REPORT

16833 - HW1: Particle Filter Localization

Jae seok Oh (jaeseoko) , Ivan Wong (ivanw)

I. APPROACH

A. Initialization

Generate a set number of particles in the free space within the occupancy grid. (i.e. particles are placed only on the coordinates where occupancy probability is zero.)

B. Motion Model

Update each particle state whenever odometry data is received. Tunable parameters are used to induce noise into the motions.

C. Sensor Model

Update weights of the particles whenever laser scan data is received. Ray casting is performed for each particle to get true laser scans. Then, compare the true laser scans with incoming laser scan data from the actual robot position and calculate the weights (plausibility of the particles) and update the weights accordingly.

D. Resampling

Resample the particles after updating the weights of all particles based on the new weights. Particles with higher importance weights will be more likely to be resampled more in the process.

II. IMPLEMENTATION

A. Initialization

"Free-space Initialization"
Weights of all particles are set to 1 initially. Each state of the particles (x,y,angle) is sampled from uniformly random distribution.
X State Init = [ ]
Y State Init = [ ]
X - range from (3000,7000)
Y - range from (0,7500)
Angle - range from $(-\pi, +\pi)$
While: number of particles sampled < total number of particles
    Try: sample (x,y,angle) while State Init size < # particles
        If (x,y,angle) in free space, append to State Init.

Returns the initial state belief (x, y, θ) to be used in the motion model.

B. Motion Model

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

Reference: Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. MIT press, 2005 [Chapter 5]

Odometry motion model used from above reference.

Noise parameters ($\alpha 1$, $\alpha 2$. $\alpha 3$, $\alpha 4$) are tuned from multiple testing.
Parameters used in the algorithm:

$\alpha 1$ =0.0005

$\alpha 2$ =0.0005

$\alpha 3$ = 0.001

$\alpha 4$ = 0.001

Tuning process:

We first tried to see the motion of particles based only on motion model update from odometry data without sensor model. We found out having lower variance, or depending more on the odometry data, represented better trajectories. After implementing the sensor model, we occasionally observed divergence of the particles after converging. In particular, we observed that the particles occasionally deviate radially. We deduced that the motion model update in rotation of the poses needed less noise, and finally ended up reducing $\alpha 1$ and $\alpha 2$, which contributes to the particle pose(θ).

Sample motions are implemented by sampling from normal random distribution in lines (5,6,7) in the above algorithm with zero mean and standard deviations, respectively:

$(\alpha 1\delta_{rot1}^2 + \alpha 2\delta_{trans}^2)$

$(\alpha 3\delta_{trans}^2 + \alpha 4\delta_{rot1}^2 + \alpha 4\delta_{rot2}^2)$

$(\alpha 1\delta_{rot2}^2 + \alpha 2\delta_{trans}^2)$

Similar to the referenced algorithm, our motion model returns an array with dimension $(\#\ of\ particles\ x\ 3)$ as our state belief to be fed into our sensor model.

C. Sensor Model

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: $p = z_{\text{hit}} \cdot p_{\text{hit}}(z_t^k \mid x_t, m) + z_{\text{short}} \cdot p_{\text{short}}(z_t^k \mid x_t, m)$

6: $+ z_{\max} \cdot p_{\max}(z_t^k \mid x_t, m) + z_{\text{rand}} \cdot p_{\text{rand}}(z_t^k \mid x_t, m)$

7: $q = q \cdot p$

8: *return* q

Reference: Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. MIT press, 2005 [Chapter 6.3]

Beam Range Finder Model used in Sensor model from above reference.

Let:

K - Number of laser beams being tested. This is determined by how many evenly split angle increments we are testing.

Z_t^k - Incoming Laser scan data from actual robot position.

Z_t^{k*} - Actual laser scan for each particle using “ray casting”.

For all lasers k in K, perform “Ray Casting”:

Z_t^{k*} = [ ]

For each angle i, try:

Propagate forward by set step size.

Check occupancy grid for current coordinate (x,y).

If (occupancy probability at (x,y) > threshold):

Append current laser range as cumulative step to Z_t^{k*}

Else if (cumulative step >= laser max range):

Append current laser range as laser max range to Z_t^{k*}

Return Z_t^{k*}

Then use Z_t^{k*} and Z_t^k in line 5,6 in the referenced algorithm with the probability definitions listed below:

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

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

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

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

Reference: Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. MIT press, 2005 [Chapter 6]

Following probabilities calculated from above reference.

$P_{hit}(z_t^k \mid x_t, m)$ - Probability of hitting an obstacle.
$P_{short}(z_t^k \mid x_t, m)$ - Probability of hitting an unexpected object.
$P_{max}(z_t^k \mid x_t, m)$ - Probability of reaching laser max range.
$P_{rand}(z_t^k \mid x_t, m)$ - Probability of random measurements

The calculated probabilities are put into a dot product with its priors as indicated in line 5,6, and a new weight is determined through conditional independence assumption by multiplying the individual laser weights. The likelihood of laser range from a particle is returned, and the process is repeated for all particles until a new weight distribution across all particles is formed. The new weight distribution, along with the new state belief, are sent to be used as data to resample the particles.

