ESE650 Learning in Robotics, 2018 Spring
Project 4: SLAM
Wudao Ling

Introduction

In this project, we are provided with dataset from Humanoid Robot THOR. And we need to achieve indoor Simultaneous Localization and Mapping (SLAM).
I used occupancy grid algorithm for mapping and particle filter for localization. Iteratively and alternately running mapping, localization and particle update led to a good SLAM performance in accuracy and speed.

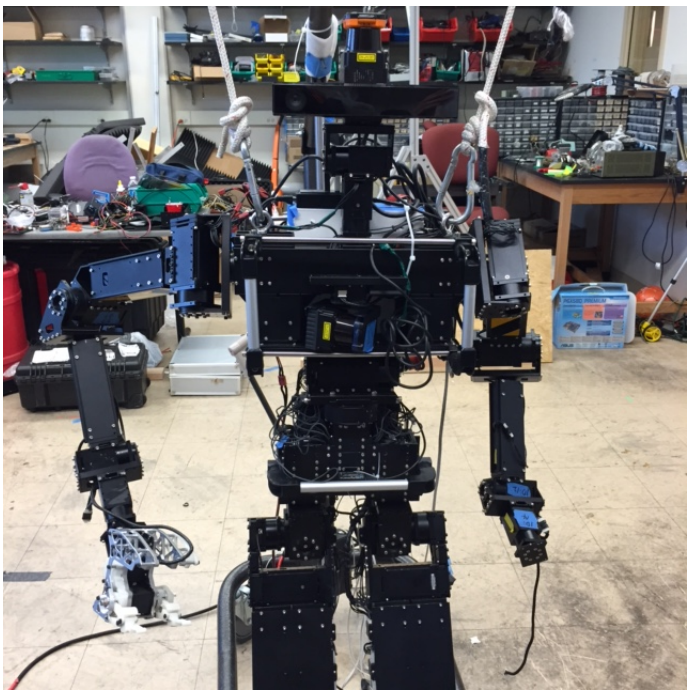


Figure 1: THOR robot

1 Dataset

The dataset contains various sensor data: angles from IMU, pose from encoder odometry, lidar scan and kinect data.
Since this project is on 2D, the robot's pose is represented as x, y and yaw. And the only data processing i did is to remove bias in IMU's yaw and odometry's xy, by substracting the first entry.
IMU angles are used for Homogeneous Transformation from body frame to world frame, while yaw also involves in localization prediction because it's more accurate than odometry's yaw. Also odometry's xy is for localization. Lidar scan is for mapping.

2 Occupancy Grid

Occupancy Grid uses log odds ratio to represent map, and usually comes with threshold to determine whether the position is occupied or free and bound that allows recovery.
When lidar hits a certain position, this grid as well as grids between it and lidar will be updated as follows:

$$logodd+ = log\frac{p(z|m_{x,y}=1)}{p(z|m_{x,y}=0)}$$

where z could be 1 or -1 depending on hit or no hit, and p depends on confidence on lidar.

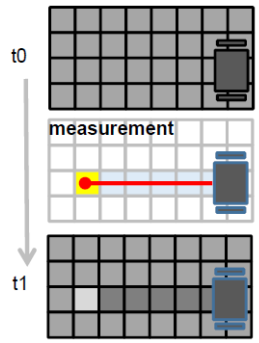


Figure 2: occupancy grid

3 Particle Filter

Particle Filter uses a set of particles to approximate belief distribution on localization. Each particles has its own pose and weights.
Particles are initialized with uniform weights, then odometry and noise are applied to predict their next pose. Later these particles generate hypothesis maps, the correlation-based matching between hypothesis and real map determines particles' weights.
In the update process, sometimes most of weights is on a few particles, thus resampling is needed. The criteria for resampling is $N_{eff} = \frac{\sum(weights)^2}{\sum(weights^2)}$, a higher N_{eff} indicate more uniform distribution and we should consider resampling when it drops under a threshold. The resampling method I use is stratified resampling.

Stratified resampling
Generate N ordered random numbers

$$u_k = \frac{(k-1) + \tilde{u}_k}{N}, \text{ with } \tilde{u}_k \sim \mathrm{U}[0, 1)$$

and use them to select x_k^* according to the multinomial distribution.

