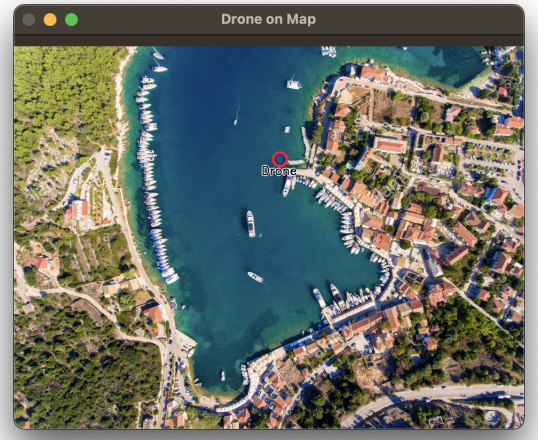


Figure 2: Drone overlaid on the map in a randomly generated position

Figure 3: Observation image from the done (with added noise) with dimension mxm . Here m = 20

The vector is then multiplied with a scalar variable *speed* that users can choose to control the speed. The velocity that takes the drone off of the map are rejected and replaced with a different velocity vector. This movement vector is known to the agent using the particle filter to localize against the map.

- The actual position x_{t+1} is 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.). Variance for the movement noise is selected to be 0.1. This noise data is not shared with the agent implementing the particle filter. Thus only the movement vector (i.e., intended displacement) before noise is applied is shared with the agent.

  Figure-4 shows a sequence of randomly generated velocities in terms of dx and dy in one of the trials. The drone moves along the generated coordinates with some added noise in the position. and Figure-5 shows the actual drone position in the 10 time steps.

2 Particle Filter Implementation

For the particle filter implementation we do that in two steps: 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 (added with some randomly generated noise).

Following, are the guidelines:

```
mdahal01@Milans-MacBook-Pro Codes % python3 ParticleFilter.py
velocity in terms of dx and dy :  dx = %f, dy = %f (0.37945, -0.3256)
velocity in terms of dx and dy :  dx = %f, dy = %f (-0.2672, -0.4226)
velocity in terms of dx and dy :  dx = %f, dy = %f (0.5, -0.00255)
velocity in terms of dx and dy :  dx = %f, dy = %f (-0.5, 0.00455)
velocity in terms of dx and dy :  dx = %f, dy = %f (-0.4839, -0.12585)
velocity in terms of dx and dy :  dx = %f, dy = %f (0.4497, 0.2186)
velocity in terms of dx and dy :  dx = %f, dy = %f (0.46505, -0.1836)
velocity in terms of dx and dy :  dx = %f, dy = %f (0.4665, 0.1799)
velocity in terms of dx and dy :  dx = %f, dy = %f (0.0207, 0.49955)
velocity in terms of dx and dy :  dx = %f, dy = %f (-0.49425, -0.07575)
```

Figure 4: Printout of velocity data in terms of dx and dy for the drone.

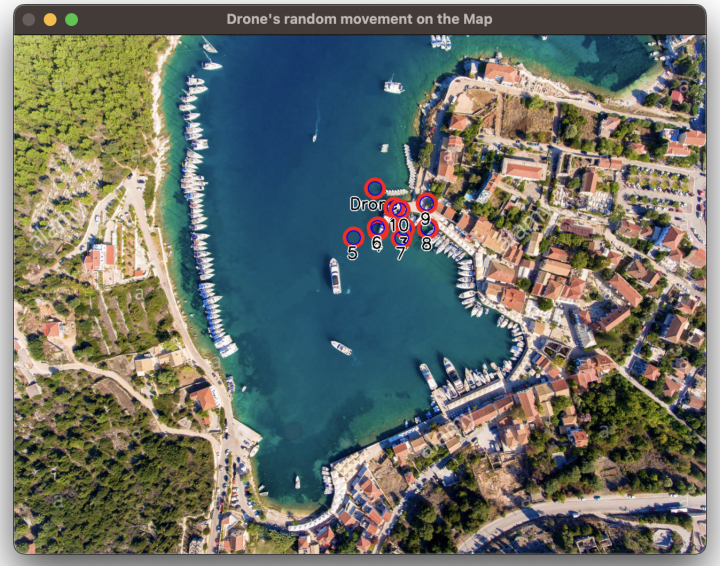


Figure 5: Actual drone positions on the map based on the randomly generated velocity.

