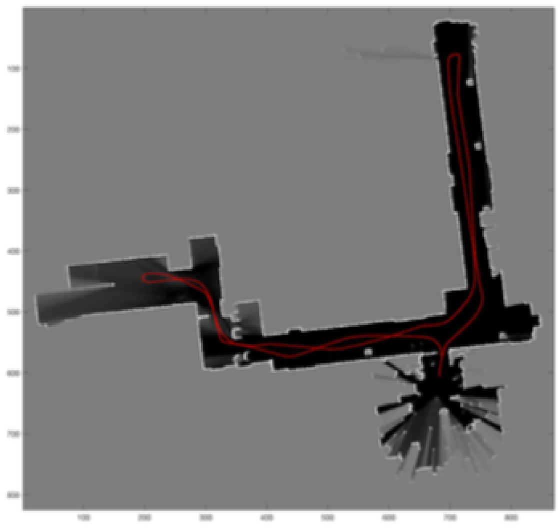
Final project on positioning

翦卓著 2021214425

Requirements

In this assignment you will be implementing a particle filter for pose tracking in 2D space. Imagine a robot wants to understand its position in order to determine where to next explore. The figure below depicts an example of robot localization from LIDAR measurements projected onto a map.



> 1. *You will complete a function that takes the set of parameters, sensor data and given map as input, and returns a the positions and orientations of the robot. The signature of the function is given as: posePF = particleLocalization(ranges[:,0:num-1], scanAngles, M, param)*
> 2. *The param variable will include the map resolution (param.resol) as the number of cells per meter, map size (param.size) as the number of cells in the map, and origin of the robot (param.origin) as the starting cell coordinates of the robot in the map.*
> 3. *Your function should return the entire path of the robot, in a 3 by n matrix, where n is the number of LIDAR observations made. You will keep track of the x, y position components and the theta angular component, in that order, for each n observations*
> 4. *example_test.py is provided to help visualize your result.*

Algorithm introduction

Particle filter (PF) is a method to solve nonlinear problems. The general idea is as follows:

1. Initial state: a large number of particles are used to simulate the motion state, so that the particles are evenly distributed in space;
2. Prediction stage: according to the state transition equation, plug in each particle to get a prediction particle;
3. Correction stage: the predicted particles are evaluated (the weight is calculated). The closer the particles are to the real state, the greater the weight is;
4. Resampling: the particle is screened according to the weight of the particle. In the screening process, a large number of particles with significant weight should be reserved, while a small number of particles with small weight should be selected;
5. Filtering: The re-sampled particles are put into the state transition equation to obtain the new predicted particles, namely step (2).

Experiment

Initialization

Lidar is taken as the measurement quantity, scanAngles represents the Angle parameter of lidar, and ranges represents the specific value of lidar. A total of N=3701 steps are run. We need to estimate the state quantity of the robot in N steps, expressed as

$$P_j = [x, y, \theta]^T$$

Use the system model to predict

$$x_{k+1} = x_k + radius * cos(\theta_k)$$
$$y_{k+1} = y_k + radius * sin(\theta_k)$$

```
P(1:2, :) = P(1:2, :) + radius * [cos(P(3, 1:M)); -sin(P(3, 1:M))];
```

Store the location information in an array of myPoses

```
% Number of poses to calculate
N = size(ranges, 2);

% Output format is [x1 x2, ...; y1, y2, ...; z1, z2, ...]
myPose = zeros(3, N);
```

Set the map parameters

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
% Map Parameters
%
% the number of grids for 1 meter.
myResolution = param.resol;
% the origin of the map in pixels
myOrigin = param.origin;

% The initial pose is given
myPose(:,1) = param.init_pose;
```

Set the motion model parameters of the robot

```
% System model parameters
noise_sigma = diag([0.05 0.05 0.03]); % System noise covariance
noise_mu = [0 0 0];
radius = 0.02; % Consider robot movement
```

Decide the number of particles, M.

