Alexander Hay
alexanderhay2020@u.northwestern.edu

ME 469 - Machine Learning and AI in Robotics

Assignment #0 - Filtering Algorithms
Particle Filter - Dataset 1

10/14/2019

1. Motion model

Inputs: t - time (s)
v - velocity (m/s)
ω - angular velocity (rad/s)

Outputs: angle (rad)
x displacement (m)
y displacement (m)

Parameters: noise in the system (ie. slip, encoding, etc.)

Equations:

$\theta = \omega * t$ Eq. 1

$x = v * t * cos(\theta)$ Eq. 2

$y = v * t * sin(\theta)$ Eq. 3

The model is not linear, it uses nonlinear functions to determine its position.

2. Motion model plot

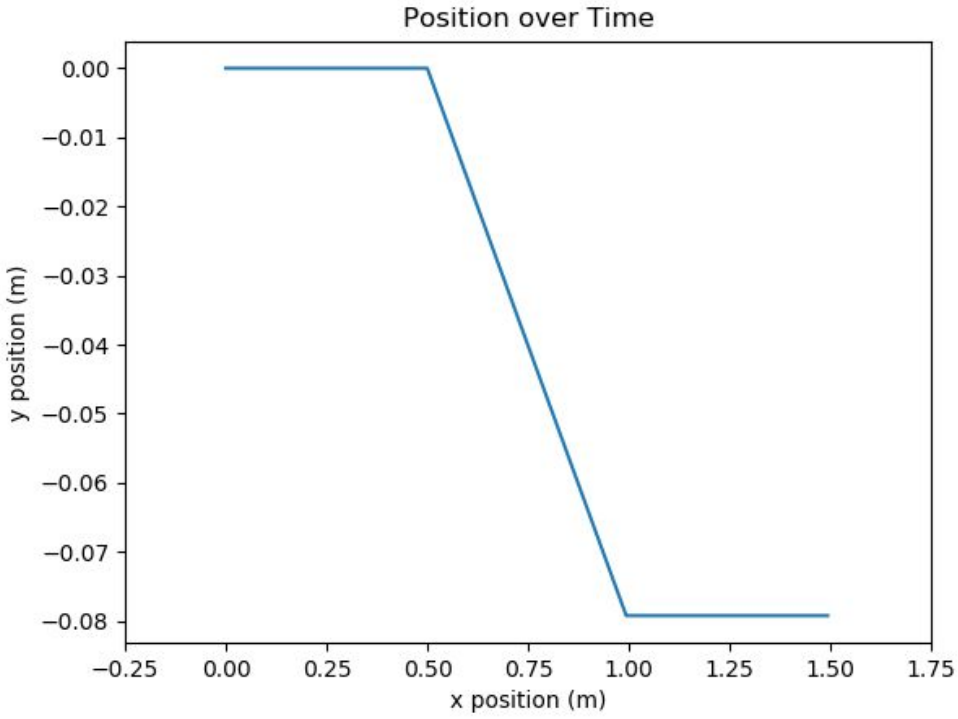

Figure 1: Motion model plot for step 1

3. Applied motion model

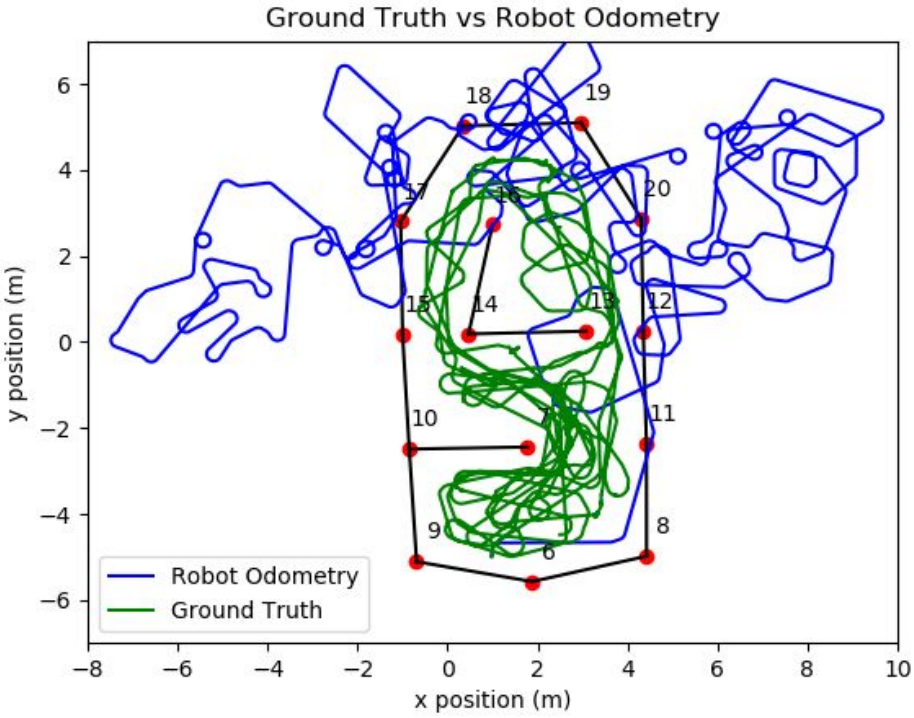

Figure 2: Applied motion model for step 3

Discussion: It's clear that relying on the odometry data by itself presents some clear issues. Early in the robot's path we see that it is already significantly off about where it is versus where it think it is, showing that even small variations early on cause errors to propagate wildly later on. Slippage, encoders, and environmental factors all play a role in how that error is introduced into the system.

4. The Particle Filter (PF) is a nonparametric filtering algorithm that approximates a posterior belief. It does so by using a set of *m* random samples, or particles, each representing a belief. The denser the collection of particles is in a region, the more likely the true state is also in the region. Each particle is given a standardized weight, which is incorporated once a measurement is taken. If the measurement taken is close to what the particle thinks the measurement will be, the particle is given a higher weight. Conversely, if the measurement is not close to what the particle thinks it will be, it is given a lower weight. Particles with low weight can introduce error into the system. To counteract this, a resampling function is implemented where low weight particles are replaced with duplicated high weight particles. It's clear that accuracy increases with the number of particles, however that also increases computational time, which depending on the implementation of the filter, can be a significant factor.