4 Implementation

Steps

Initialize map with 0s and N particles with pose [0,0,0] and uniform weights

1. Mapping
   (a) find particle with the highest weight, set its pose as robot pose
   (b) use pose's xy and IMU's angles to get transformation from body to world
   (c) use joint angles and robot spec to get transformation from lidar to body
   (d) convert lidar scan from polar to cartesian coordinate (in lidar frame)
   (e) transform lidar hit from lidar to world, remove hits that are too far, too close or hit on the ground
   (f) update map, add $logodd_{occ}$ to hit position and add $logodd_{free}$ to pass-through position. The pass-through region can be addressed by cv2.drawContours()
2. Localization Prediction
   (a) load odometry's xy and IMU's yaw as odometry
   (b) compute relative movement between previous and current odometry

(c) load particles' pose and apply relative movements, also apply some noise

3. Particle Update

   (a) for each particles, perform transformation from lidar to world like (b)-(e) in Mapping step to obtain their hits

   (b) compute correlation between these hypothesis and real map. An effective implementation is to calculate number of occupied grids that indexed by particle's hits

   (c) update particles' weight, logweights = log(weights) + correlation, then logweights are normalized and exponentialized

   (d) resample particles if necessary

Details

Map

I set 50m*50m as map size and 0.05m as map resolution. The occupied and free threshold are log odds of 0.9 and 0.2.
I use 0.7 as lidar confidence, which means $logodd_{occ} = log\frac{0.7}{1-0.7} = 0.85$. $logodd_{free} = log\frac{1-0.7}{0.7} * 0.5 = -0.42$ because usually we weigh more when a cell is detected as occupied.
And the bound for map is 100, this means there won't be any grid with log odds more than 100 or less than -100. We need to allow recovery in probabilistic robotics, especially when robot stays still, the algorithm would increase or decrease the same positions many times.

Particles

100 particles are utilized to track localization, and noise covariance on prediction is [0.005, 0.005, 0.005].
N_{eff} I apply is 10 percent of particle numbers, which is 10. However I found the weight update method (in log space) easily focus weights on 1 particle and leads to lots of resampling. I tried adding a temperature to the softmax function by dividing the correlations by some number. At the end I choose 10 as divisor, even though N_{eff} still stays around 1, the performance is good and larger divisor may cause other problem.

