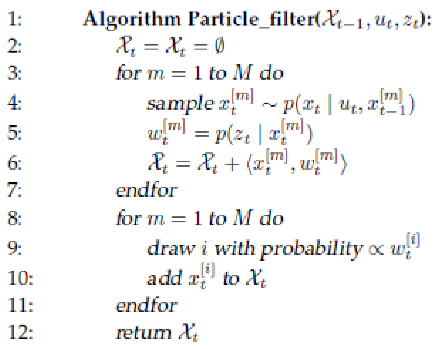
PARTICLE FILTER REPORT

Hanjiang Hu
AndrewID: hanjianh
Homepage: `https://hanjianghu.net/`
Code: `https://github.com/HanjiangHu/particle-filter-localization`

10/02/2022

1 Particle Filtering Approach

As a non-parametric localization method based on Bayesian filtering, particle filtering (PF) method uses particles to represent the belief of the state, i.e. the localization (position and orientation in 2D) of the robot. As the following figure [2] shows, the PF algorithm takes the last states of particles $\mathscr{X}_{t-1}$, current observation z_t and current motion from odometry u_t and return the current updated states $\mathscr{X}_t$ of particles. There are three



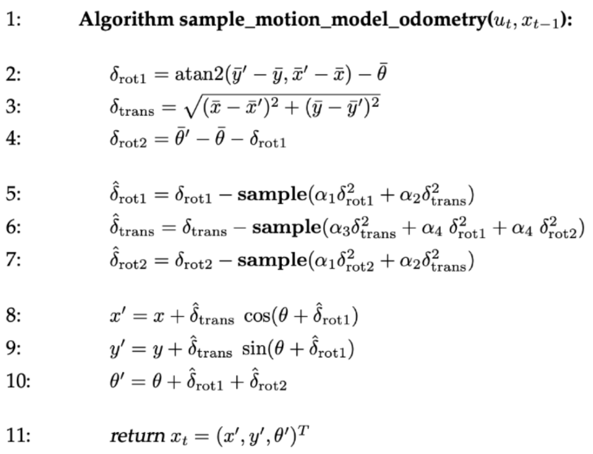
```
1:  Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
2:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3:      for $m = 1$ to $M$ do
4:          sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5:          $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6:          $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:      endfor
8:      for $m = 1$ to $M$ do
9:          draw $i$ with probability $\propto w_t^{[i]}$
10:         add $x_t^{[i]}$ to $\mathcal{X}_t$
11:     endfor
12:     return $\mathcal{X}_t$
```

Figure 1: The particle filter algorithm overview

important steps in the algorithm, motion model on Line 4, sensor model on Line 5, and resampling on Line 9 in Figure 1, which will be introduced below.

1.1 Motion Model

The motion model takes the odometry measurement u_t and the last state of each particle $\mathscr{X}_{t-1}$ as input, and outputs the predicted state $\hat{\mathscr{X}}_t$ as a distribution. Specifically, the motion model in the 2D localization problem is shown below [2]. It can be seen that the new state of postion x, y and orientation θ comes with noises of



```
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 2: Motion model overview

range translation δ_{trans}, position rotation δ_{rot1} and orientation rotation δ_{rot2}. The odometry gives the mean values of these three random variables, which are calculated on Line 2-4 in Figure 2, while the real motion sampling follows the normal distribution with variance shown on Line 5- 7 in Figure 2. The variances serve as hyperparameters and will be introduced in 2.1.

1.2 Sensor Model

The sensor model predicts the weight of each particles w_t based on the conditional probability or likelihood of the current observation u_t given the predicted state $\hat{\mathscr{X}}_t$ from motion model. As shown in the following figure [2], we adopt ray tracing algorithm as the beam range finder. For each beam z_t^k, the range distance follows the mixed distribution of Gaussian distribution of hitting the obstacles in the map p_{hit}, exponential distribution of decaying p_{short}, point-mass distribution at the maximal limit of the range p_{max} and the uniform random noise applied in the whole range p_{rand}. The likelihood of all beams for each particle is calculated as the weighted multiplication of all four probabilistic density functions. The weight of four algorithms are introduced in 2.2.

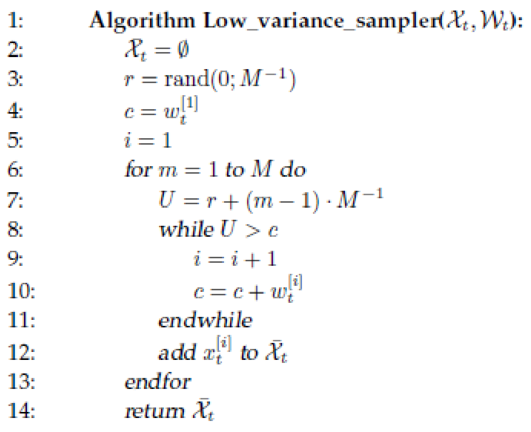
```
Algorithm beam_range_finder_model($z_t, x_t, m$):

