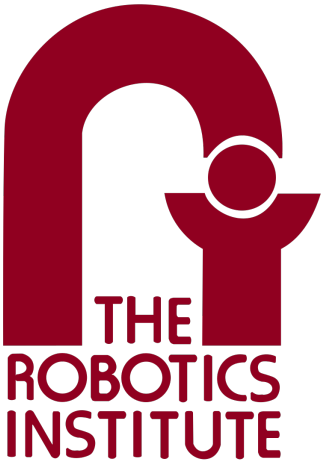
Homework 1



Robot Localization using Particle Filters

16-833A Spring 2025

Author: **Boxiang (William) Fu**
Andrew ID: boxiangf
E-mail: *boxiangf@andrew.cmu.edu*

Author: **Joshua Pen**
Andrew ID: jpen
E-mail: *jpen@andrew.cmu.edu*

February 6, 2025

Contents

1 Overview

This report details the implementation of a particle filter for the localization of a robot in Wean Hall. The algorithm is a realization of the Bayes filter using a large number of particles to represent likely robot states. It includes a prediction step for updating particle poses using a motion model; a correction step for computing importance weights for each particle using a sensor model; and a resampling step to draw particles for the next time step using a resampling process. The following sections details the approach, description, and implementation of each step in turn. Additionally, a discussion of the tuned parameters and the performance of our implementation is discussed. Finally, we present opportunities for further work and potential improvements to our implementation.

2 Motion Model

In the particle filter's motion model, odometry readings from the robot are treated as control signals. By comparing the current and previous odometry readings, we can calculate the relative motion parameters:

$\delta_{\mathbf{rot1}}$: The relative angle of initial orientation and translation direction.

$\delta_{\mathbf{trans}}$: The translation distance the robot has moved between the two time steps.

$\delta_{\mathbf{rot2}}$: the relative angle between the translation direction and the final orientation.

These values were calculated using the following equations using the odometry readings from u_{t0} and u_{t1}:

$$\delta_{\text{rot1}} = \tan^{-1}\left(\frac{u_{t1}[1] - u_{t0}[1]}{u_{t1}[0] - u_{t0}[0]}\right) - u_{t0}[2] \tag{1}$$

$$\delta_{\text{trans}} = \sqrt{(u_{t1}[0] - u_{t0}[0])^2 + (u_{t1}[1] - u_{t0}[1])^2} \tag{2}$$

$$\delta_{\text{rot2}} = u_{t1}[2] - u_{t0}[2] - \delta_{\text{rot1}} \tag{3}$$

To account for uncertainties such as drift and slippage, we add noise to these relative motion parameters. This noise is modeled as zero-mean Gaussian noise, with variances proportional to the magnitude of the respective motion components. The noisy motion parameters ($\hat{\delta}_{\text{rot1}}$, $\hat{\delta}_{\text{trans}}$, $\hat{\delta}_{\text{rot2}}$) are obtained by adding random samples from these distributions to the computed parameters.

Each particle's state from the previous time step is then updated by applying these noisy motion parameters:

Rotate by $\hat{\delta}_{\text{rot1}}$.

Translate by $\hat{\delta}_{\text{trans}}$.

Rotate by $\hat{\delta}_{\text{rot2}}$.

The noisy motion parameters were calculated using the folowing equations using porportional constants alpha:

$$\hat{\delta}_{rot1} = \delta_{\text{rot1}} - \mathcal{N}\left(0, \sqrt{\alpha_1\delta_{\text{rot1}}^2 + \alpha_2\delta_{\text{trans}}^2}\right) \tag{4}$$

$$\hat{\delta}_{trans} = \delta_{\text{trans}} - \mathcal{N}\left(0, \sqrt{\alpha_3\delta_{\text{trans}}^2 + \alpha_4\delta_{\text{rot1}}^2 + \alpha_4\delta_{\text{rot2}}^2}\right) \tag{5}$$

$$\hat{\delta}_{rot2} = \delta_{\text{rot2}} - \mathcal{N}\left(0, \sqrt{\alpha_1\delta_{\text{rot2}}^2 + \alpha_2\delta_{\text{trans}}^2}\right) \tag{6}$$

