16833 - Robot Localization and Mapping
Fall 2023
Homework - 1
Robot Localization using Particle Filters

Name	Andrew id
Akshay Badagabettu	abadagab
Sai Sravan Yarlagadda	saisravy

Introduction

A particle filter is a probabilistic filtering technique used for state estimation in dynamic systems especially when the model states are non parametric. Particle filters cannot make assumptions regarding the functional form of the probabilistic distributions it represents. One of the main advantages of particle filters is that it can be applied even to multimodal distributions unlike Kalman filters. In this report, we will go through our approach, the challenges we faced, the decisions we took, and the reasons behind taking those decisions.

Approach

In this homework we performed localization using Monte Carlo Localization on a lost indoor robot.

The approach is divided into three main steps.

1. The Prediction step
2. The Correction step
3. Resampling step.

The prediction step updates the particle's state dynamically and predicts the future state of the particles. The correction step filters the particles based on the importance weights assigned to the particles. By correcting particles based on measurements, the particle filter is robust to changes in the system's behavior or any other external disturbances. It adapts to the new information, making it suitable for tracking and estimation in dynamic and uncertain environments. It also helps in the convergence of the particles to the true state of the system. Finally in the resampling step we use the weights that were assigned to the particle in the correction step. During the resampling step particles with higher weight get resampled in larger numbers when compared to the particles with smaller weights.

Final stage in solving this problem included tuning parameters such as number of particles, process noise and measurement noise based on the filter's performance and convergence.

Implementation and tuning

Before we discuss the main elements of the particle filter algorithm, we will briefly discuss the particle initialization. For all of our runs, we kept the number of particles as 500. This is because we did not feel the need to increase the number of particles except for a few times when the particles converged at the wrong location. More particles would reduce the chances of this happening. The function given to us produces random particles over a part of the map. These locations include places that are outside the hallways of Wean Hall. These particles are useless and they take much longer to converge. So, we wrote a function that initializes particles such that they are only inside the hallway. In other words, places on the occupancy_map where probability is zero. This made our model converge considerably faster. The 4 main elements of the particle filter are explained below.

1. Motion Model

The algorithm for the motion model was taken from [1]. The Odometry Motion Model was chosen over the velocity motion model as we are provided with odometry measurements and the model uses it directly as a basis for computing the next pose of the robot.

Algorithm sample_motion_model_odometry(u_t, x_{t-1})**:**

$$\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$$
$$\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$$
$$\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$$

$$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta^2_{\text{rot1}} + \alpha_2 \delta^2_{\text{trans}})$$
$$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3 \delta^2_{\text{trans}} + \alpha_4 \, \delta^2_{\text{rot1}} + \alpha_4 \, \delta^2_{\text{rot2}})$$
$$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta^2_{\text{rot2}} + \alpha_2 \delta^2_{\text{trans}})$$

$$x' = x + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}})$$
$$y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$$
$$\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$$

return $x_t = (x', y', \theta')^T$

Fig. 1 - Motion model algorithm

The algorithm is quite straightforward to implement. Sampling is done from a Normal Distribution which has zero mean and the value of the variance is the term in the bracket. In other words, sample(variance). There are four alpha values that need to be tuned here. The method for tuning that we adopted was to create 500 particles at the same point in the map and run it through the motion model. The way the particles moved were noted. The particles after a few iterations formed a 'banana' shape for the finalized alpha values. We kept playing with the alpha values until we achieved the 'banana' shape. We also plotted and noted the movement of these particles on the map itself to see if there was too much noise (moves too aggressively) or too little noise added (moves too slowly). Final alpha values were alpha1 = 0.00005, alpha2 = 0.00005, alpha3 = 0.0001, alpha4 = 0.0001.