    $q = 1$
    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 3: Sensor model overview

1.3 Resampling Model

The resampling step gives the updated particle states $\mathscr{X}_t$ with the posterior probability weighted by w_t. To avoid the high variance by resampling all the particles where some particles with low weights are oversampled due to randomness, we adopt low variance resampling as shown in the following figure [2]. The low variance sampling uses constant number r in the first place and increase it uniformly until it matches the summation of weights by traversing all the particles. In this case, the chance that low-weighted particles are sampled is much lower.



```
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 4: Resampling algorithm overview

2 Particle Filtering Implementation

In this section, all the details to implement the PF algorithms are shown. To speed up the implementation of robot localization problem, the code is vectorized in NumPy. Code is available on `https://github.com/HanjiangHu/particle-filter-localization`.

2.1 Motion Model

As shown in 1.1, the Gaussian variance of δ_{trans} depends on all $\delta_{trans}, \delta_{rot1}, \delta_{rot2}$ by α_3, α_4 on translation and rotation respectively. The variance of rotation δ_{rot1} depends on $\delta_{rot1}, \delta_{trans}$ by α_1, α_2. The variance of rotation δ_{rot2} depends on $\delta_{rot2}, \delta_{trans}$ by α_1, α_2. Since the fluctuation of rotation influence translation more seriously compared to the influence of translation on rotation, α_3, α_4 are larger than α_1, α_2. Followed by this fine-tuning idea, we have $\alpha_1 = \alpha_2 = 0.0005$ and $\alpha_3 = \alpha_4 = 0.001$.

2.2 Sensor Model

As shown in 1.2, we tune the parameters of weights of four mixed probabilistic density functions. For the Gaussian distribution by hitting the map, it happens very commonly as the surrounding of the robot is within its range, e.g. the corridor. And the uniform noise is also very common due to the movement or reflection of 2D beams. So their weights are higher than exponential decay and maximal cutting. As a result, the weight of four distribution is $z_{hit} = 300, z_{rand} = 200, z_{max} = 30, z_{short} = 355$. For the hitting distribution, the variance is 1cm so $\sigma_{hit} = 100$ and for decay distribution, $\lambda_{short} = 15000$. The minimal probability of obstacle in the map is 0.35. The resolution is 10 and the maximum range of laser is 8183.