1. Initially, we generate a set $\mathcal{P}$ of N particles, uniformly distributed across the map. The N particles are generated within the bounds of the map generated from the simulation section. They are shown with red and blue circle for visibility in both dark and bright areas of the map.

2. For each particle we generate a reference image and calculate the histogram of the pixels. All three colors in RGB are used to generate the histogram. The numbers of bins used to create the histogram for the experiment was 80. Experiments were done with different number of bins.

3. We compare the histograms of the reference and observed image, to calculate the $P(\mathbf{z}|\mathbf{x})$, using correlation. We square the correlation value to get a positive number, and to exaggerate the results. This number that directly corresponds to how similar the image from particle is from the image from the drone, is then normalized across all particles and set as weight for the particle.

4. A roulette wheel re-sampling algorithm is used find the new positions for the particles based on the weights of those particles.

5. Finally, we move each particle according to the known movement vector $[dx, dy]^T$ from the drone movement function, and add some position noise. The particles are redrawn in their new position.

6. At each stage, we show the particles on top of the map image along with the location of the drone. The size of the circles are based on the weight of the particles in that iteration. A large radius indicated the reference and observed image has high correlation.

3 Experiments and Evaluation

The metric I chose to define for my particle filter was the variance of the distances from the centroid of the clusters to the center of the particles. I calculate the centroid of the particles by averaging all the x and y coordinates of the particles. I then calculate the distance between each particle and the centroid using distance formula. I then sort the distances in an ascending order and ignore the highest 50 points. I considered those points to be the outliers, and that helped me localize faster. I then calculated the variance of the distance between the particles and the centroid, which is my metric. I used 0.1 as the cut off point, meaning if the cluster of particle was clustered close enough to have variance of less than 0.1 I considered that the particles had localized itself around the drone.

As you can see in the Figure **??** the filter ran for 5 iterations before the variance fell below 0.1. The distance between the predicted drone position and actual drone position is always reducing.

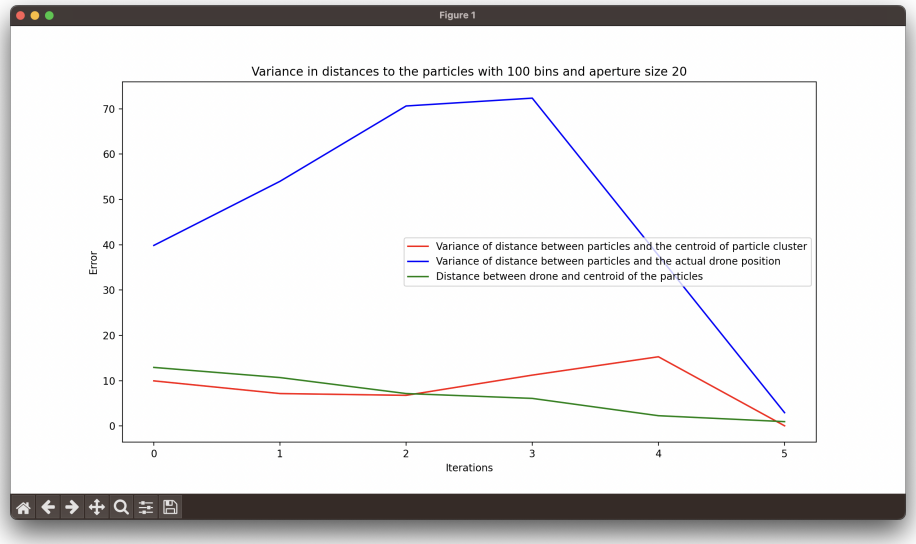


Figure 6: A graph showing how the variance of distance between the particles and drones actual position , as well as distance between the particles and the centroid of particles evolved over time. It also shows the actual distance of the centroid of particles and the actual drone.