Where the alphas are the tuning parameters for the model, representing the ratio of noise to amount of movement. The tuning parameters were adjusted to reflect actual randomness in the movement of the robot, without making it too large which would imply the motion of the particle is completely random. We apply these noisy motion parameters to the previous believed particle position with the following equations:

$$x = x_{t0}[0] + \hat{\delta}_{trans} \cos(x_{t0}[2] + \hat{\delta}_{rot1}) \tag{7}$$

$$y = x_{t0}[1] + \hat{\delta}_{trans} \sin(x_{t0}[2] + \hat{\delta}_{rot1}) \tag{8}$$

$$\theta = x_{t0}[2] + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \tag{9}$$

This process yields the particle's new pose (x, y, θ).

By implementing this approach, the particle filter effectively propagates each particle's state through the motion model, incorporating both the intended control inputs (derived from odometry) and the stochastic nature of real-world robot movement. The complete pseudo-code for the implementation is provided in Reference [2] and shown in Figure 1.

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

2: $\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3: $\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$

4: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

5: $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1 \delta^2_{\text{rot1}} + \alpha_2 \delta^2_{\text{trans}})$

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

7: $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1 \delta^2_{\text{rot2}} + \alpha_2 \delta^2_{\text{trans}})$

8: $x' = x + \hat{\delta}_{\text{trans}} \ \cos(\theta + \hat{\delta}_{\text{rot1}})$

9: $y' = y + \hat{\delta}_{\text{trans}} \ \sin(\theta + \hat{\delta}_{\text{rot1}})$

10: $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

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

Figure 1: Odometry Motion Model

3 Sensor Model

For the sensor model of the particle filter, we use the beam range finder model. Given the believed state of the particle, we estimate the expected location of the laser reading—i.e., the point where the laser should hit an obstacle—which we denote as z_{t1s}. Then, using the actual laser range readings from the sensor and employing a combination of a normal distribution p_{z_hit}, an exponential distribution p_{short}, a max range distribution p_{max}, and a uniform distribution p_{rand}, we calculate the probability for each laser measurement. This probability is used to compute the overall likelihood of the laser scan z_{t1}, which corresponds to its weight.

For each of the 180 laser range readings (covering angles from $-90°$ to $90°$), ray casting is performed starting from a point 25 cm in front of the robot's position, which

corresponds to the sensor's location. Some sub-sampling of the lasers is performed to improve computational efficiency. During the ray casting process, the distance from the sensor is incrementally increased along the beam's direction on the occupancy map until an obstacle is detected (or until the probability of an obstacle at that location falls below a predetermined threshold). If the laser beam extends beyond the map boundary, the expected point of impact is set to the maximum range.

Once the expected laser reading for each beam is estimated, the probability for that reading is calculated using a combination of four different probability distributions. The first distribution, p_{z_hit}, represents the probability that the laser reading corresponds to an obstacle hit within the expected range (i.e., between zero and the maximum range). If the reading is out of this range, the probability is set to zero. Otherwise, the probability is computed by multiplying a tuning parameter z_{hit} by the value obtained from the Gaussian probability density function, whose mean is z_{t1s} (the expected laser reading) and whose standard deviation is σ_{hit}, reflecting the deviation of the actual laser reading z_{t1} from the expected value.

The second distribution, p_{short}, accounts for the possibility that the sensor reading is shorter than expected due to an unexpected obstacle. If the sensor laser reading is between zero and the expected laser reading, the probability is calculated as

$$p_{short} = \lambda_{short}\, \exp\left(-\lambda_{short}\, z_{t1}\right),$$

and it is zero otherwise.

The third distribution, p_{max}, represents the probability that the laser reading is at the maximum range. This probability is computed by multiplying a tuning parameter z_{max} by 1 when the sensor laser reading is greater than or equal to the maximum range; otherwise, it is set to zero.

Finally, the fourth distribution, p_{rand}, captures the effect of random measurement noise. If the sensor reading is within the range from zero to the maximum range, the probability is given by z_{rand}/z_{max}; if not, the probability is zero.