Particle Filter Algorithm (X_{t-1}, u_t, z_t)

1. Generate Particle Set [M]
2. for m = 1 to M do:
3. $P(x_t^m \mid x_{t-1}^m, u_t)$
4. $w_t^m = P(z_t \mid x_t^m)$
5. $*X_t = *X_t + \langle x_t^m, w_t^m \rangle$
6. /for
7. for m = 1 to M do:
8. i = random integer between 1 and M
9. $X_t = X_t.append(x_t^i)$
10. /for
11. return X_t

1. Particles are generated randomly using the initial belief
2. Begin loop
3. Given the previous state and current control, find its current state. In this exercise u_t is comprised of a timestamp, velocity, and angular velocity. The motion model for interpreting that data is given in question 1.
4. A weight is assigned to each particle based on the measurement (sensor) reading, z, and the particle's estimation of the measurement reading, *z. In this exercise, the measurement is comprised of a range and bearing while looking at a known landmark.

$$\text{Range} = \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2}, \qquad \text{Eq. 4}$$

$$\text{Bearing} = arctan\left(\frac{y_i - y_t}{x_i - x_t}\right) - \theta_t \qquad \text{Eq. 5}$$

where (x_t, y_t) is the robot's location and (x_i, y_i) is the landmark's location. In this exercise the weight is calculated using the following formula:

$$\text{wt} = average\left(\frac{1}{abs(z_{t,\,range} - z_{i,\,range})} + \frac{1}{abs(z_{t,\,bearing} - z_{i,\,bearing})}\right) \qquad \text{Eq. 6}$$

5. The intermediate state, given the set of particles and their weights
6. End loop
7. Begin resample loop
8. Find a random integer between 1 and the number of particles (M)
9. Picks a particle from the set using the random number generated
10. End loop
11. Return state

5. The measurement model used was calculated using Equations 4 and 5. For range, it takes the square root of the difference in x, squared, plus the difference in y, squared. For bearing, it uses the arc-tangent function of the difference in y over the difference in x, minus its current bearing.