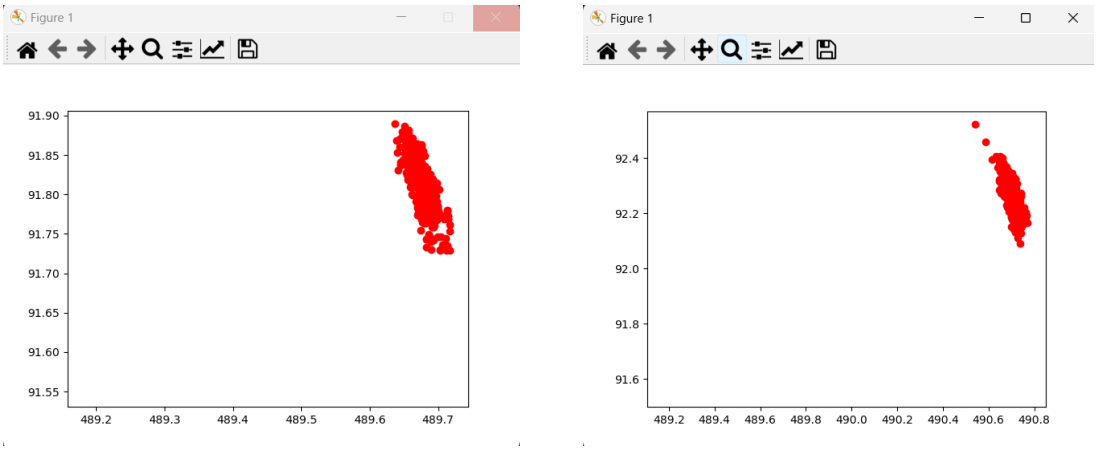

Fig. 2 - Particles after 70 and 100 time steps respectively

2. Ray Casting

```
for i=1 to 180 do
        Xnew = x_start
        Ynew = y_start
        Prob_of_finding_obj = 0
        while(Prob_of_finding_obj < min_req_prob) do
                X_new = X_new + stride * cos(theta + i in radians)
                Y_new = Y_new + stride * sin(theta + i in radians)
                X_new_pass = X_new
                Y_new_pass = Y_new
        if X_new_pass or Y_new_pass lies outside map dimensions do
                get Prob_of_finding_obj at X_new_pass and Y_new_pass
        else do
                X_new_pass = X_start
                Y_new_pass = Y_start
        if X_new_pass != X_start do
                return Euclidean Dist of (X_new_pass, Y_new_pass) & (X_start, Y_start)
        else do
                return distance =1
```

As explained in the pseudo code above the robot was made to cover a total range of 180 degrees. So the robot was made to turn 1 degree in every iteration for 180 times and was made to check the distance to the boundary. We calculated the euclidean distance by incrementally giving a constant stride to the robot and checking the probability of finding an object. If the probability of finding an object is greater than the minimum probability, we get the coordinates of the point and calculate the distance from the start point. For validation, ray casts were plotted onto the map and those images were shown in the figures below. The red point is the robot location we initialized to test the ray cast. The blue points are the laser scan end measurements.

