SLAM-homework2

YA JU
juya@shanghaitech.edu.cn
2021533088

March 2024

1 Proof of particle filtering

weight of the m-th particle at time t is obtained by:

$$w_t^{[m]} = p(z_t | z_{1:t-1}, x_t^{[j]})$$

using conditional probability,

$$= \frac{p(x^{[j]} | z_{1:t-1}, z_t) p(z_t | z_{1:t-1})}{p(x_t^{[j]} | z_{1:t-1})}$$

since $x_t^{[j]}$ does not depends on z_t,

$$= \frac{p(x^{[j]} | z_{1:t-1}) p(z_t | z_{1:t-1})}{p(x_t^{[j]} | z_{1:t-1})}$$

$$= p(z_t | z_{1:t-1})$$

finally, odometry information and scan information are independent, so $u_{1:t}$ can be added:

$$= p(z_t | z_{1:t-1}, u_{1:t})$$

According to strong law of large numbers, since the particles at time t are identical independently distributed and they all subject to the same distribution where $w_t^{[m]} = p(z_t | z_{1:t-1}, u_{1:t})$, there should be:

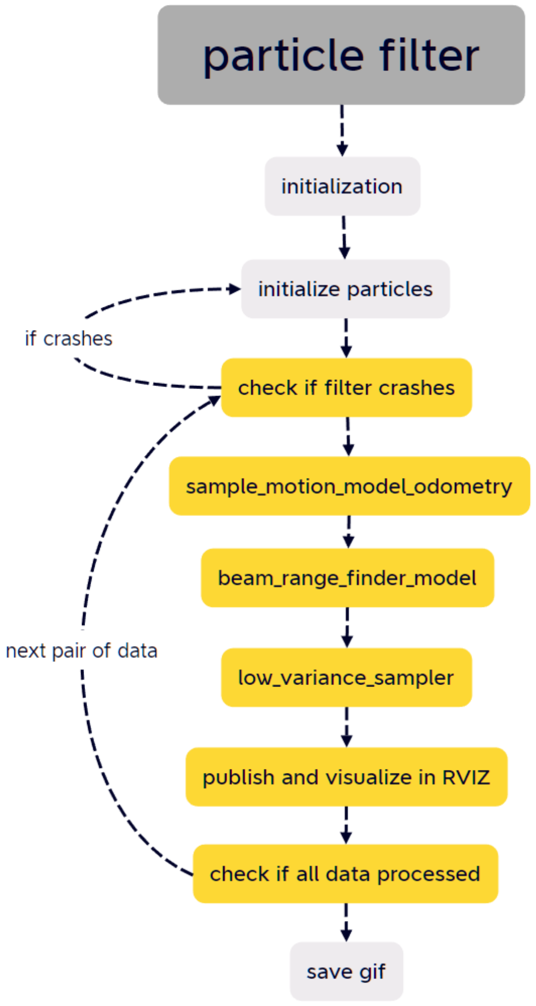
$$\lim_{M \to +\infty} \frac{1}{M} \sum_{m=0}^{M} w_t^{[m]} = p(z_t | z_{1:t-1}, u_{1:t})$$

2 Particle filtering

2.1 platform of implementation

I implemented particle filtering on Ubuntu using Python and ROS packages. Packages used are:
rosbag, rospy, numpy, yaml, random, math, sys, math, cv2, scipy, tf

2.2 structure of the filter



All of the modules are implemented exactly according to the methods in homework instructions, except for my self-designed synchronization function, it matches the closest odometry message in time to every scan message as ”data pairs” because the scan frequency is much lower than that of the odometry messages. This ensures that the odometry and scan data are properly synchronized for accurate localization.

As shown above, yellow part are in a loop over data pairs.

2.3 more technical details

1. The pose of the robot that is visualized is determined by the particle with the largest weight since it represents the most likely pose of the robot.

2. To improve running efficiency, particles that fall out of the grid or into occupied places are assigned very small weights and are then disregarded in the subsequent steps of the filtering process.

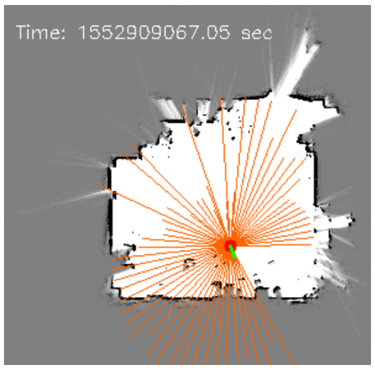
3. At first, the filter often crashes when the robot moves near the bottom of the grid. After discovering that, I add resample process to the filter, it will reinitialize particles when the filter crashes due to moving into walls. That's why you may see in my gif results the particles will suddenly spread to everywhere and cluster again after several loops.

2.4 experiments and final results

I've generated some GIFs in my homework folder so you can observe the performance of the particle filter.

Regarding efficiency, my Python-based filter, which relies heavily on loops, falls short of achieving real-time performance. Future improvements should focus on vectorizing the entire process to enhance its running speed.

Due to the filter's ability to detect when a particle lands in occupied areas and discard it, it tends to encounter crashes near walls. Fortunately, the resampling process can relocalize the robot's position. Therefore, I believe my filter has achieved a reasonable outcome.

An example of how the visualized filter looks like: 
Time: 1552909067.05 sec