6. Applied measurement model:

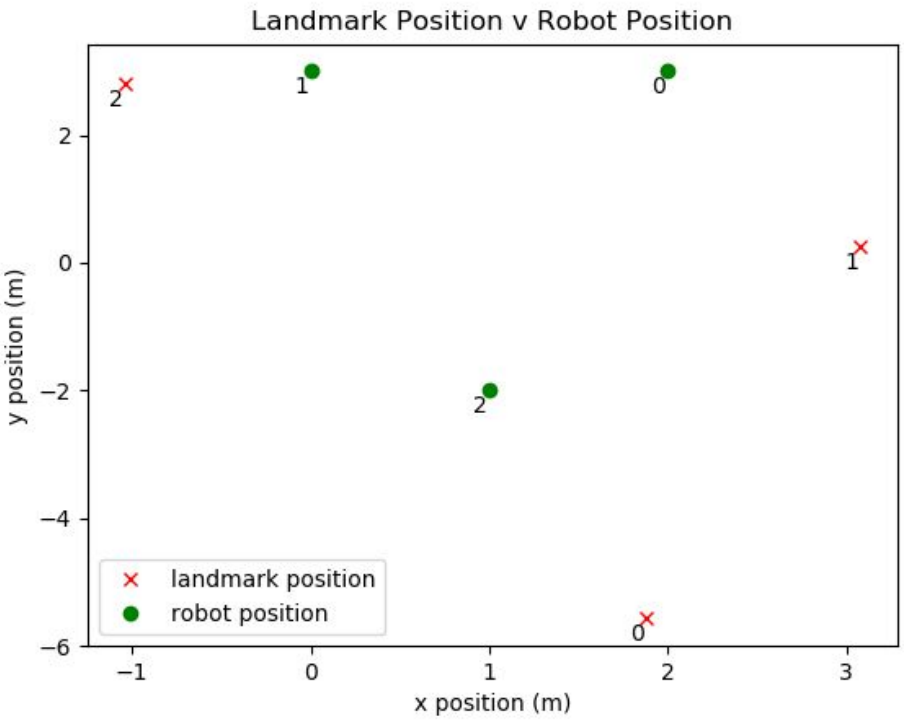

Figure 3: Applied sensor model for step 6

```
point 0
range: 8.573130405567785 (m)
bearing: -1.5847560464461656 (rad)
point 1
range: 4.129145351091766 (m)
bearing: -0.7290156012301658 (rad)
point 2
range: 5.216301745218219 (m)
bearing: 1.972918786232824 (rad)
```

Figure 4: Terminal output of the sensor model for step 6

What is important to note here is the terminal output of the model rather than the graph. What the output is saying is that at the robot's given points, here is the robot's range and bearing from the given landmark. Marking out the range and bearing on the graph can be left for future updates.