2.3 Efficient Initialization

To make the algorithm more efficient, we adopt the particle initialization to the free space of the map. we first sample the coordinates in the map uniformly and then choose the particles where the pixel of map is not occupied. It is not verterized but it does not affect the speed too much because it runs just in the beginning.

3 Results and Analysis

3.1 Performance Overview

The performance is very good in terms of several provided log files. The videos of log file 1 and log file 3 can be found on `https://youtu.be/h8rn5NSPXpk`. Usually, there are over 2000 frames, and the wrong particles decay very quickly and it would be 10% left after 400 frames due to the proper hyperparameters and vectorized matrix calculation in NumPy. The initial and final results for log file 1, 2, 3, 4, 5 are shown below. The number of initial particles are 500 in the free sapce.

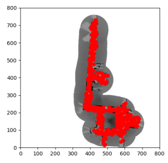


Figure 5: Robotlog 1 initial 500 particles

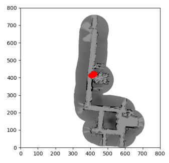


Figure 6: Robotlog 1 final particles

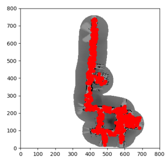


Figure 7: Robotlog 2 initial 500 particles

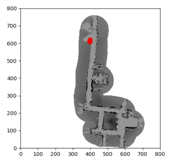


Figure 8: Robotlog 2 final particles

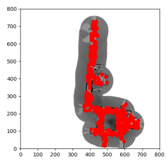


Figure 9: Robotlog 3 initial 500 particles

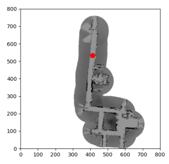


Figure 10: Robotlog 3 final particles

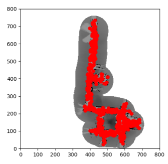


Figure 11: Robotlog 4 initial 500 particles

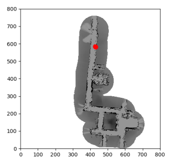


Figure 12: Robotlog 4 final particles

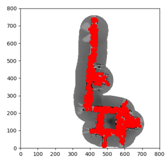


Figure 13: Robotlog 5 initial 500 particles

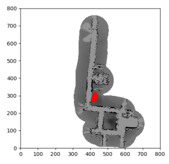


Figure 14: Robotlog 5 final particles

3.2 Different Number of Particles

To further study the influence of different number of particles, we choose robotlog1 with particle number of 10, 20, 50, 100, 200, 500, 1000, 2000, 5000 and implement the same algorithm with same hyper parameters, as shown in the following figures. It can be seen that when the number of particles is too small (n=10), the particle filter algorithm cannot converge correctly due to limited samples. When the number of particles increases, the final particles become correct and steady. However, if the number of particles is too large (n=1000,2000,5000), the PF algorithm cannot converge in the limited frames. There is a trade off between convergence rate and convergence accuracy regarding the number of particles.