Once these probabilities are computed, they are combined—typically by summing their log-likelihoods—to obtain the overall probability of the sensor reading based on the expected sensor reading. Figure 2 shows the pseudo-code of the implementation provided in Reference [2].

```
1:  Algorithm beam_range_finder_model(z_t, x_t, m):

2:      q = 1
3:      for k = 1 to K do
4:          compute z_t^{k*} for the measurement z_t^k using ray casting
5:          p = z_hit · p_hit(z_t^k | x_t, m) + z_short · p_short(z_t^k | x_t, m)
6:              +z_max · p_max(z_t^k | x_t, m) + z_rand · p_rand(z_t^k | x_t, m)
7:          q = q · p
8:      return q
```

Figure 2: Ray Casting Sensor Model

4 Resampling Process

After running the motion and sensor models, the last part of the particle filter algorithm is the resampling process. The particles are resampled according to their importance weights obtained from the sensor model. Briefly summarizing, particle locations that are better aligned with the sensor readings obtain higher importance weights, and so the probability of them being resampled increases. Particle locations that align poorly with the sensor readings obtain low importance weights, thus they are more likely to drop out from the sample.

For particle initialization at time-step 0, the particles were initialized randomly in the free-space of the map with equal weights. A point is determined to be free-space if the occupancy grid at that location was a value of 0. Equal importance weights were assigned as we have no existing knowledge of the prior distribution of where the robot is located. Implementation-wise, we initialized the number of particles at 1500, although this number will dynamically decrease to increase speed and performance.

For the resampling process, two different resampling procedures were implemented. They are the multinomial resampler and the low-variance resampler respectively. For each procedure, a vanilla and dynamic resampler variant were further implemented. The vanilla variant does change the particle count during resampling, whereas the dynamic resampler increases or decreases the particle count based on the normalized effective sample size (ESS) information criterion [1]. The normalized ESS is computed according to the equation:

$$ESS = \frac{1}{N \sum w_i^2}$$

Where N is the number of particles, and w_i is the weight of particle i. The normalized ESS is bounded between 0 and 1. The criterion will be near 1 if all the weights are similar, meaning that the diversity of particles is high and decreasing the number of particles will not affect performance too much. Contrastingly, the criterion will be near 0 if there is a few particles dominating the weights, thus indicating the diversity of particles is low and we should increase the number of particles to improve accuracy.

Implementation-wise, we determined that an ESS threshold of 0.5 was appropriate to dynamically adjust the particle count. For each time-step the ESS is below the threshold, the number of particles is increased by 1%. If the ESS is above the threshold, the number of particles is decreased by 1%. The maximum and minimum particle count was capped at 2000 and 100 respectively. In practice, the particle count stably decreases when the robot is localizing itself, and stables to around 300 when there is a few clusters left. Once there is only 1 cluster left, the particle count reduces to around 100.

4.0.1 Multinomial Implementation

The multinomial resampler is the simpler resampling implementation of the two. Essentially, N particles is chosen through random sampling with probabilities proportional to the weights of each particle. The number of particles N is static for the vanilla implementation and dynamically changes for the ESS implementation.

4.0.2 Low-Variance Implementation

The low-variance resampler aims to reduce sampling error during the resampling phase. The pseudo-code for its implementation can be found in Chapter 4.3 of [2] and is shown

in Figure 3. Essentially, a random number r is chosen near the very beginning of the array. We then take strides of size M^{-1} across the array and include the particle at that array location to our resampled set. A graphical representation is shown in Figure 4. As with the multinomial implementation, the number of particles N is static for the vanilla implementation and dynamically changes for the ESS implementation. The final implementation and the videos generated uses the low-variance resampler as experimentally it is better at converging to the correct robot location.

Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):
- $\bar{\mathcal{X}}_t = \emptyset$
- $r = \text{rand}(0; M^{-1})$
- $c = w_t^{[1]}$
- $i = 1$
- *for* $m = 1$ *to* M *do*
  - $U = r + (m-1) \cdot M^{-1}$
  - *while* $U > c$
    - $i = i + 1$
    - $c = c + w_t^{[i]}$
  - *endwhile*
  - *add* $x_t^{[i]}$ *to* $\bar{\mathcal{X}}_t$
- *endfor*
- *return* $\bar{\mathcal{X}}_t$

Figure 3: Low-Variance Resampling

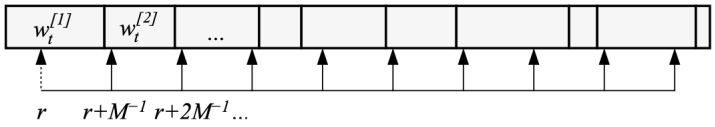


Figure 4: Graphical Representation of Low-Variance Resampling

5 Parameter Tuning

The following section discusses the chosen tuned values for each step of the particle filter algorithm.

5.1 Initialization

Final tuned values:

- **Number of particles**: 1500

A large number was chosen so as to adequately cover the free space on the map. We found that the robot was not able to robustly localize itself if the particle count was below 500 as there is quite a high chance that no particles are initialized close to its ground truth position. Implementation-wise, any value above 1000 should suffice as the ESS resampling procedure we implemented will dynamically adjust the number of particles. We ended up choosing 1500 to increase the robustness of establishing initial localization.

5.2 Motion Model

Final tuned values:

- α_1: 0.0005
- α_2: 0.0005
- α_3: 0.005
- α_4: 0.005

The intuition of the α values are as follows:

- α_1 is the effect of rotation on rotational noise
- α_2 is the effect of translation on rotational noise
- α_3 is the effect of translation on translational noise
- α_4 is the effect of rotation on translational noise

The initial values given were $\alpha_i = 0.01$. However, we found that the robot was moving quite slowly as thus the noise effects were not that high due to the robot's movement. This is our rationale to reduce the parameters. Additionally, we reasoned that heuristically, the effects on rotational variance should be a order of magnitude smaller than translational variance due to the fact that the translational motion in Wean Hall could be over an approximately 60×20 meter-squared area, whereas the translational motion is bounded from 0 to 2π. Therefore, the translational variance should be smaller simply due to its limited range of possible values. We tried different orders of magnitudes for the parameters and settled on our final tuned values. Higher orders of magnitude causes the robot's motion model to become too inundated with noise, causing poor localization. Lower orders of magnitude causes the robot to sometimes become stuck as no new close points are explored due to putting too much trust on the odometry measurements.

5.3 Sensor Model

Final tuned values:

- z_{hit}: 5
- z_{short}: 0.2
- z_{max}: 1
- z_{rand}: 250
- σ_{hit}: 50
- λ_{short}: 0.1
- **Max Range**: 1000
- **Min Probability**: 0.35
- **Subsampling**: 5

The sensor model took the longest to tune due to the number of parameters. We kept the original max range and min probability parameters as it did not affect performance very much. Looking into the `wean.dat` occupancy grid and `robotdata.log`, the reason for this is because in most scenarios, the data-points never reach these boundaries. The occupancy grid is quite robust and in most cases detects obstacles with an occupancy value above 0.7. Similarly, the robot almost always registers the majority of its sensor readings at values below 1000. This is because the robot is navigating in relatively narrow hallways and almost never has a direct line of sight above 10 meters (unless it is looking directly down a hallway).

We also left the σ_{hit} and λ_{short} parameters unchanged as we did not have access to the physical sensor to determine its noise (to find σ_{hit}) nor the environment in which the sensor readings were conducted (to find λ_{short}). The original intention was to tune these two values last once the other parameters are finalized. However, we ended up finding that the two initial values worked reasonably well during the tuning process. As such, we have decided to leave them unchanged. The subsampling parameter is set at 5 as a compromise between computational efficiency and accuracy. Setting 5 degree intervals did not end up losing too much accuracy since the robot is operating in a hallway and not much Cartesian distance is traveled for each degree.

