Travis Clauson
COMP 150 – Probabilistic Robotics
Professor Sinapov

Homework 2: Image-Based Particle Filter for Drone Localization

Assignment: "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."

Initialization: The first step was to create a simulated environment where the drone randomly flies around the given map. To illustrate the simulation, I displayed the actual position of the drone with a blue circle and the camera perspective of the drone with a blue square. I represent the particles with red circles and the algorithm's best guess (the particle with the most weight) with a single yellow circle. Initially, the drone is positioned according to a random uniform distribution, while 1000 particles are positioned evenly and uniformly. We were told in the assignment that 1 unit of distance = 50 pixels, but that does not play a large role in the algorithm.

Drone Movement: At each time step, the drone moves in a random direction, with a distance determined by the velocity of the drone. The default velocity is 1 (50 pixels/step), but this can be edited. Then noise is added in the x and y direction using a gaussian distribution of mean 0 and standard deviation of .1 units (5 pixels) to mimic wind and motor inaccuracies.

deltaX/Y = random.uniform(0,2pi)
*noiseX/Y = random.gaussian(0, .1*50)*

Image Comparison: Next, the drone takes a photo of the ground below it with a size of 50x50 pixels. Then, the algorithm compares it to a reference photo from the perspective of each particle. I used the cv2.compareHist() function from the Open CV library to compare the images. It outputs a similarity score by comparing pixels of each image using separate bins for each color intensity in the RGB spectrum. Finally, this similarity is converted into a weight for each particle.

Particle Resampling: At the beginning of the next timestep, the drone moves, and particles are resampled. This time though, the particles are spawned based on the normalized weights of each particle with replacement. Then, each particle is moved with a gaussian distribution of mean 0 and 10 pixels. This step is crucial because it scatters particles that were sampled from the same location. Reference images of each particle are again compared to the drone's photo, and the cycle repeats.

```
newParticles = numpy.random.Generator.choice(oldParticles, p=weights, size = (N,))
for particle in newParticles:
        particle += deltaX/Y
        particle += random.gaussian(0,.1*50)
```

Localization: The drone is considered localized when 90% of the particles are within a 75 pixel radius.

Tuning the Algorithm: The algorithm can be most efficiently tuned with the Filter Speed parameter(shown below). This tuner increases the relative weight of similar photos and decreases the weight of images deemed not similar. Most importantly, it speeds up the algorithm. However, if the value is too high, the filter moves too fast and may lose track of the drone. Using visual inspection, I picked a value of 1.5. Other variables such as Drone Image Size, Drone Speed, Total Particles, can also be edited to alter the algorithm.

*ParticleWeight[i] = img_comparison(droneImage, particleImage[i])***filter_speed***