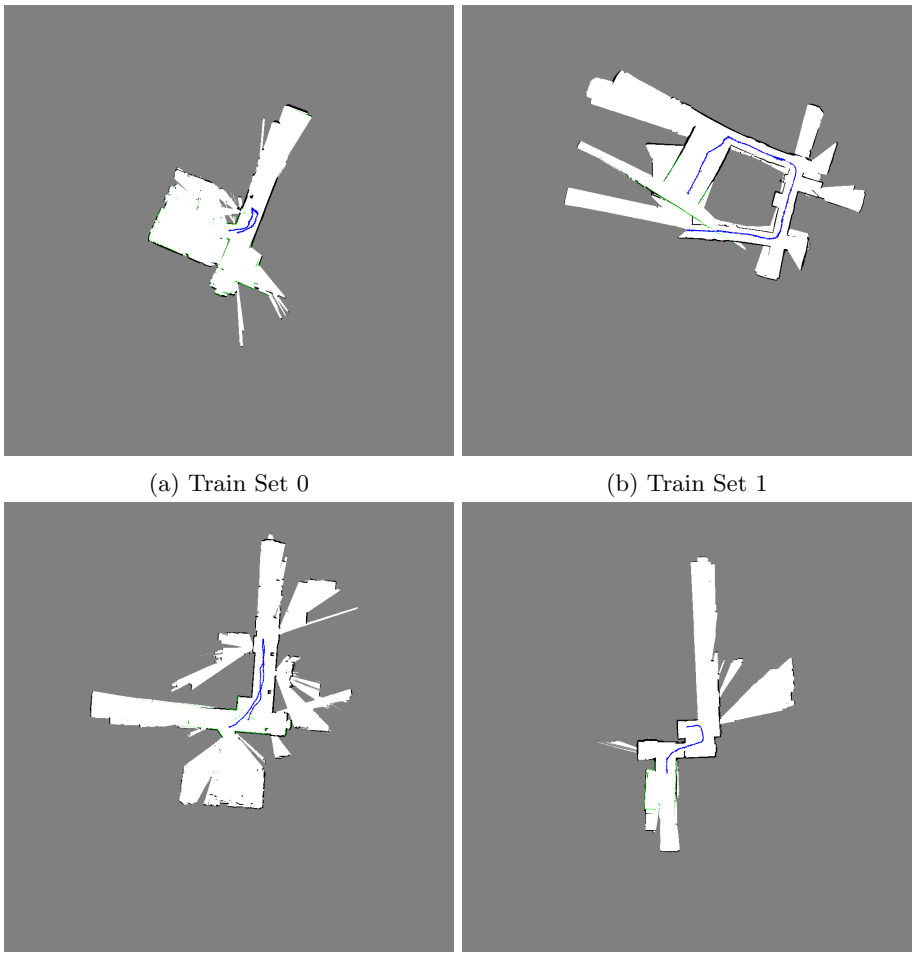
Speed

I introduced 2 intervals for speed. First is the plot interval, I choose to plot SLAM once in every 100 steps, and the update on map and particles continues with or without plot. Second is the SLAM interval, I find it's not necessary to use all sensor data, but this interval is a trade-off of accuracy. By trial and error on train set, I decide to update SLAM in every 5 steps. In this way, my program can run any train set within 1 minute.

Plot

In this project, I used OpenCV for plot. Therefore I need one more transformation from world frame to pixel frame, the world's origin is at center and its y is upward while pixel's origin is at upper left and its y points downward.
In my plot, occupied grids are black, free ones are white and undetermined grids are gray. My plot also include robot trajectory in blue and lidar scan in green, these are based on best particles.

5 Testing



(a) Train Set 0

(b) Train Set 1

(c) Train Set 2

(d) Train Set 3

Figure 3: Train Set

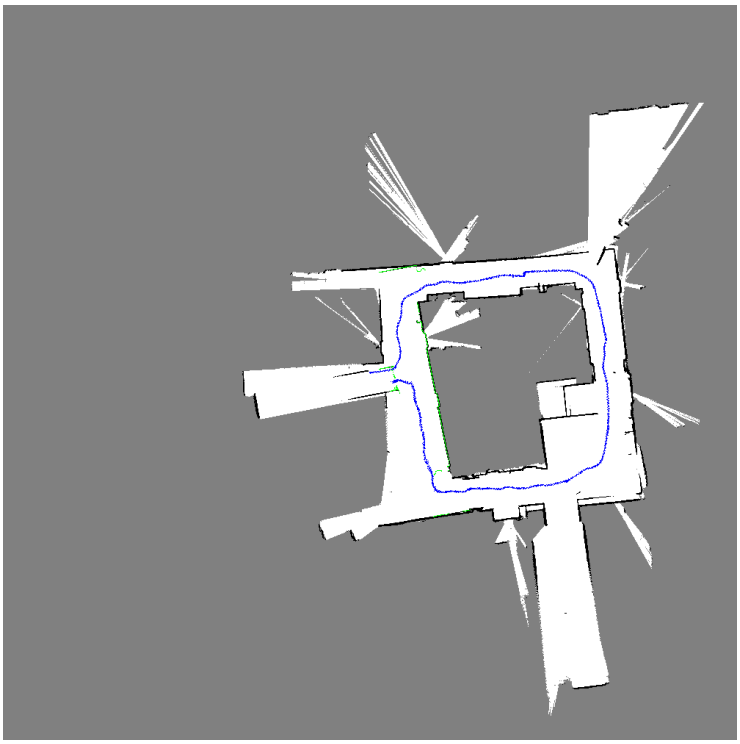


Figure 4: Test Set

As seen in Train Set 1 and Test Set, I should improve the performance on loop closure.
A change I can make is particle weight update, weights *= correlation then normalize. This method usually give us higher N_{eff} thus the localization distribution is more diverse than update in log space does, but it also needs tuning. Relatively, I could introduce a very low probability in motion model that robot could be anywhere, to address the problem of "kidnapped" robot.
Another approach is to recognize the loop and then refine localization and map. I can implement this by recording features from the initial pose and checking whether robot is hitting same set of features. With the loop, the localization and map could be adjusted sequentially.

References

[1] Thrun, S. Learning Occupancy Grid Maps with Forward Sensor Models *Autonomous Robots 15*, 2 (2003), 111–127.

[2] Thrun, S.. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium 1*, (2002), 1–35.

[3] Thrun, S., Fox, D., Burgard, W., and Dellaert, F. Robust monte carlo localization for mobile robots. *Artificial intelligence 128*, 1 (2001), 99–141.

[4] LaValle, S., Yershova, A., Katsev, M. and Antonov, M. Head tracking for the Oculus Rift *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (2014).

[5] Hol, J., Schon, T. and Gustafsson, F. On Resampling Algorithms for Particle Filters *Nonlinear Statistical Signal Processing Workshop, IEEE*, (2006).