```
M = 30;
% Create M number of particles
P = repmat(myPose(:,1), [1, M]);
```

The weights assigned to the particles in the initial phase are equal

$$W_j = 1/M$$

And set the initial states of M particles

```
P = repmat(myPose(:,1), [1, M]);
```

Particle filter

Propagate the particles

Add gaussian noise to the particle

```
for m = 1:M
P(:, m) = P(:, m) + mvnrnd(noise_mu, noise_sigma)';
```

Weight correction

Find grid cells hit by the rays (in the grid map coordinate frame)， find the obstacle cell

$$\begin{vmatrix} x_{1k} \\ x_{2k} \end{vmatrix} = \begin{vmatrix} d_k cos(\theta + \alpha_k) \\ -d_k sin(\theta + \alpha_k) \end{vmatrix} + \begin{vmatrix} x_1 \\ x_2 \end{vmatrix}$$

```
x_occ = ranges(:, j) .* cos(scanAngles + P(3, m)) + P(1, m);
y_occ = -1 * ranges(:, j) .* sin(scanAngles + P(3, m)) + P(2, m);
occ_id = ceil([x_occ, y_occ] * myResolution + repmat(myOrigin', [n,
1]));
```

For each particle, calculate the correlation scores of the particles， We calculate the weight according to the accuracy of occupied and free identification. The calculation formula is as follows

$$Score = w_1 * f(occupied) + w_2 * g(free)$$
$$f(occupied) = w_{11} * num(occupied\ is\ false) + w_{11} * num(occupied\ is\ true)$$
$$g(free) = w_{21} * num(free\ is\ false) + w_{22} * num(free\ is\ true)$$

We take the following values for the arguments

$$w_1 = 3$$
$$w_2 = 1$$
$$w_{11} = -5$$
$$w_{12} = 15$$
$$w_{21} = 1$$
$$w_{22} = -5$$

```
P_corr(m) = -5 * sum(map(lidar_occupied) <= map_threshold_low) + 15
*sum(map(lidar_occupied) >= map_threshold_high);
P_corr(m) = 3 * P_corr(m) + sum(map(lidar_free) <=
map_threshold_low) - 5 * sum(map(lidar_free) >=
map_threshold_high);
```

Update the particle weights

```
P_corr = P_corr - min(P_corr);
W = W .* P_corr;
W = W / sum(W);
```

Choose the best particle to update the pose, the position of the particle with the maximum weight is taken as the update position

```
[max_val, id] = max(W);
myPose(:, j) = P(:, id);
```

Resample

Resample if the effective number of particles is smaller than a threshold.At this time, the weight distribution of particles is more uniform, so the inverse of the sum of squares is smaller. If this is the case, then we replace the positions and weights of the particles with those of the particles with the highest weights

```
resampling_factor = sum(W)^2 / sum(W.^2)
threshold = 0.8
if (resampling_factor<threshold*M)
idx=zeros(M,1);
```

```
[a,b]=sort(W);
c=cumsum(a);
for k=1:M
idx(k)=b(find(rand<=c,1,'first'));
end

P=P(:,idx);
W=W(idx);
W=W/sum(W);

replace = 10;
for k=M-replace:M
P(:,k)=P(:,1);
W(k)=W(1);
end
end
W = W / sum(W);
```

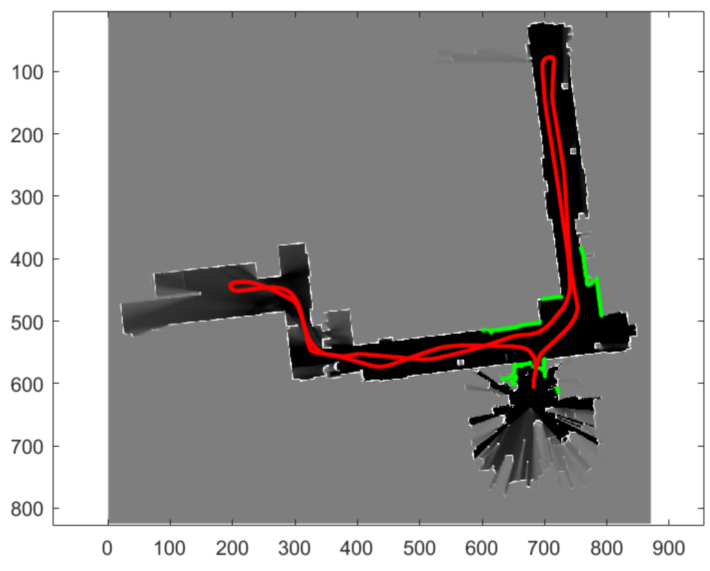
Predict pose

Then the re-sampled particles are put into the state transition equation to obtain the new predicted particles.