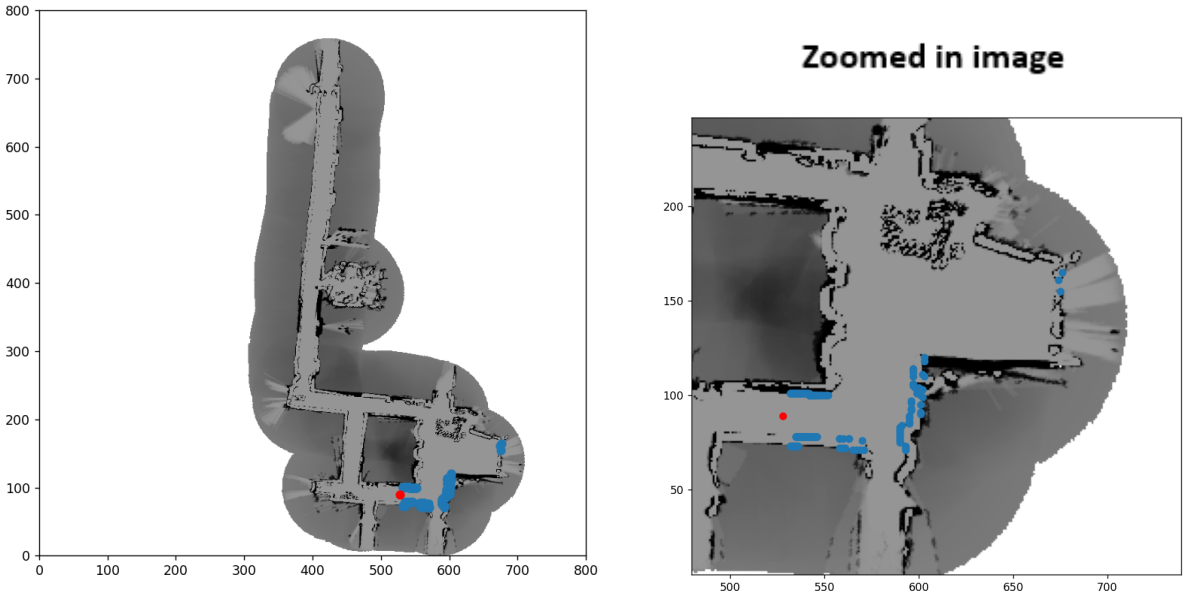

Fig. 3 - Ray cast measurements at Robot Location: x_t1 = [5286, 887, 0]

3. Sensor Model

The sensor model is essential to the model as the particle weights are computed in this step and the resampling takes place with respect to these weights. The model used to compute these weights takes into consideration 4 types of measurement errors.

a. Sensor measurement error

No sensor is perfect in giving out the reading. Error in the sensor come from varying factors, some of which include temperature effects, type of the wall (for lasers) etc. This uncertainty is computed by using a narrow gaussian with mean as true range of the object (computed by ray cast) and standard deviation as sigma_hit (which should be tuned).

$$p_{\text{hit}}(z_t^k \mid x_t, m) = \begin{cases} \eta\, \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) & \text{if } 0 \le z_t^k \le z_{\max} \\ 0 & \text{otherwise} \end{cases}$$

where z_t^{k*} is calculated from x_t and m via ray casting, and $\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)$ denotes the univariate normal distribution with mean z_t^{k*} and standard deviation σ_{hit}:

$$\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) = \frac{1}{\sqrt{2\pi\sigma_{\text{hit}}^2}}\, e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{\text{hit}}^2}}$$

Fig. 4 - p_hit computation

b. Unexpected objects

The environment in which the robot operates in is not static and there can be various obstructions that come up in front of the reading unexpectedly. Because of this, some scans will produce very short ranges which cannot be correlated with the map. The probability of range measurements in this case is described by an exponential distribution.

$$p_{\text{short}}(z_t^k \mid x_t, m) = \begin{cases} \eta\, \lambda_{\text{short}}\, e^{-\lambda_{\text{short}} z_t^k} & \text{if } 0 \le z_t^k \le z_t^{k*} \\ 0 & \text{otherwise} \end{cases}$$

Fig. 5 - p_short computation

The lambda value needs to be tuned.

c. Failures

Sometimes a measurement is say 1000 cm but the laser scan has a maximum range z_max = 200 cm. The laser will return a value of 200cm. This probability is described by an indicator function as shown below.

$$p_{\max}(z_t^k \mid x_t, m) = I(z = z_{\max}) = \begin{cases} 1 & \text{if } z = z_{\max} \\ 0 & \text{otherwise} \end{cases}$$

Fig. 6 - p_max computation

d. Random measurements
   Sometimes sensors produce random readings which don't really have an explanation and the probability for this uncertainty can be described by a uniform distribution spread over [0, z_max].

$$p_{\text{rand}}(z_t^k \mid x_t, m) = \begin{cases} \frac{1}{z_{\max}} & \text{if } 0 \le z_t^k < z_{\max} \\ 0 & \text{otherwise} \end{cases}$$

Fig. 7 - p_rand computation

The final probability is calculated by the vector multiplications of these 4 probabilities by their mixing parameters. These mixing parameters also need to be tuned.

$$p(z_t^k \mid x_t, m) = \begin{pmatrix} z_{\text{hit}} \\ z_{\text{short}} \\ z_{\max} \\ z_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k \mid x_t, m) \\ p_{\text{short}}(z_t^k \mid x_t, m) \\ p_{\max}(z_t^k \mid x_t, m) \\ p_{\text{rand}}(z_t^k \mid x_t, m) \end{pmatrix}$$