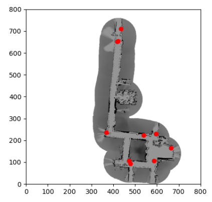


Figure 15: Robotlog 1 initial 10 particles

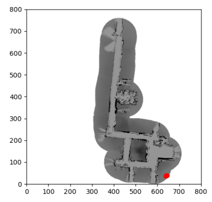


Figure 16: Robotlog 1 final 10 particles

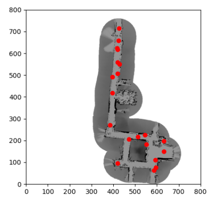


Figure 17: Robotlog 1 initial 20 particles

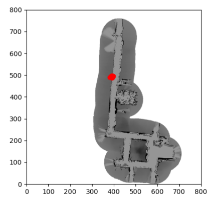


Figure 18: Robotlog 1 final 20 particles

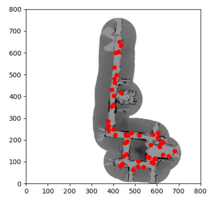


Figure 19: Robotlog 1 initial 50 particles

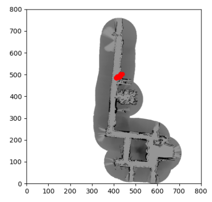


Figure 20: Robotlog 1 final 50 particles

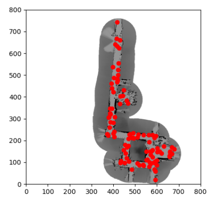


Figure 21: Robotlog 1 initial 100 particles

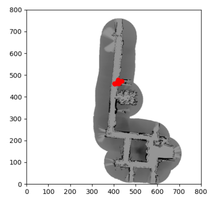


Figure 22: Robotlog 1 final 100 particles

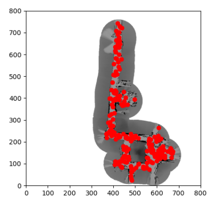


Figure 23: Robotlog 1 initial 200 particles

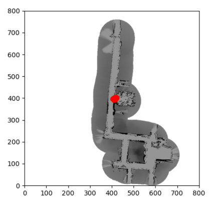


Figure 24: Robotlog 1 final 200 particles

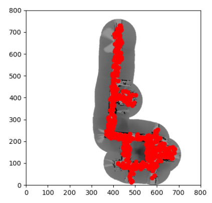


Figure 25: Robotlog 1 initial 500 particles

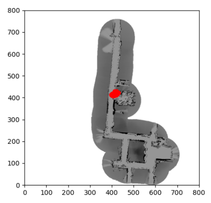


Figure 26: Robotlog 1 final 500 particles

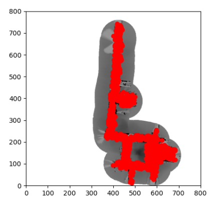


Figure 27: Robotlog 1 initial 1000 particles

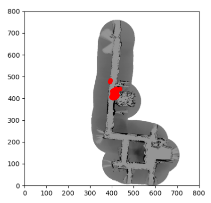


Figure 28: Robotlog 1 final 1000 particles

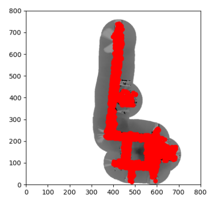


Figure 29: Robotlog 1 initial 2000 particles

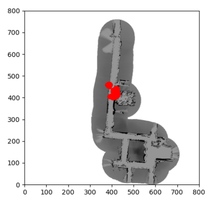


Figure 30: Robotlog 1 final 2000 particles

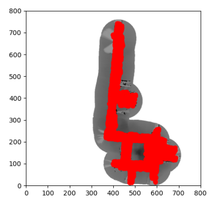


Figure 31: Robotlog 1 initial 5000 particles

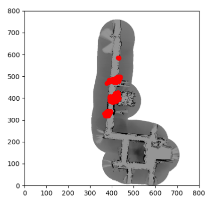


Figure 32: Robotlog 1 final 5000 particles

3.3 Repeatability by Random Initialization

In this section, we study the repeatability of PF through different random initialization. Given fixed number of particles oif 500 for robotlog1, we run PF algorithm for 5 times and the initialization and final results are shown below. It can be seen that although there is some localization difference for the 5 repeated test, the final results are similar in the same area, showing that the implementation of PF algorithm is quite robust.

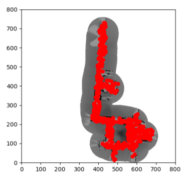


Figure 33: Robotlog 1 initial 500 particles, repeat 1

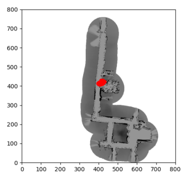


Figure 34: Robotlog 1 final 500 particles, repeat 1

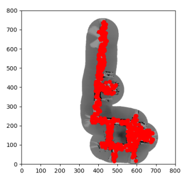


Figure 35: Robotlog 1 initial 500 particles, repeat 2

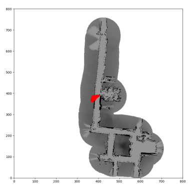


Figure 36: Robotlog 1 final 500 particles, repeat 2