```
P = repmat(myPose(:,j-1), [1, M]);
P(1:2, :) = P(1:2, :) + radius * [cos(P(3, 1:M)); -sin(P(3, 1:M))];
```

Experimental result

We take the actual trajectory of the robot as a reference，and run the first 1800 steps. Here's the real trajectory

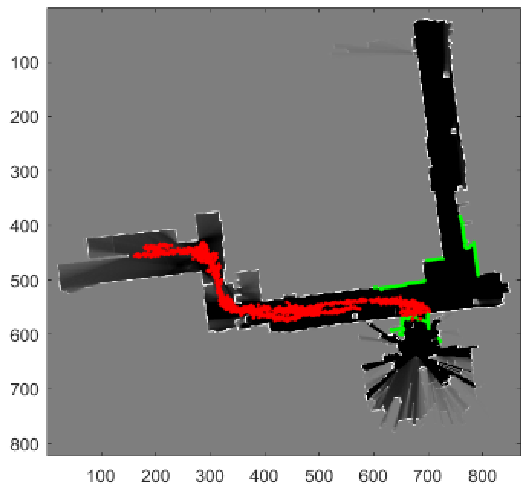


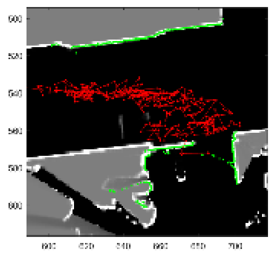
During the experiment, the variance matrix of the system was uniformly set as

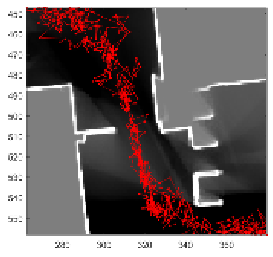
$$\begin{bmatrix} 0.05 & 0 & 0 \\ 0 & 0.05 & 0 \\ 0 & 0 & 0.03 \end{bmatrix}$$

EXPERIMENT NUMBER	PARTICLE NUMBER	RESAMPLING THRESHOLD	REPLACE NUMBER	TIME	PERFORMANCE
1	50	0.8	10	64min	Point cloud matching is stable, but occasionally there is a large pose deviation
2	30	0.7	3	23min	Lost position at the corner
3	40	0.7	5	40min	Point cloud matching is relatively accurate, without large deviation
4	45	0.7	5	48min	Matching is poor;missing

Expriment No.1

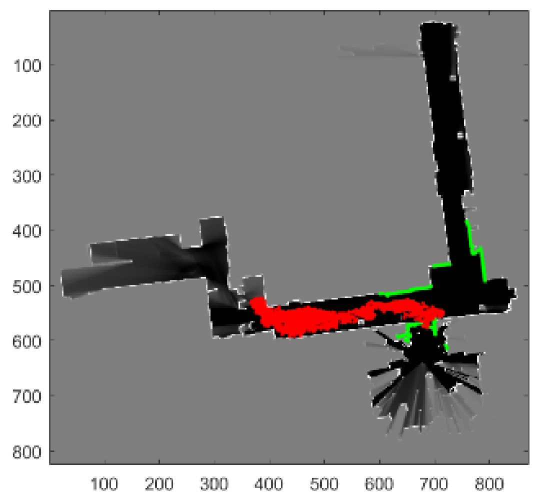
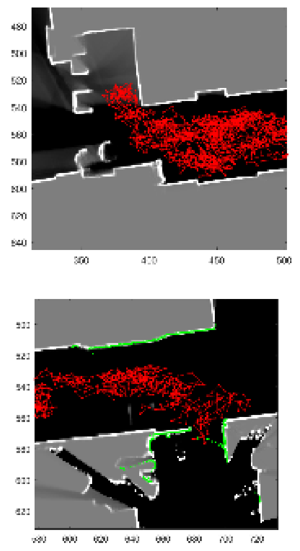






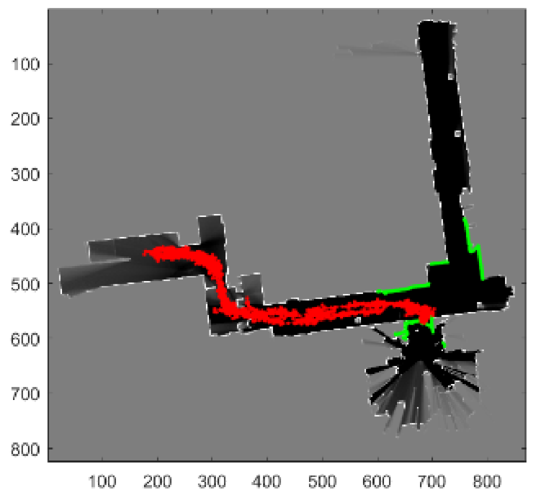
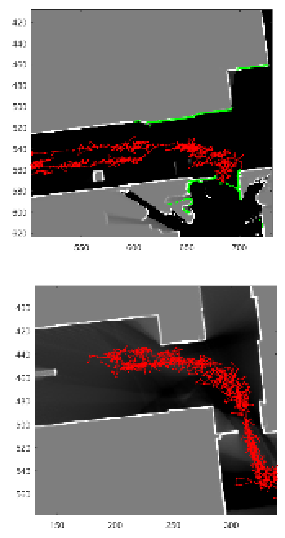
Expriment No.1 has the largest number of particles, and it also take the most time. In the process of expriment, there is no great deviation between the position and the actual trajectory.

Expriment No.2

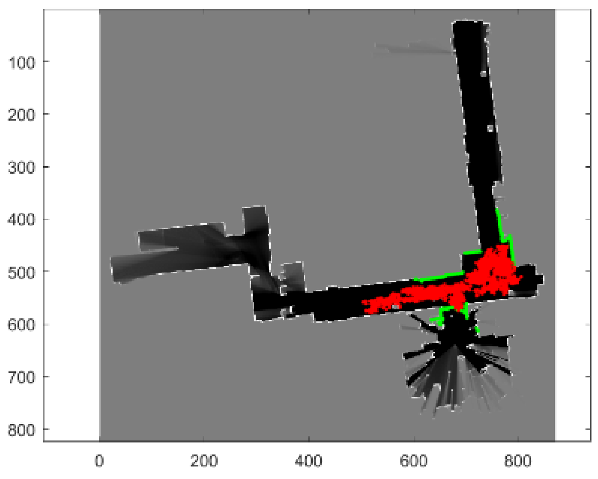
