16-833: Robot Localization and Mapping, Spring 2021

Homework 1-Robot Localization using Particle Filters

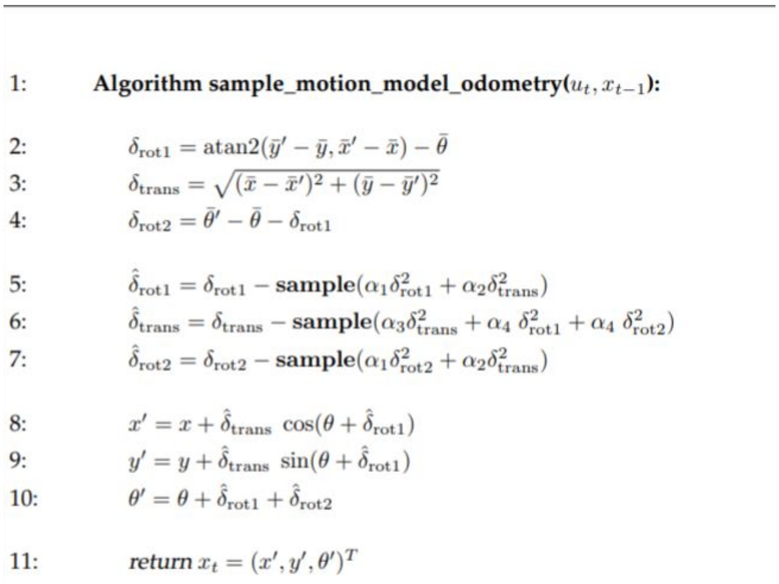
Yichen Xu(yichenxu) Chang Shi(changshi) Aaron Guan(zhongg)

1 Approach and Implementation

For this assignment, we sovled the robot localization problem using the Mote Carlo Localization (MCL) alogrithm or particle filter localization. Given a map, the alogrithm is able to estimate its position and orientation as the robot moves and senses the environment. The alogrithm uses a particle filter to represent the distribution of likely states. Starting with a unifrom distribution of particles over the state space, the alogrithm shifts the particles accordingly as the robot moves, predicting its new state given the movement. At the end of the iteration, the particles are resampled based on how all the sensed data correlate with the predicted state. Lastly, the particles would converge towards the actual position of the robot. The main software framework is done by Python.

1.1 Motion Model

Motion model addresses the state transition probability using mobile robot kinematics and probabilistic techniques. We implemented the odometry motion model following Table 5.6 at Page 136 in [1] to compute posteriors over poses using the odometry measurements.



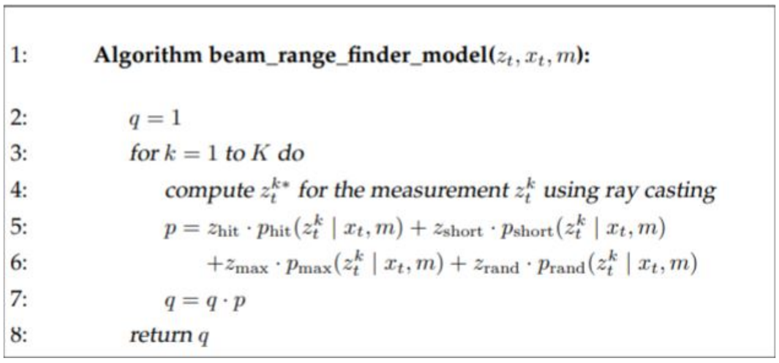
```
1:  Algorithm sample_motion_model_odometry($u_t, x_{t-1}$):

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

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: Algorithm for sampling from state transition probability based on odometry information

1.2 Sensor Model

Sensor models describe the measurement formation process using probabilistic techniques to incorporate the measurement noise. For this assignment, we implemented the range finder sensor model following Table 6.1 at Page 158 in [1] to replicate the physical model of range finders.

This model incorporates four types of measurement errors: small measurement noise, errors due to unexpected objects, errors due to failures to detect objects, and random unexplained noise. The output is a mixture of the four error densities.



