HW3 Report: Robot Localization using Particle Filters

Aleksandra Shchetinina
`BS17xxxx`

Anastasia Pichka
`BS17xxxx`

Bekpasha Dursunov
`BS170808`

1 Introduction

2 Motion Model

According to Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005. [Chapter 5] Motion Model that was used is the following. The odometry data are used in this motion model to estimate the nature of the motion depending on the current particle frames. Gaussian noise is included in this model to show that there are inaccuracies in the odometry measurement. With the help of odometry, we can learn about the corresponding achievement from $\hat{x}_{t-1} = (\hat{x}\ \hat{y}\ \hat{\theta})$ to $\hat{x}_t^{'} = (\hat{x}^{'}\ \hat{y}^{'}\ \hat{\theta}^{'})$ Bars means odometry measurements. Relative odometry is extracted by transforming u_t into the following sequence: rotation, transfer and subsequent rotation. The initial rotation is called δ_{rot1}, the transfer δ_{trans}, and the second rotation is called δ_{rot2}. We found them using the following formulas:

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

$$
\begin{aligned}
\hat{\delta}_{\text{rot1}} &= \text{atan2}(y' - y, x' - x) - \theta \\
\hat{\delta}_{\text{trans}} &= \sqrt{(x - x')^2 + (y - y')^2} \\
\hat{\delta}_{\text{rot2}} &= \theta' - \theta - \hat{\delta}_{\text{rot1}}
\end{aligned}
$$

δ_{rot1}, δ_{trans}, δ_{rot2} are sufficient statistics of relative motion encoded by odometry. Independent noise is added to them, as this model of motion suggests. So, there are formulas for them:

$$\hat{\delta}_{rot1} = \delta_{rot1} - \epsilon_{\alpha_1 \delta^2_{rot1} + \alpha_2 \delta^2_{trans}}$$

$$\hat{\delta}_{trans} = \delta_{trans} - \epsilon_{\alpha_3 \delta^2_{trans} + \alpha_4 (\delta^2_{rot1} + \delta^2_{rot2})}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} - \epsilon_{\alpha_1 \delta^2_{rot2} + \alpha_2 \delta^2_{trans}}$$

Here e is independent noise. And then we update parameters in the following way:

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

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

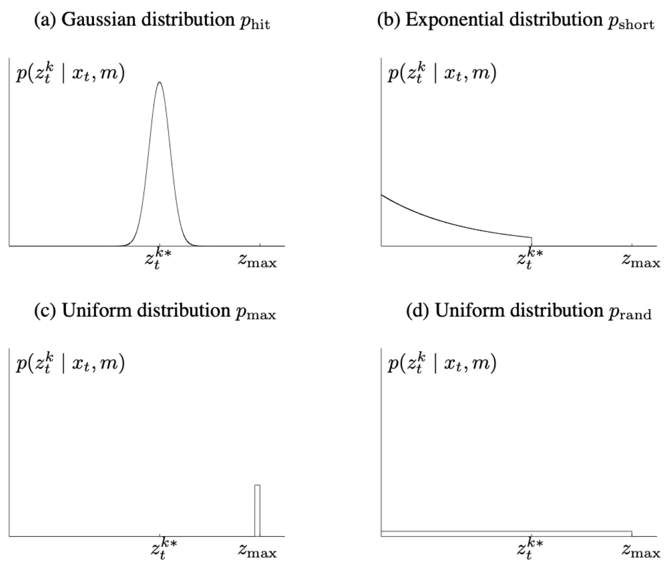
$$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$$

The motion model is adjusted using the following four parameters α_1, α_2, α_3, α_4. The dispersion of the model is determined by them.

3 Sensor Model

The sensor model p$(z_t|x_t, m)$ is needed as part of the correction step for computing importance weights (proportional to observation likelihood) for each particle

The model should incorporate four types of measurement errors, all of which are essential to making this model work:



- ***small measurement noise*** ***(p_{hit})***- arises from the limited resolution of range sensors, atmospheric effect on the measurement signal, and so on - usually modeled by a narrow Gaussian with mean z_t^{kstar} and standard deviation

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

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

- ***errors due to unexpected objects*** ***(p_{short})*** - objects not contained in the map can cause rangefinders to produce surprisingly short ranges - at least when compared to the map, typical example of moving objects are people that share the operational space of the robot - the probability of range measurements in such situations is described by an exponential distribution