Lost its position on the corner, my analysis is due to the low number of particles. But it's a lot faster than experiment No.1.

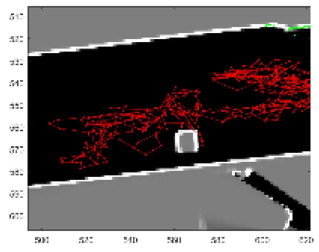
Expriment No.3

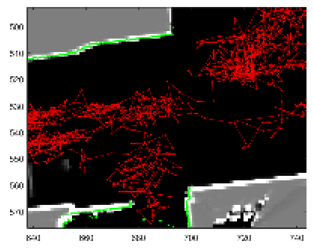



Taking the total number of particles as 40 can not only ensure the relatively small time of operation, but also ensure the accuracy of the basic position.

Expriment No.4







The missing condition occurred ,I think it's possible that the comparison resampling process is too simple.

All the code is in the attachment

Conclusion

1. It can be seen that due to the existence of variance, errors are inevitable when using particle filters to estimate the state of an object. However, an appropriate increase in the number of particles can reduce the occurrence of loss of position, but increasing the number of particles will also result in a waste of computing resources, for real-time requirements are not suitable for systems with higher real-time requirements.
2. For correction links and resampling, different strategies can be adopted to improve the accuracy of the filter's state estimation. For different scenarios, the parameters can be adjusted to improve the filter's performance.
3. Due to weight sampling, there is a dearth of particles and some data is lost, which is one of the reasons for the loss of location.