```
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
        $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$
            $+ z_{\text{max}} \cdot p_{\text{max}}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$
        $q = q \cdot p$
    return $q$
```

Figure 2: Algorithm for computing the likelihood of a range scan

1.3 Ray Casting

In the sensor model, the importance weight for a particle is computed from the difference between the laser measurement z_t^k at time t and the true range of the object measured by z_t^{k*}. In location-based maps, the range z_t^{k*} can be determined using ray casting. Since ray casting is one of the most computationally expensive operations, we improved the efficiency by pre-computing a look-up table for the raycasts. The process goes as follows: (1) We iterate over all obstacle-free positions in the map (2) We ray cast 360 laser beams from the position, which span 360 degrees. (3) We find the location where the rays hit for the first time an obstacle in the map. (4) The offset from the laser position to the obstacles location is the true laser range measurement.

The Algorithm 1 shows the process of computing the ray cast at the position $[x_{laser}, y_{laser}, \theta_{laser}]$. The variable M, $step$, $maxRange$, and H represent the map, the step move forward along the beam, the maximum laser range measurement, and the occupied probability threshold.

Algorithm 1 Ray-cast Pre-computing

```
Require: $M, x_{laser}, y_{laser}, \theta_{laser}, step, maxRange, H$
$x = x_{laser}$, $y = y_{laser}$
$dx = step * cos(\theta_{laser})$, $dy = step * sin(\theta_{laser})$
while $\|[dx, dy]\|_2 <= maxRange$ do
    $x = x_{laser} + dx$
    $y = y_{laser} + dy$
    if $M[y][x] >= H$ or $M[y][x] == -1$ or $x >= M.shape[1]$ or $y >= M.shape[0]$ then
        break
    end if
    $dx = dx + step * cos(\theta_{laser})$, $dy = dy + step * sin(\theta_{laser})$
end while
raycastTable$[x_{laser}][y_{laser}][\theta_{laser}] = \|[dx, dy]\|_2$
```

After the look-up table is obtained, we can easily get the true range measurement z_t^{k*} at the laser position $[x_{laser}, y_{laser}, \theta_{laser}]$ of a particle. Since the laser is approximately 25 cm offset forward from the true center of the robot, we need to calculate the laser origin by applying a translation transform on the particles. Therefore, the laser pose is calculated by $x_{laser} = x + offset * cos(\theta)$,

$y_{laser} = y + offset * sin(\theta)$, $\theta_{laser} = \theta + i * \pi/180 - \pi/2$ when querying the raycasts loop-up table, where x, y, and θ are sampled particle pose, and i represent the i-th beam starting from right to left. Figure 3 shows an example of the ray cast beam of one particle.

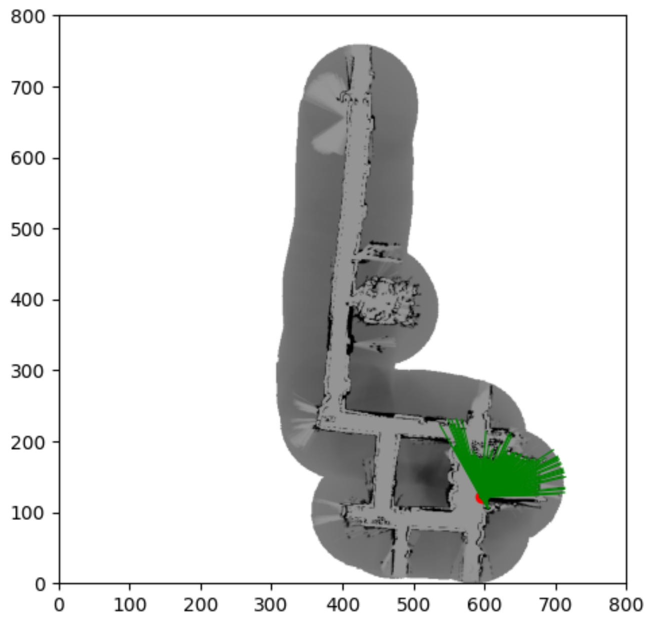


Figure 3: Example of the ray cast "true"" laser beam of one particle

1.4 Resampling

We implemented the low variance resampling following Table 4.4 at Page 110 in [1] to resample our particles. It uses a single random number to sample from the particle set with associated weights, yet the probability of a particle to be resampled is still proportional to its weight. This helps us to reduce the variance of the particles during the resampling.

2 Results

We used the logs "robotdata1.log" and "robotdata5.log" to verify our implementation and the demonstration video indicates our particle filter can converge and generate correct localization results for both logs.