3.1 Experiments

I performed many experiments with the particle filter. Checking how number of bins effect the speed of localization or how it minimizes the error, among other things. In this report I would like to report on the effect of aperture size, the pixel width of observation and reference images, on the number of iterations required to localize the drone, and also on how well it can detect the location.

To achieve that I do the following.

1. Create a drone in a random location on the map within the bounds of the map.
2. Create 1000 particles within the bounds of the map
3. Move the drone with dx and dy such that the magnitude of the vector is 1, and add some noise
4. Move all the particles with the same dx and dy
5. Create an observation image of the size, starting with 20 pixels, and add gaussian noise
6. Create reference images of of the same size, i.2. 20 pixels from each of the 1000 particles
7. Create histogram of observation image as well as reference images and find the correlation
8. Using the correlation between the observation and reference images, I normalize them to designate the weight of each particle
9. Use roulette algorithm to resample the particles based on the weights.
10. Calculate the centroid of the particles
11. Calculate the distance from each particle to the centroid of the particles
12. Ignore the outliers (50 largest distances) and calculate the variance in the distance
13. Repeat 3 to 12, as long as the variance is more than 0.1
14. Repeat 3 to 13 for different aperture sizes (40,60,80,100,120,140,160,180 and 200)
15. Calculate the error in terms of error in distance between the actual position of the drone and predicted position from the particle filter divided by the total width and height of the map.
16. Plot the errors as well as number of iteration required to localize, against the number of pixels of the width and height of reference images.

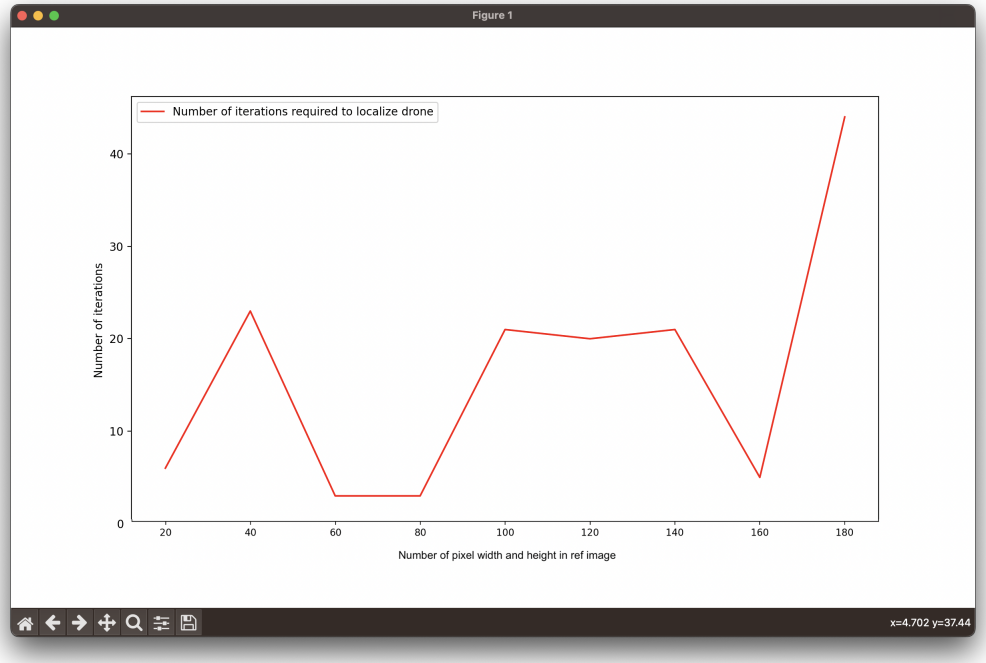


Figure 7: A graph showing how many iterations it took to localize the drone based on the number of pixels of reference image (m)