7. Implement the filter

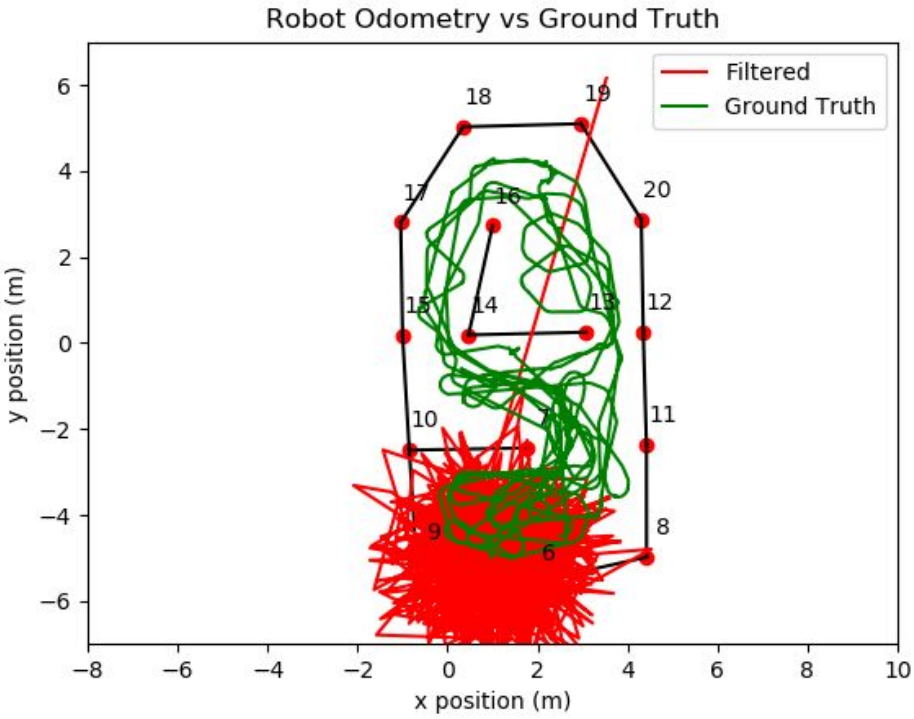

Figure 5: Position data plotted while implementing the filter

8. The filter was not implemented successfully. What is clear is that the particles did not move with the odometry data. In the filter code itself the particle set is passed to the motion model function via a loop, and returns a new state for each particle that is then placed back into the particle set. From there it is passed to a sampler function that returns the state of the chosen particle. The math, in theory, is sound and follows the algorithm, but the implementation into Python is not. Runtime is also *incredibly* slow. When comparing runtimes with other classmates I am led to believe that I am running another loop that others are not, making mine exponentially slower.

   The first loop runs for the length of the odometry data, which in this exercise is 11,524 iterations. It emulates the robot collecting odometry data. The second loop, within the first loop, iterates for every particle, which in this exercise I have set to 1,000. The second loop passes each particle through the motion model, Line 3 of the Particle Filter Algorithm. This is a parameter that can be tuned for performance. The third loop, which runs within the first loop as well, iterates for every weight, which is also 1,000 since there are 1,000 particles. The third loop however is nested within an if statement, so that it only resamples if a measurement has been taken.

   Another factor is my own ability to put the algorithm into code. As a novice programmer, especially to Python, I have more confidence that this is the case. Lines 4, 5, and 9 proved to be the most challenging which leads me to believe that I did not complete those steps successfully. A missed indent or tweak may be the cause of the long runtime or bad plot data. It could also be passing the wrong type of data to each function, despite completing part A of the assignment correctly. As in, putting the data was put into an array, passing to

a function, making sure the function returned the correct data in the correct format and shape, all of that proved to be a challenge in its own right and became a classroom for debugging.

9. The filter can be computationally expensive. Changing the number of particles can increase computation speed, but at the cost of accuracy of the filter. Noise is also a factor. It can be addressed by using more particles, but again, at the cost of computational speed. Noise can also be addressed outside of the filter by creating a better environment for the robot to operate in, or using motors/encoders that respond to input better. This can't always be the case but they are things to consider.

   Regarding sensor data. Lines 227-230 of the Python code were to address the case if the robot ever received more than one sensor reading. The goal was to get the weights of both measurements, or of however many there were, and pick the best one (best average, or best weight). It would find these measurements by grabbing the timestamp of the sensor measurement last then compare the timestamp with any other readings that came before it. Any sensor readings that matched would be added to an array and passed along to the weight function. Almost none of the code exists; as I was hashing out how to go about doing that I figured it would be best to have a working filter to test on. That is why it pulls the last sensor reading.

   Regarding noise. None of my code implements any noise. It is something that can be tuned and adjusted but I never implemented it. Given the difficulty of getting the filter to work properly and my experience in Python, I decided to focus on creating a working filter to tune than possibly implementing the noise wrong and not having a working filter.

Bibliography

"Robotic Perception." *Artificial Intelligence: a Modern Approach*, by Stuart J. Russell et al., Prentice Hall, 2010.

"Nonparametric Filters." *Probabilistic Robotics*, by Sebastian Thrun et al., MIT Press, 2010.

Leung K Y K, Halpern Y, Barfoot T D, and Liu H H T. "The UTIAS Multi-Robot Cooperative Localization and Mapping Dataset". International Journal of Robotics Research, 30(8):969–974, July 2011.