We used the same set of parameters for both data logs. 3000 particles are sampled in the obstacle-free area. The motion models parameters $\alpha_1 = 0.00001$, $\alpha_2 = 0.00001$, $\alpha_3 = 0.0001$ works well for both logs. The parameters for sensor models are shown in Table 1. The maximum expected value for a laser measurement was set as 8193 cm and the minimum occupied probability is 0.35.

Parameter	z_hit	z_short	z_max	z_rand	sigma_hit	lambda_short	subsampling
Value	1	0.12	0.05	800	100	0.1	2

Table 1: Sensor model parameters

As shown in Figure 4, the 3000 particles are initialized as uniform distribution over all obstacle-free area in the map with occupied probability 0. Our particle filter then converges to the correct global position within around few iterations. The total time for the particle filter to process all the data lines are 2351s and 2636s for “robotdata1.log”” and “robotdata5.log””, respectively.

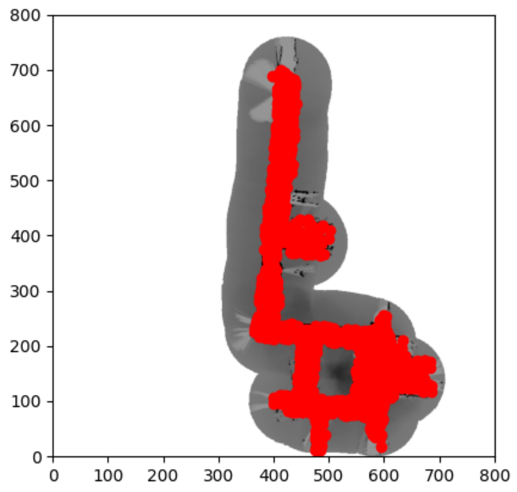


Figure 4: Map with initialization of 3000 particles

Although the 3000 particles are still insufficient to generate robust localization results given that the state space is quite large, our parameters can allow the particle filter to converge to the correct location and generate the correct trajectory for most of the time in an efficient way. We avoided looping and applied vectorization in sensor model to compute the importance weight for the particle, which significantly speed up our algorithm. Additionally, thanks to our implementation of the ray cast look-up table pre-computation, the speed of our algorithm is really fast and it uses about 1.7s to process each laser measurement data for 3000 particles, and about 2s for 5000 particles. The Figure 5 shows the approximate final robot’s position for the “robotdata1.log”” and “robotdata5.log””, which indicates that our particle filters can converge to the correct location. In the video, it also demonstrates the correct global trajectory, which is consistent with the local trajectory generated from the odometry.

The video link: https://www.youtube.com/watch?v=ccdhu0KU7sE. The video consists of localization results from both log1 and log5.

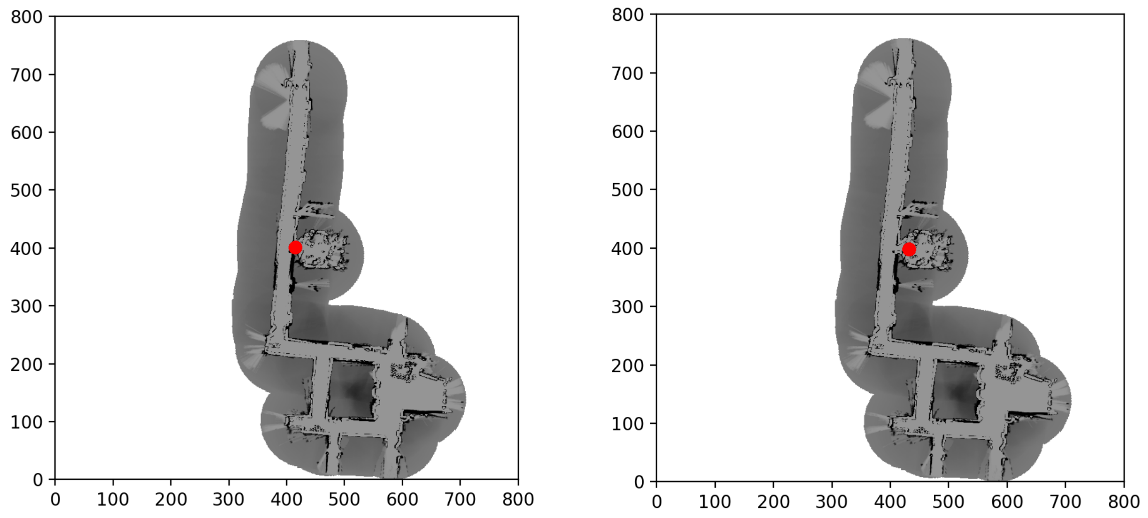


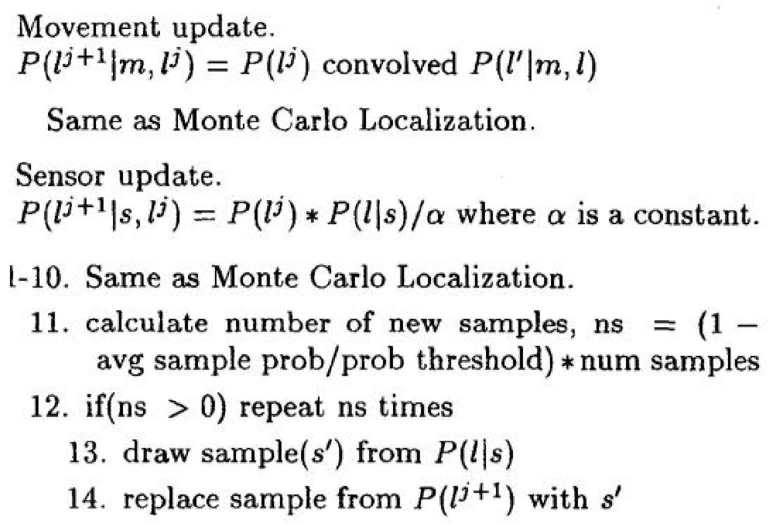
Figure 5: Plots of the robot’s final location (left) For ‘robotdata1.log’ and (right) For ‘robotdata5.log’

3 Future Work

To further improve the robustness of our particle filter, we may better tune the parameters such as decreasing the angle resolution of the sensor model. Also comparing with a fixed number of particles, a better solution would be to implement a particle filter with adaptive number of particles which can wisely add more particles when even the probabilities of the most confident particle is low, and reduce number of particles when all particles have similar poses without adding new information.

4 Extra credit questions

1. Kidnapped robot problem
   One potential solution is to add an extension to the existing MCL called sensor resetting localization[2], or SRL. SRL adds a new step to the sensor update phase of the algorithm. If the probability of the locale designated by the samples we have is low given the sensor readings, we replace some samples with samples drawn from the probability density given by the sensors. The logic behind this step is that the average probability of a locale sample is approximately proportional to the probability that the locale sample set covers the actual location of the robot.



