Anna Lvova
BS17-RO-01

Homework 3

Robot Localization using Particle Filters

Motion Model

The implemented model is based on an algorithm from the proposed textbook, table 5.6.

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

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

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

$x' = x + \hat{\delta}_{\text{trans}}\ \cos(\theta + \hat{\delta}_{\text{rot1}})$
$y' = y + \hat{\delta}_{\text{trans}}\ \sin(\theta + \hat{\delta}_{\text{rot1}})$
$\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

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

Table 5.6 Algorithm for sampling from $p(x_t \mid u_t, x_{t-1})$ based on odometry information. Here the pose at time t is represented by $x_{t-1} = (x\ \ y\ \ \theta)^T$. The control is a differentiable set of two pose estimates obtained by the robot's odometer, $u_t = (\bar{x}_{t-1}\ \ \bar{x}_t)^T$, with $\bar{x}_{t-1} = (\bar{x}\ \ \bar{y}\ \ \bar{\theta})$ and $\bar{x}_t = (\bar{x}'\ \ \bar{y}'\ \ \bar{\theta}')$.

This algorithm relies on sampling the position from a probabilistic model rather than computing the exact probability for each position.

Sensor Model

The sensor model implementation is based on an algorithm from table 6.1.

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

Table 6.1 Algorithm for computing the likelihood of a range scan z_t, assuming conditional independence between the individual range measurements in the scan.

The algorithm assumes that the measurements have zero covariance or in other words independent, so that we can compute the final likelihood by multiplying each individual probability. The likelihood probability depends on the measurement noise, the probability to sense unexpected objects, sensor failures, maximum range measurements and unexplainable measurements.

Resampling Processs

I implemented two versions of the resampling.

The first one is the one based on an algorithm from table 4.4.

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 $\bar{\mathcal{X}}_t$

Table 4.4 Low variance resampling for the particle filter. This routine uses a single random number to sample from the particle set $\mathcal{X}$ with associated weights $\mathcal{W}$, 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 of the low variance sampler is that we only randomize one value, and then sample the particles based on that value. Variable called U plays an important role in the sampling, because each value of U points to exactly one particle. Due to this reason we can sample the particles by incrementing U.

The second implementation relies on the numpy random choose function. Although the second option is much simpler to implement, it is better in terms of performance. Hence, I decided to record the demo video using the second option.

Performance

The final implementation takes several hours for the proposed number of steps. It also resamples the particles closely together and often results in too few varying particles left in the set. This gets the algorithm to confuse and lose the robot's location.

Future Work

In my opinion, the part that needs revision is the resampling process. I tried to resample the particles only when we get the sensor measurements, and it resulted in increased performance. Seemingly, this part can be improved even more.

Moreover, the performance can be improved by refactoring and adjusting the hyperparameters.

I also need to remove the unnecessary timestamp visualizations, as it may radically slow the program.

Video

The speeded up part of the video is x64.

Links & References

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. MIT press, 2005.

[github repository]