3.2 Results

Based on the experimentation, it seems that the number of pixels for the reference image should be chosen wisely. Bigger values yields fast localization but even larger values of m take longer and result in more errors. While running the tests (not mentioned in this report) it was also observed that for smaller values of m (20,40 and 60) if the initial particles grouped around a different area of the map, and the drone hovered around similar terrain (blue water or green patch of the map) then it is harder to recover the location.

3.3 Conclusion

Particle Filters is a great way of finding the position of robots in a familiar terrain i.e. if we have a map available. It is great if we have a robot moving around a building with a known configurations, or as done in the exercise, a drone flying over a known terrain. However, it might need help of extra sensors to locate in case of intrusions from external factors that changes the map.

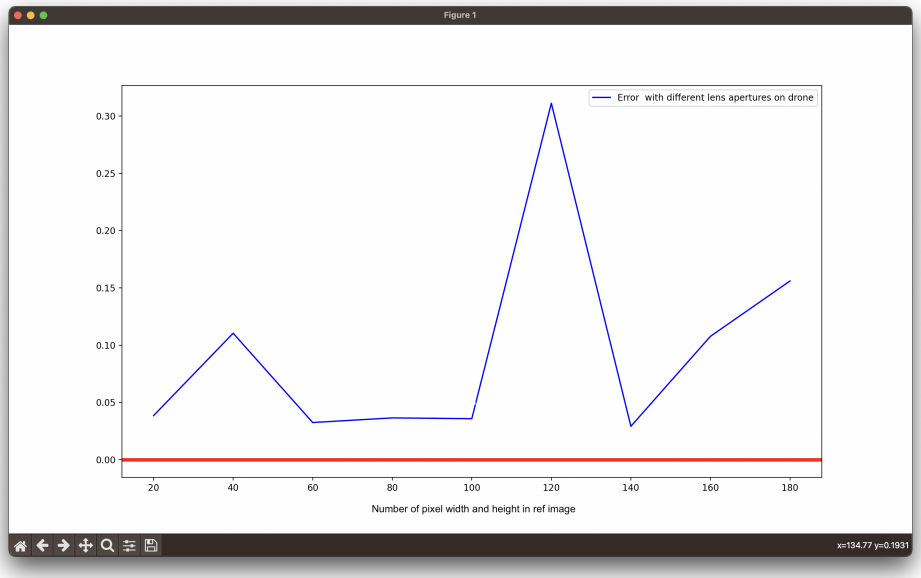


Figure 8: A graph showing errorr in the localization of the drone for different pixels of reference image (m)

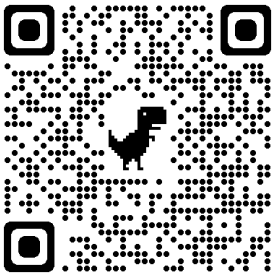


Figure 9: QR code to the github with the codes.

4 Code

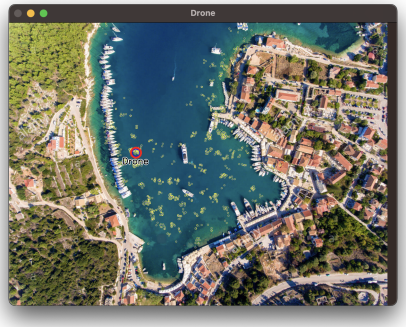
The code for particle filter can be found on
$https://github.com/milandahal213/ProbRobotics/tree/master/HW2$

5 Images from the Experiments

Here is a sample of images from the particle filter showing how the particles were resampled around the position of the drone. As the drone moved randomly around the map the particles followed the drone, resampling more near the drones position. Eventually in 20 iterations the particles were able to minimize the variance of the distance from the centroid of particles to all the particles below 0.1.