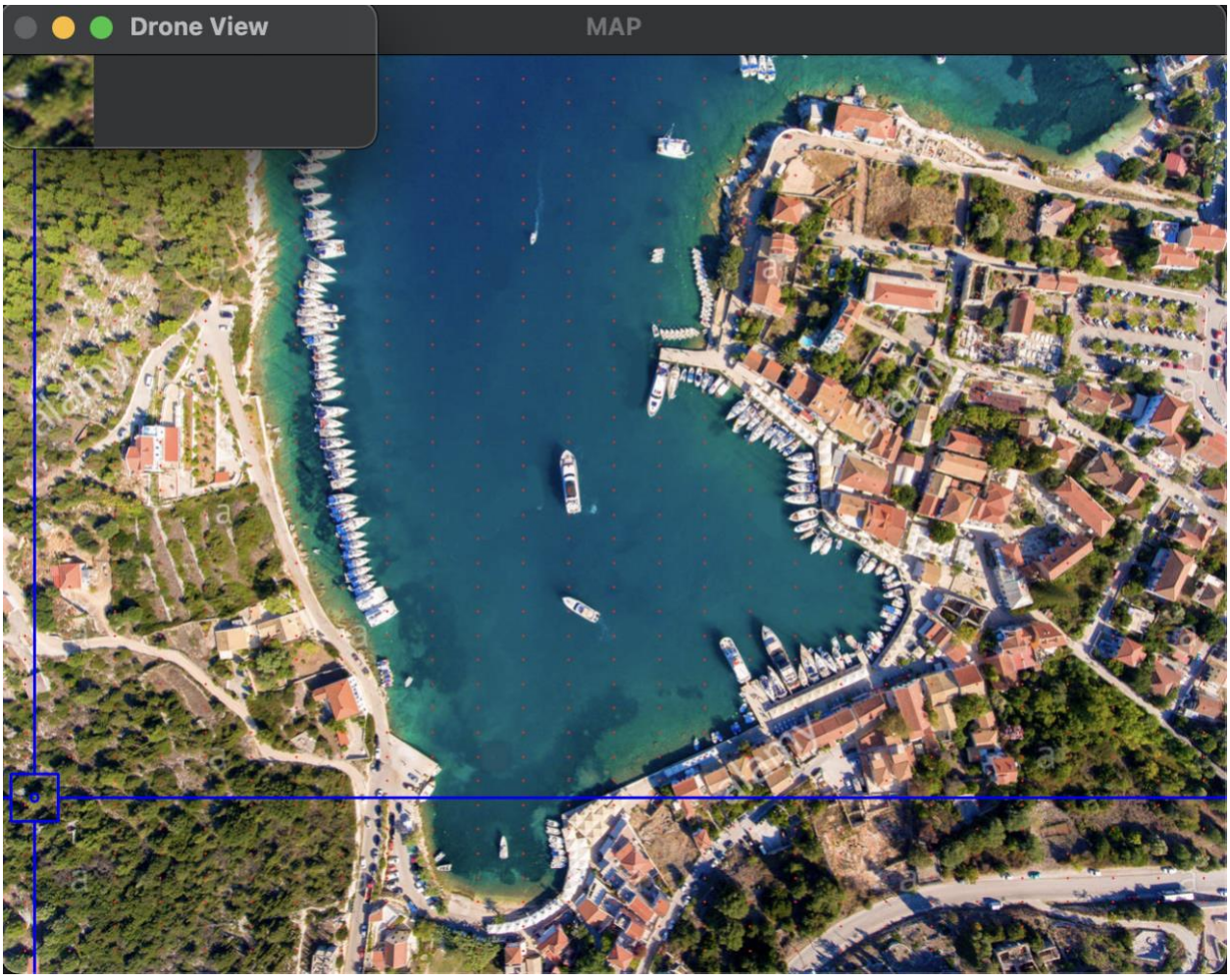

Figure 1: Uniformly Distributed Particles at t=0

Figure 2: Particles at t=2

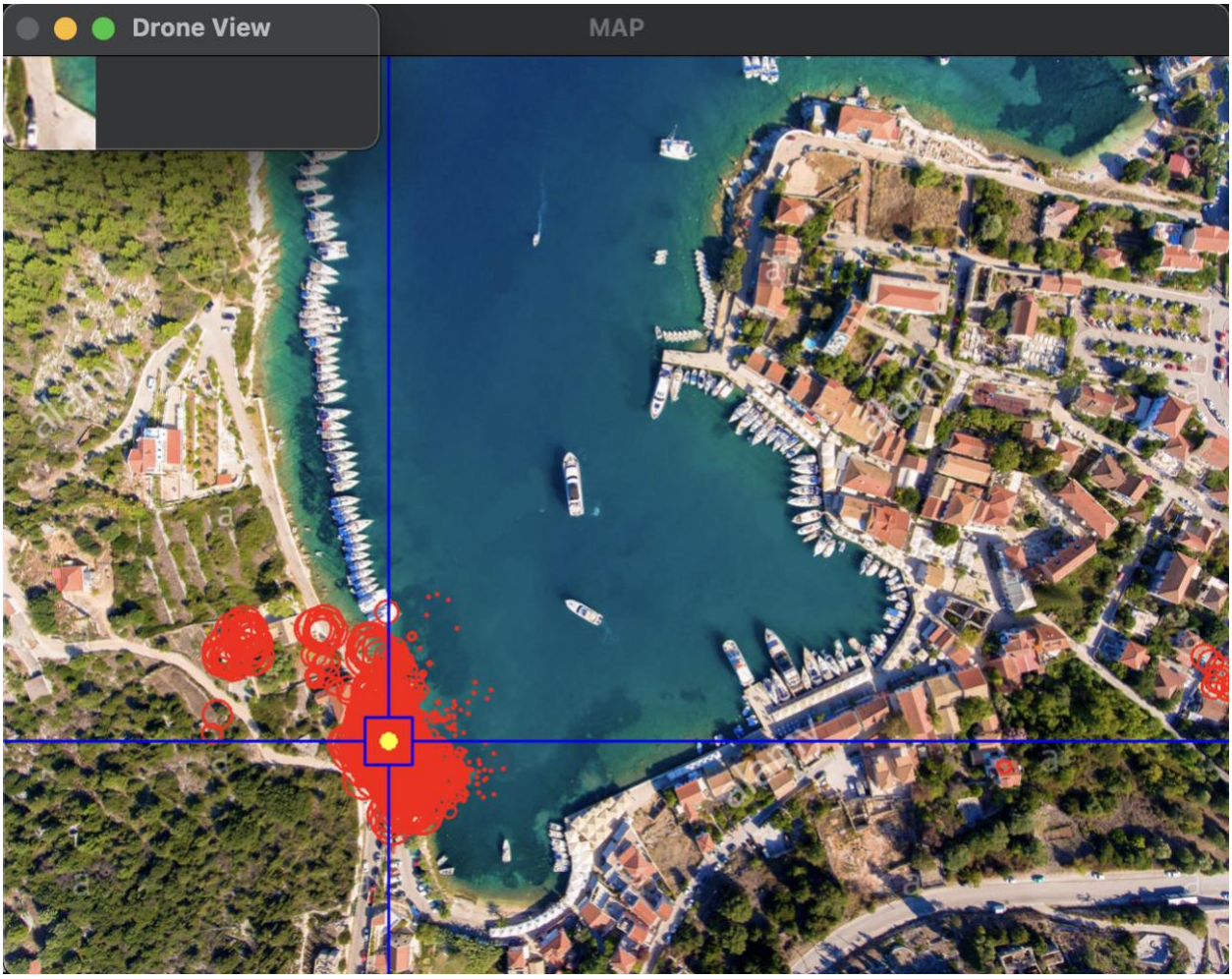

Figure 3: Localization at t=11; Error < 1px

Experiment and Evaluation: This experiment attempts to determine how the speed of the drone effects the time to localize and the average error. The drone has 'localized' when 90% of the particles are within a 75-pixel radius. The 'error' is calculated as the distance in pixels from the best matching particle to the real position of the drone at the time of localization. For each speed, I ran 20 simulations and reported the mean.

Drone Speed (pixels/timestep)	Time Steps to Localize	Average Error at the time of Localization
25	30.95	2.55
50	20.4	2.21
100	12.7	5.57
200	11.3	9.56

When I altered the drone's speed, the particle filter algorithm demonstrated a tradeoff between efficiency and accuracy. The algorithm was more efficient when the drone moved faster because the photos conveyed more new information at each time step. However, its pinpoint accuracy was reduced since there were fewer time steps to zero in on the exact position.