Algorithm beam_range_finder_model(z_t, x_t, m)**:**

$q = 1$
for $k = 1$ *to* K *do*
  compute z_t^{k*} *for the measurement* z_t^k *using ray casting*
  $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
    $+ z_{\max} \cdot p_{\max}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
  $q = q \cdot p$
return q

Fig. 8 - Sensor model algorithm

All the above values where it was mentioned that tuning was required, the model was run all the time by changing one parameter at a time and results being noted.
From the robotmovie1.gif, we saw that there were a lot of people walking around the robot. Due to this, we increased the values of lambda_short and z_short. The unexpected objects noise is quite dominating in this scenario. The z_hit and sigma_hit values were fine tuned by running the model itself. Instead of multiplying all the probabilities, which would lead to numerical stability issues, the log of the 'q' values were calculated and summed. The exponent of the final 'q' value was then returned.

4. Resampling

```
1:    Algorithm Low_variance_sampler(X_t, W_t):
2:        X̄_t = ∅
3:        r = rand(0; M^-1)
4:        c = w_t^[1]
5:        i = 1
6:        for m = 1 to M do
7:            U = r + (m − 1) · M^-1
8:            while U > c
9:                i = i + 1
10:               c = c + w_t^[i]
11:           endwhile
12:           add x_t^[i] to X̄_t
13:       endfor
14:       return X̄_t
```

Fig. 9 - PseudoCode of low variance sampling

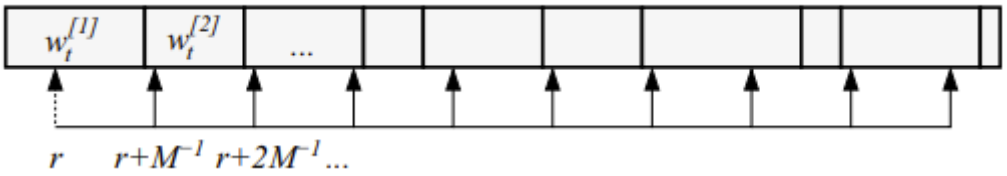

Fig. 10 - Principle of low variance sampling procedure

In this pseudo code rather than choosing particles with random indexes, we get a random number and select samples based on this number and a probability which is proportional to the weight. Initially we start by normalizing the weights and then getting a random value between 0 and 1/(number of particles). At the start c is made equal with the first particle weight. And then U is made equal to the sum of the random value and (m - 1)/(number of particles), where m lies between 1 and number of particles. When the value of U is greater than that of c, c gets added to the next weight. Only when c becomes greater than U we add the particular importance and its corresponding state vectors to a list and return that list.
Particles with higher weight have more probability of being resampled in the next iteration. One of the main advantages of low variance sampling is that it reduces particle degeneracy in particle filters.

Discussion of Performance

We compared our results to the robotmovie1 file provided and the results achieved had a close resemblance to the expected results.