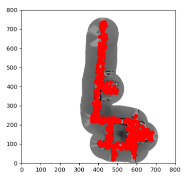


Figure 37: Robotlog 1 initial 500 particles, repeat 3

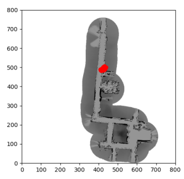


Figure 38: Robotlog 1 final 500 particles, repeat 3

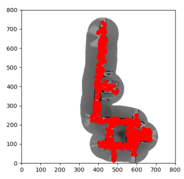


Figure 39: Robotlog 1 initial 500 particles, repeat 4

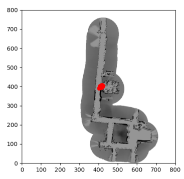


Figure 40: Robotlog 1 final 5000 particles, repeat 4

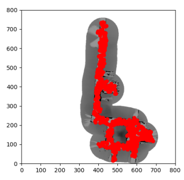


Figure 41: Robotlog 1 initial 500 particles, repeat 5

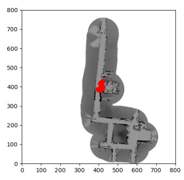


Figure 42: Robotlog 1 final 5000 particles, repeat 5

4 Discussion and Future Work

As a conclusion, the implementation of PF algorithm satisfies the needs of robot localization. Although the code is successfully vectorized and the speed is quite fast, some future work lies in to run it through CUDA acceleration by CuPy or PyTorch. Also it is necessary to implement the adaptive number of particles to save the human effort to tune the number of particles.

4.1 Kidnapped Robot Problem

To mimic the kidnapped robot problem, we chunked parts of logs in robotlog 1. Specifically, we deleted 958-1234 rows of measurement or motion logs and run the PF with 500 particles, and the results are shown below. It can be seen that before kidnapping, the PF works correctly, and when the kidnapping happens, the particles are randomly chosen and become a mess. When all the frames run up, there are many particles left to be filtered due to kidnapping, but the remaining particles are around the correct position, showing that PF algorithm is robust against the kidnapping of robot.

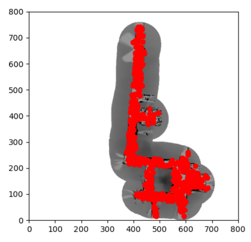


Figure 43: Robotlog 1 initial 500 particles before kidnapping

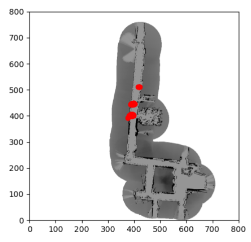


Figure 44: Robotlog 1 957th frame before kidnapping

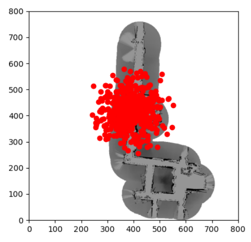


Figure 45: Robotlog 1 958th frame after kidnapping

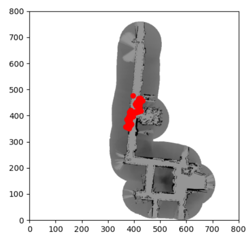
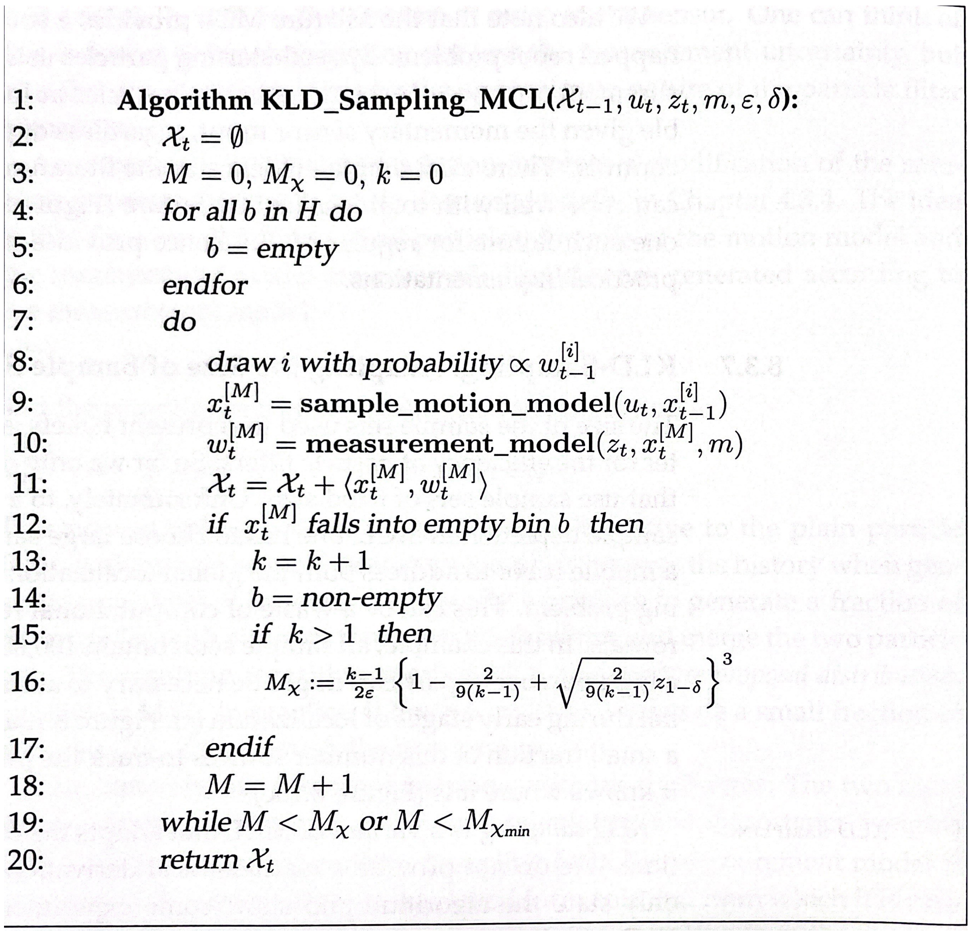


Figure 46: Robotlog 1 final result after kidnapping

4.2 Adaptive number of particles

The main idea to use adaptive number of particles is to make the updated sampled posterior distribution is close to the true posterior through KL-Divergence, called KLD-sampling [1]. The detailed procedure [2] can be seen below. At each iteration of PF, KLD sampling determines the number of samples that the error between the tru posterior and the sample-based approximation is less than ϵ with probability of $1-\delta$.