```
Movement update.
$P(l^{j+1}|m, l^j) = P(l^j)$ convolved $P(l'|m, l)$
  Same as Monte Carlo Localization.

Sensor update.
$P(l^{j+1}|s, l^j) = P(l^j) * P(l|s)/\alpha$ where $\alpha$ is a constant.

1-10. Same as Monte Carlo Localization.
  11. calculate number of new samples, ns = (1 − avg sample prob/prob threshold) * num samples
  12. if(ns > 0) repeat ns times
    13. draw sample($s'$) from $P(l|s)$
    14. replace sample from $P(l^{j+1})$ with $s'$
```

Figure 6: Summary of sensor resetting localization

   Thus, for the kidnapped robot problem, SRL is able to to localize the robot by replacing some samples of the estimated locations with samples drawn directly from the probability distribution based on the sensor readings. By slowly replacing these samples, the estimated probability distribution slowly shifts to the correct robot location

2. Adaptive number of particles
   Using a fix sample size for particle filter is inefficient, wasting computational resources. Adaptive (or KLD-samplig) Monte Carlo localization approach, that adapting the number of samples over time, can be used to address this issue. The idea behind AMCL or KLD-sampling is to bound the error introduced by the sample-based belief representation of the particle filter. On the high level, at each iteration, the algorithm generates samples until their number is large enough to guarantee that the K-L distance between the maximum

likelihood estimate and the underlying posterior does not exceed a pre-specified bound. Thus, this model picks a small sample size if the density is focused on small subspace of the state place, and picks a large sample size if they have to cover a bigger part of the state space. The K-L distance is a measurement of how one probability distribution is different from a second.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* MIT press, 2005.

[2] S. Lenser and M. Veloso, "Sensor resetting localization for poorly modelled mobile robots," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 1225–1232 vol.2, 2000.