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

- ***errors due to failures to detect objects*** ***(**p_{max}**)*** - when obstacles are missed altogether, for example, when laser range finders when sensing black, light-absorbing objects, or when measuring objects in bright light - the sensor returns its maximum allowable value, explicitly model max-range measurements in the measurement model

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

- ***random unexplained noise*** ***(**p_{rand}**)*** - entirely unexplained measurements, sonars often generate phantom readings when they bounce off walls, or when they are subject to cross-talk between different sensors - such measurements will be modeled using a uniform distribution spread over the entire sensor measurement range [0; zmax]

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

Having all distributions, we can build algorithm for beam range finder model

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

It was also important to implement ray casting - a method to detect intersections within an environment using rays which are sent out from a position with a certain angle.

From instructions we know map file format:

- -1 = don't know

- 1 = occupied with probability 1
- 0 = unoccupied with probability 1
- 0.5 = occupied with probability 0.5

For each degree we check, until we reach map borders or find occupied place Now, we can use distance in beam range finder model

4 Resampling

According to the book which was given as reference for the derivation of the particle filter, it shall prove useful to discuss the resampling step in more detail.

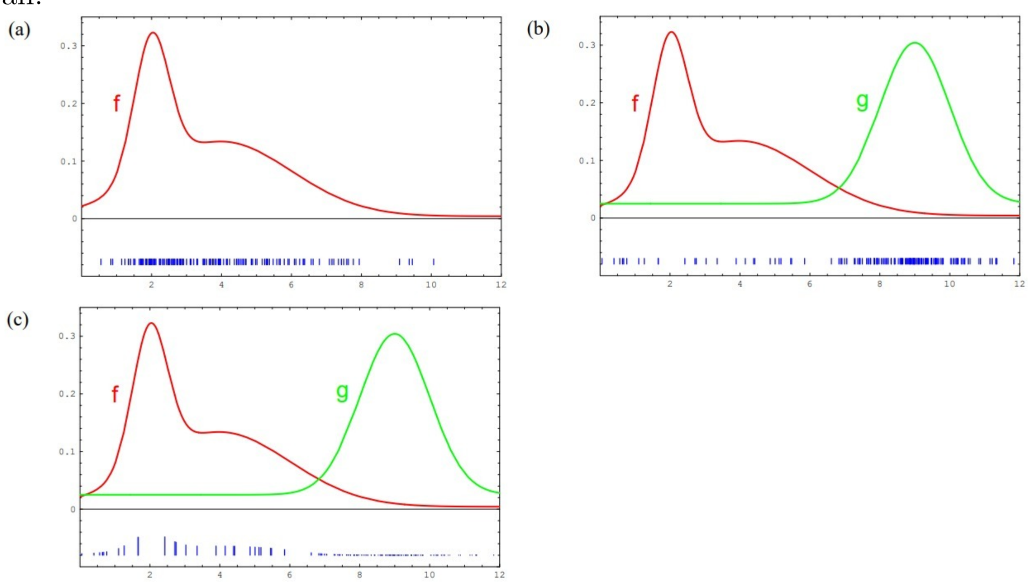


Figure above illustrates the intuition behind the resampling step. Figure (a) shows a density function f of a probability distribution called the target distribution. What we would like to achieve is to obtain a sample from f. However, sampling from f directly may not be possible. Instead, we can generate particles from a related density, labeled g in Figure (b). The distribution that corresponds to the density g is called proposal distribution. The density g must be such that $f(x) > 0$ implies $g(x) > 0$, so that there is a non-zero probability to generate a particle when sampling from g for any state that might be generated by sampling from f. However, the resulting particle set, shown at the bottom of Figure (b), is distributed according to g, not to f. In particular, for any interval A $\leq$ range(X) (or more generally, any Borel set A) the empirical count of particles that fall into A converges to the integral of g under A:

$$\frac{1}{M}\sum_{m=1}^{M} I(x^{[m]} \in A) \quad \longrightarrow \quad \int_A g(x)\, dx$$

To offset this difference between f and g, particles $x^{[m]}$ are weighted by the quotient.