```
Algorithm KLD_Sampling_MCL($\mathcal{X}_{t-1}, u_t, z_t, m, \varepsilon, \delta$):
    $\mathcal{X}_t = \emptyset$
    $M = 0,\ M_\chi = 0,\ k = 0$
    for all $b$ in $H$ do
        $b$ = empty
    endfor
    do
        draw $i$ with probability $\propto w_{t-1}^{[i]}$
        $x_t^{[M]}$ = sample_motion_model($u_t, x_{t-1}^{[i]}$)
        $w_t^{[M]}$ = measurement_model($z_t, x_t^{[M]}, m$)
        $\mathcal{X}_t = \mathcal{X}_t + \langle x_t^{[M]}, w_t^{[M]} \rangle$
        if $x_t^{[M]}$ falls into empty bin $b$ then
            $k = k + 1$
            $b$ = non-empty
            if $k > 1$ then
                $M_\chi := \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}\, z_{1-\delta}\right\}^3$
        endif
        $M = M + 1$
    while $M < M_\chi$ or $M < M_{\chi_{\min}}$
    return $\mathcal{X}_t$
```

Figure 47: KLD sampling based particle filtering

REFERENCES

References

[1] Dieter Fox. “KLD-sampling: Adaptive particle filters”. In: *Advances in neural information processing systems* 14 (2001).

[2] Sebastian Thrun. “Probabilistic robotics”. In: *Communications of the ACM* 45.3 (2002), pp. 52–57.