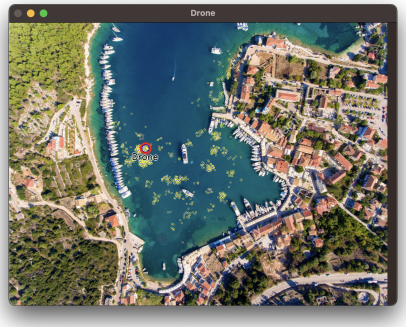
(a) Iteration 1

(b) Iteration 2

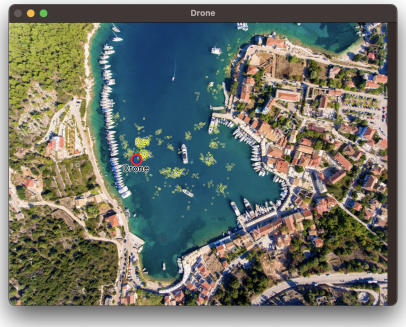


(c) Iteration 3

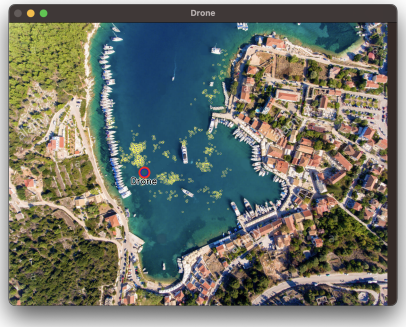
(d) Iteration 4



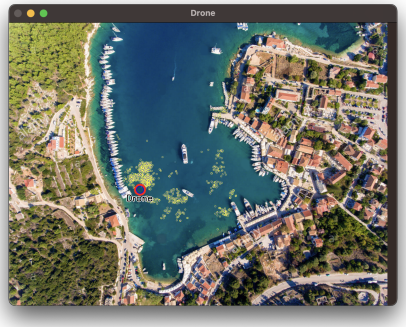
(e) Iteration 5



(f) Iteration 10



(g) Iteration 15



(h) Iteration 20

Figure 10: Pictures showing different steps of implenting particle filter

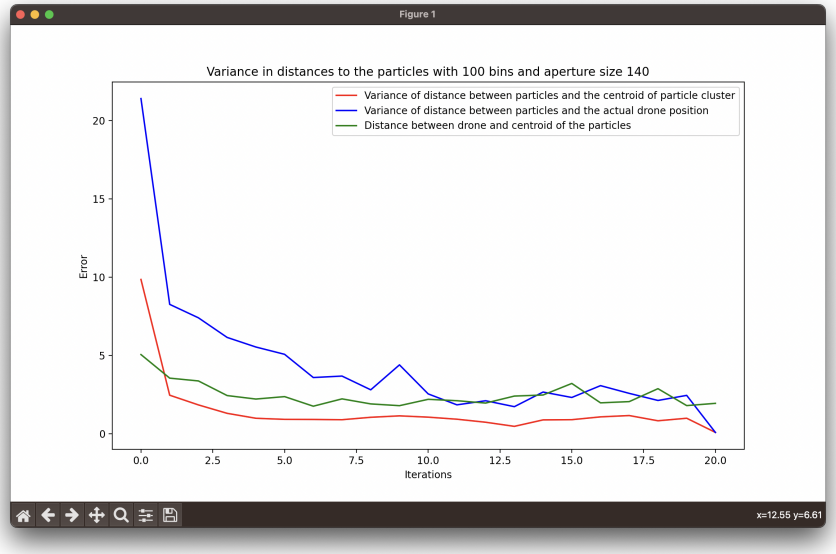


Figure 11: Graph Showing the closing of variance of distance between the particles and the centroid as well as the actual position of the drone.