The z_i values determine the weighting of the four probability functions discussed in the sensor model section (albeit the max function is actually a generalized function). There were a lot of random noise from the environment, resulting in us increasing the z_{rand} value from its original 100 to 250. We decided to increase the z_{max} weighting from 0.1 to 1 since we decided on a relatively close max range of 1000 (i.e. 10 meters). This means that whenever the robot's sensor is pointed down the hallway, it would exceed the max range. A higher z_{max} weighting accounts for this. Next, we decided to increase the z_{short} value from 0.1 to 0.2 as it caused the model to converge better. We noticed that there appears to be a pair of legs near the robot in the `robotmovie1.gif` file. Therefore, increasing z_{short} accounts for this sensor obstruction from the legs. Finally, we increased z_{hit} from 1 to 5 as our ray-casting and LIDAR visualization function showed that the sensor measurements are fairly accurate. However, it is not exceedingly accurate, so we decided to increase the weighting to not exceed an order of magnitude.

5.4 Dynamic Resampling

Final tuned values:

- **Min Particles**: 100
- **Max Particles**: 2000
- **ESS Threshold**: 0.5
- **Multiplier**: 0.01

The min particle and max particle sizes were set at 100 and 2000 respectively. This is a trade-off between efficiency and accuracy. The particles should not decrease too much to impact the accuracy of the localization. On the other hand, it should not explode in number and cause the computation to be exceedingly slow. The ESS threshold is set at 0.5 as we found it was appropriate to dynamically adjust the particle count. Finally, the multiplier is set at 0.01 so as to not drastically change the number of particles over a single time-step.

6 Performance

The resources section contains the URLs for the YouTube videos of the robot localizing on `robotdata1.log` and `robotdata3.log`. The parameters were tuned using `robotdata1.log` but the convergence performance remains relatively well when applied to other data logs. In instances where the localization fails, we found that this is mostly due to particles randomly not initializing near its ground truth pose. This is because the particle not only needs to be near its ground truth location in the Cartesian plane, it also needs to be roughly oriented in the same direction for the sensor model to work. However, localization failures only accounted for less than 20% of the trails that we ran.

To increase the speed performance of our algorithm, we implemented several methods to decrease the computation required. The first is to only initialize the particles in the unoccupied areas of the map. This decreases the number of particles required to uniformly cover the whole map to act as our prior belief. In the sensor model step, we increased the sub-sampling angle of the laser scan to 5 degrees. This is done to decrease the number of ray-tracing computations down from 180 to 36. Performance is relatively unaffected due to the close proximity nature of the Wean Hall environment. Finally, for the resampling step, the number of particles is dynamically adjusted using the ESS information criterion. This has the effect of decreasing the number of particles as the robot becomes increasingly localized, thus increasing computation speed as the time-steps progress.

During some runs of the particle filter model, we noticed that sometimes the weights become very close to zero prior to normalization. This resulted in weird behaviors where the weights would normalize erroneously. To increase the numerical stability performance of our implementation, we switched to using log-likelihoods and log-probabilities in our sensor model step. This made our weights more stable and the weird behaviors went away as the weights are no longer diminishing towards zero.

In general, the algorithm along with the tuned parameters are quite robust and repeatable. With few exceptions, the algorithm always converges to a single blob after around 150 time-steps and almost always converges to the ground truth location near the large room to the right of the long hallway. The algorithm is robust and repeatable on the different data logs given and convergences reasonably well for all of them.

7 Future Work

While the particle filter generally converges to the correct location—allowing the robot to navigate the map without colliding with obstacles—it sometimes converges to an incorrect location. This issue could likely be addressed by further tuning the sensor parameters or by solving the kidnapped robot problem. Additionally, parallelizing the motion and sensor models may speed up the simulation and enhance the overall efficiency of the particle filter. Another potential improvement is to integrate simultaneous localization and mapping (SLAM) techniques, which would enable the robot to localize itself and map its surrounding environment concurrently.

8 Resources

The YouTube video of the robot localizing on `robotdata1.log` is available below. The speed multiplier is 1.52 times.

`https://youtu.be/epW15a-Xa-o`

The YouTube video of the robot localizing on `robotdata3.log` is available below. The speed multiplier is 1.11 times.

`https://youtu.be/kHVk6riAgs8`

The GitHub repository for the code is available here:

`https://github.com/williamfbx/CMU-16833/tree/main/PS1`

9 References

[1] Tim Bock. *What is Effective Sample Size?* Accessed: 2025-02-06. 2024. URL: https://www.displayr.com/what-is-effective-sample-size/.

[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.