D. Resampling

1: **Algorithm Low_variance_sampler($\mathcal{X}_t, \mathcal{W}_t$):**
2: $\bar{\mathcal{X}}_t = \emptyset$
3: $r = \text{rand}(0; M^{-1})$
4: $c = w_t^{[1]}$
5: $i = 1$
6: *for* $m = 1$ *to* M *do*
7: $U = r + (m-1) \cdot M^{-1}$
8: *while* $U > c$
9: $i = i + 1$
10: $c = c + w_t^{[i]}$
11: *endwhile*
12: *add* $x_t^{[i]}$ *to* $\bar{\mathcal{X}}_t$
13: *endfor*
14: *return* $\bar{\mathcal{X}}_t$

Reference: Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. MIT press, 2005 [Chapter 4.3]

Low variance sampler used for resampling from above reference.

Normalized weights are used in line 4 for $c = w_t^{[i]}$
Particles with higher importance weights will be more sampled from the while loop in lines 8 - 11 in the referenced algorithm above.

The sampler returns a new array of states with their respective weights. This new array is then referenced back as the prior states back in the motion model and the Monte Carlo localization algorithm is performed recursively until the end of the robot data log is reached.

E. Parameter Tuning

Set parameter:

Minimum probability (threshold) = 0.35

Tunable parameters:

Z_{hit} = 150 , Z_{short} = 17.5, Z_{max} = 15, Z_{rand} = 100
η = 1 , σ_{hit} = 100, λ_{short} = 5

Tuning process:

Tunable parameters were tuned based on multiple trial runs on small samples of particles.

Main intuition on setting the parameters came from looking at the map and possible robot positions within the map. We have tuned $Z_{hit}, Z_{short}, Z_{max}$ as the likelihoods of the following:

Z_{hit} - Regular laser measurement with noise, this is the most probable scenario out of all the other cases, thus it warrants the largest weight.
Z_{short} - The occasion that objects that were introduced to the map after the map has been recorded. We assumed that only a small portion of the map is affected by this change in occupancy, thus the weight is small.
Z_{max} - The occasion for hitting max range happens only when the robot is at either end of the hallway, which warrants the smallest weight since there are only two locations in the entire map for this to be true.

η - Set to 1 per advice from instructors.
Z_{rand}, $\sigma_{hit}, \lambda_{short}$ - We tuned multiple times through trial and error based on multiple test runs on a low number of particles and ray casts.

Tunable ray parameters:

- Laser beams subsampled - 30 beams (every 6 degrees)
- Max range iterated - 8183 [cm]
- Laser step size - 20 [cm] (2 occupancy grids)

Number of particles: 500

III. PERFORMANCE

The algorithm used is a standard method and non-vectorized in python, so the computation takes quite a while to complete. Using robotdata 1 log, it takes on average 40 minutes to reach the end of time step, and in robotdata 2 log, it takes a little over 3 hours. On average, the particles converge around or after halfway through the time steps (log file). On occasions, the particles diverge even after converging and converge back again. The accuracy in converging into consistent spots seems to depend on the initialization of the particles and number of particles, which we could have increased if we had more efficient algorithms.

IV. ROBUSTNESS

The parameters we used for our motion and sensor model have seen convergence in both of the test cases in robotdata 1 log and robotdata 2 log. Our

parameters seem to be more suitable for robotdata 2 log as the particles can be seen to converge very quickly in the first quarter of the log, contrasted by robotdata 1 log, in which it will take at least half of the log before converging into a cluster of particles.

V. REPEATABILITY

We aimed to conduct roughly about 20000 steps for robotdata 1 log and robotdata 2, which translated to 10 runs from using the first log, and 5 runs from using the second log. Over the past week, we were able to conclude that our parameters worked 7 out of 10 runs using the first log, and 4 out of 5 runs using the second log. We deduced that the repeatability depends on how the particles are first initialized.

VI. RESULTS

A. Link to Videos
   1. From robotdata1.log: https://youtu.be/XbNUJAORO4w
   2. From robotdata2.log: https://youtu.be/P-boSyNz5PQ

B. Future Work/ Improvement Ideas
   1. More efficient algorithms couldn't be implemented due to time constraints. In the future, the ray casting speed could be improved by vectorizing and eliminating 'double for loop', or precomputing the look up tables for raycasting prior to computing laser scans Z_t^{k*}. Additional elimination of loops in the main loop could be considered to avoid slow speed in python.
   2. Alternative to vectorizing, we could try implementing the whole algorithm in C++ to expedite the computing speed and use more number of particles to improve accuracy and robustness.
   3. Our current algorithm was limited mainly by the lack of computational efficiency, which prevented us from using more particles. With the above improvements, we could use more particles to have better accuracy, robustness and repeatability.