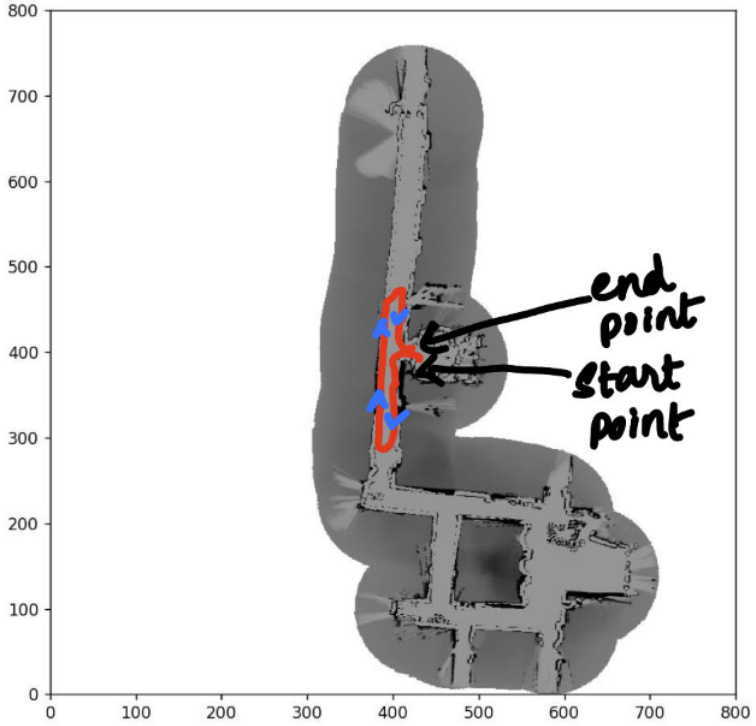

Fig. 11 - Path followed by the robot according to robotmovie1.gif

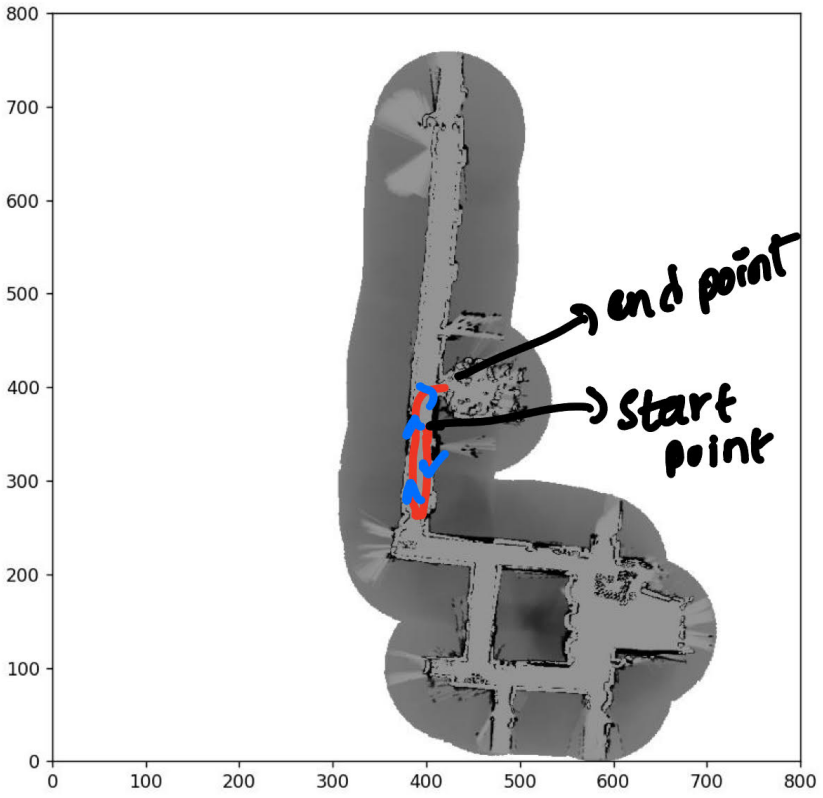

Fig. 12 - Path followed by the robot according to the results we obtained for log-4

Results

The results can be visualized by clicking on this link https://youtu.be/ZRo8cZPze0w
We ran the model log file- 1 and log file-4 (robotdata1 and robotdata4). The video is sped up by 96x.

Future work

1. While running the particle filter simulation, we observed that sometimes, as the particles kept getting resampled, and all particles were converging to one location, it was not at the same spot every time we ran it. Few times, the point that the particles converge to was so far away from the ground truth that the model could never recover from it. The motion model developed for this system cannot make large enough strides to recover from this. In some rare cases, the particles would converge in a completely different alleyway of Wean Hall and in this case the particle is truly stuck. Some work can be done in this area to completely remove wrong convergence points from ever happening. In our case, we knew where the robot was because of the gif provided but for the second log that we had to run (robotdata4.log in our case), we had no idea if the robot was moving correctly. The way we decided which run to keep was to do multiple runs and the robot movement that was the same for most of the runs was finalized. This is extremely time consuming, cumbersome, and most of all expensive.
2. The particle filter algorithm is computational intensive. For real-time implementation, especially if we are getting sensor readings in intervals as low as tenths of seconds, we would need to optimize our model a lot along with having powerful computers onboard. Future work can be done to find better algorithms for mainly the sensor model.

References

1. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005
2. https://www.cs.cmu.edu/~16831-f12/notes/F14/16831_lecture05_gseyfarth_zbatts.pdf