After that for resampling method we used pseudo code that was given in referenced book:

```
Algorithm Low_variance_sampler(X_t, W_t):
    X̄_t = ∅
    r = rand(0; M^-1)
    c = w_t^[1]
    i = 1
    for m = 1 to M do
        u = r + (m - 1) · M^-1
        while u > c
            i = i + 1
            c = c + w_t^[i]
        endwhile
        add x_t^[i] to X̄_t
    endfor
    return X̄_t
```

Low variance resampling for the particle filter. This routine uses a single random number to sample from the particle set X with associated weights W, yet the probability of a particle to be resampled is still proportional to its weight. Furthermore, the sampler is efficient: Sampling M particles requires $O(M)$ time.

The basic idea is that instead of selecting samples independently of each other in the resampling process (as is the case for the basic particle filter in table below), the selection involves a sequential stochastic process.

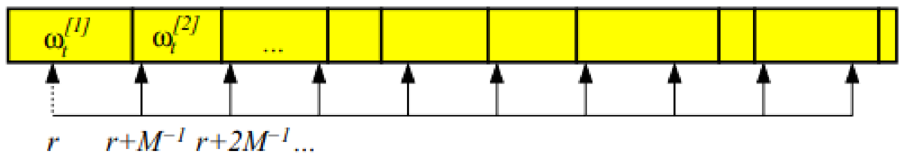


Figure 4.3 Principle of the low variance resampling procedure. We choose a random number r and then select those particles that correspond to $u = r + (m-1) \cdot M^{-1}$ where $m = 1, \ldots, M$.

Instead of choosing M random numbers and selecting those particles that correspond to these random numbers, this algorithm computes a single random number and selects samples according to this number but still with a probability proportional to the sample weight. This is achieved by drawing a random number r in the interval $[0; M^1]$, where M is the number of samples to be drawn at time t. The algorithm in pseudo code above then selects particles by repeatedly adding the fixed amount M^1 to r and by choosing the particle that corresponds to the resulting number. Any number u in $[0; 1]$ points to exactly one particle, namely the particle i for which

$$i = \operatorname*{argmin}_j \sum_{m=1}^{j} w_t^{[m]} \geq u$$

The advantage of the low-variance sampler is threefold. First, it covers the space of samples in a more systematic fashion than the independent random sampler. This should be obvious from the fact that the dependent sampler cycles through all particles systematically, rather than choosing them independently at random. Second, if all the samples have the same importance factors, the resulting sample set $\hat{X}_t$ is equivalent to X_t so that no samples are lost if we resample without having integrated an observation into X_t. Third, the low-variance sampler has a complexity of $O(M)$.

5 Parameter tuning

5.1 Motion Model

The less time between successive measurements, the more similar these different motion patterns are. Thus, if the belief is updated frequently, for example every tenth second for an ordinary room robot, the difference between these movement patterns is not very significant. We selected the parameters empirically. And tried the following: α_1 and α_2 from 0.001 to 0.0001 and α_3, α_4 from 0.1 to 0.01.

5.2 Sensor Model

In textbook, parameter η for Unexpected objects is derived in such a way:

$$\eta = \frac{1}{1 - e^{-\lambda_{\text{short}} z_t^{k*}}}$$

To choose parameters, I had to make a lot of trials, which helped to choose the optimal one Weight for errors choice:

- Correct range with local measurement noise - to make localizing better - need to give high weights - for example, 1000
- Unexpected objects - to make localizing better - need to give less weights - for example, 0.01
- Failures - to make localizing better - need to give less weights - for example, 0.01
- Random measurements - to make localizing better - need to give high weights - for example, 10000

For ray racing, I chose step from [2 to 5], it gives good results, doesn't take a lot of time

Also, we check not every degree, but choose step size, making step every 10 degree gives good result. If we make steps bigger - accuracy is not very good. Smaller - computationally costly

5.3 Resampling

The derivation is now carried out by induction. The initial condition is trivial to verify, assuming that our first particle set is obtained by sampling the prior p(x0).

$$
\begin{aligned}
& p(x_t \mid x_{t-1}, u_t)\; bel(x_{0:t-1}) \\
& \quad = \quad p(x_t \mid x_{t-1}, u_t)\; p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1})
\end{aligned}
$$

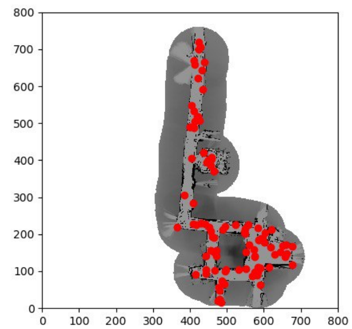
With

$$
\begin{aligned}
w_t^{[m]} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\
&= \frac{\eta\, p(z_t \mid x_t)\; p(x_t \mid x_{t-1}, u_t)\; p(x_{0:t-1} \mid z_{1:t-1}, u_{1:t-1})}{p(x_t \mid x_{t-1}, u_t)\; p(x_{0:t-1} \mid z_{0:t-1}, u_{0:t-1})} \\
&= \eta\, p(z_t \mid x_t)
\end{aligned}
$$

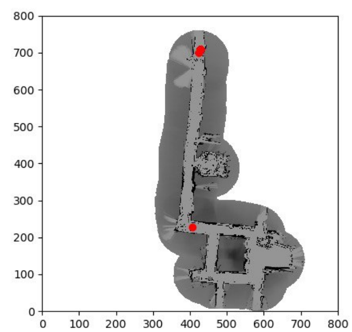
6 Performance

To analyze performance - tried different number of particles, firstly

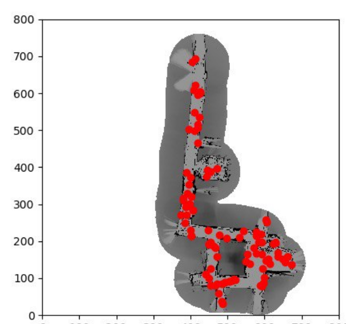
- 100. Computes fast, but results from different runnings are defferent



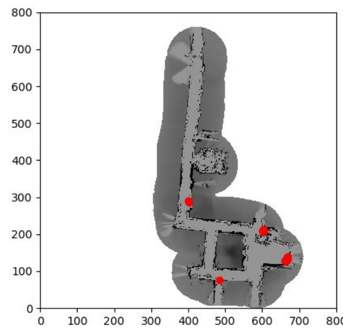
800
700
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700 800



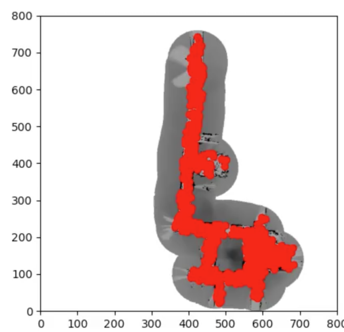
800
700
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700 800

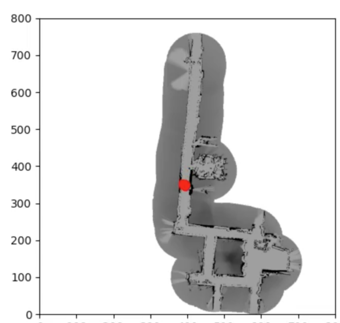


800
700
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700 800

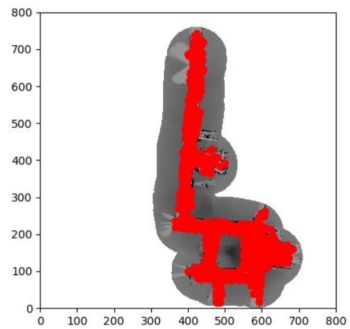


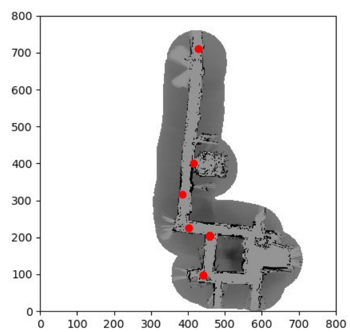
- 500-700. Gives good stable results

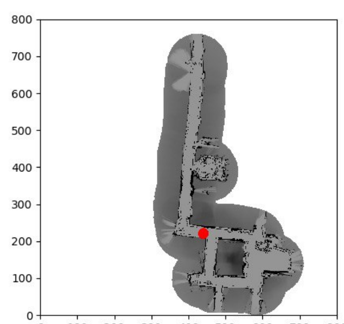




- 1000. Long computations. Results change fast, does not give good accuracy







7 Video

Here is a link: `https://drive.google.com/drive/folders/16vkpxjsUZzXYbzXUSPEhd6E_K6Fi4sff?usp=sharing`

8 Future work