ECE 276A
Project 3

SLAM

Due Date: Dec 11, 2017

Professor: Nikolay Atanasov

Chuqiao Song

A53239614

Introduction

In robotics and mapping area, Simultaneous localization and mapping (SLAM) is an important way to construct a map of the unknown environment and keep tracking where the robot is. This is kind like a chicken and egg type problem, the SLAM solve this problem simultaneously. Such computational way is implemented on driverless car, drones, and other intelligent devices having movement.

In this SLAM project, there are four steps, Mapping, Prediction, Update, and Texture map. Predication and update is achieved by particle filter to know where the robot is at one timestamp. By knowing where the robot is with the LIDAR information, the map of the space can be constructed at this timestamp. Meanwhile with the knowing robots location, and RGBD data, the occupancy grid map can be colored by floor texture. With iteration of above four steps, the map of the space can be constructed by the robot.

Problem Formulation

Give the training set of LIDAR data, which contains the beams' length and angles shoot from the laser, training set of joint data, which contains neck angles, and head angles, and training set of camera which is used to build texture map, the goal of this project is simultaneously tracking the robot and construct the map of given space, and then color the map with RGB data from camera.

1) Mapping: Given, LIDAR pose, head-neck angles, scanning data, with the particles which represent the pose of the robot, the scanning beams can be transformed from lidar frame to world frame to construct occupancy grid map.

2) Particle filter: There are two steps in particle filter, predict and update the state-information particles to find the location and orientation of robot.

3) resampling: By using the particle filter, sometimes the particles will be diming because of very small weight, and most weight accumulate on a few particles; thus, the resampling step is need, to redistribute the weight on particles.

4) Texture mapping: Stick image RGB data from RGBD camera to the occupancy grid map to texture the map.

Technical Approach

A. Mapping

Step up the occupancy grid map information such as size, resolution, and initialize it by zero array in 3D. At one timestamp, extract head and neck angles from joint data set, and formulate a transformation from lidar frame to body frame b_T_l. (h1 = 0.15, h2 = 0.33)

$$ {}_bT_l = {}_bT_h * {}_hT_l = \begin{bmatrix} R_z(neck) * R_y(head) & \begin{pmatrix} 0 \\ 0 \\ h_1 \end{pmatrix} \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} I_{3\times3} & \begin{pmatrix} 0 \\ 0 \\ h_2 \end{pmatrix} \\ 0 & 1 \end{bmatrix} $$

Also, it is important to remove the wired data from 'scan' which the beam length is greater than 30 meters and 0.1 meters, which means the beam is too far away or too much close to the robot. Then use the transformation b_T_l, and most weighted particle which formulating the transformation w_T_b, to transform the scan beam from lidar frame to world frame.

By using bresenham2D algorithm, and most weighted particle, the occupancy grid map can be built. To build this map, the log-odd should be used to calculate the probability of whether the map is occupied or empty that beam can go through, i.e. when cells is occupied plus log4, and if cell is free, minus log4. After doing these step, recover the log-odd to probability, and if the pdf >0.8, give the cell value 1, pdf<0.2 give the cell value -1, otherwise 0 is assigned.

mi = mi + log(1/b), free
mi = mi + log(b), occupied
P(mi) = 1 – 1/(1+exp(mi)) for recover step

B. Particle filter

Predication:

First initialize 100 particles with normal weights at $\mu_{0|0}^{(i)} = (0,0,0),\ \ \alpha_{0|0}^{(i)} = \frac{1}{N},\ \ i = 1,\ldots,N$

Then, by using the lidar pose to formulate w_T_l at given timestamp, and b_T_l, the transformation from body to world frame can be achieve w_T_b. By using this transformation, the Odometry of particles can be found at for t+1, Ot+1, and with the previous Odometry Ot, the delta pose of the particles can be found. In this case the delta odometry can be used to predict the 100 particles.

Delat = Ot+1 - Ot

$$u_{t+1}^{(i)} = u_t^{(i)} \oplus (O_{t+1} \ominus O_t) \oplus w^i$$

$$w^{(i)} \sim \mathcal{N}(0, \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}), i = 1,\ldots,100$$

sigmax = 0.095, sigmay = 0.095, sigma_theta = 0.095

update:

Remove the scan beams that are too far away, too close to the robot and the beams hit to ground. By using these scan beams, the map correlation can be done for each particle at this update step, such that the weight for each particle can be updated. Also, we modify the pose of the particles at this step based on location of the largest value in correlation matrix. m here means map

$$p_h(z_t|x,m) = \frac{exp(corr(z_t,m))}{\sum_z exp(corr(z,m))} = \text{softmax}(corr(z,m))$$
$$corr(z,m) = [corr(z_1,m), corr(z_2,m), \cdots, corr(z_n,m)]$$

$$\alpha_{t+1|t+1}^{(k)} = \frac{\alpha_{t+1|t}^{(k)} p_h(z_t|x,m)}{\sum_j \alpha_{t+1|t}^{(j)} p_h(z_t|x^j,m)}$$

C. Resampling

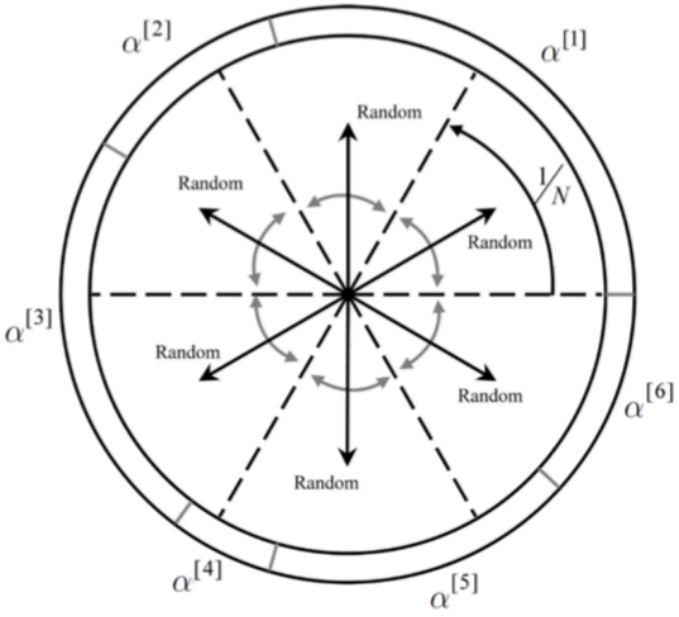
In some case, most particles will be diming, which means their weight becomes very small, and most weight are now accumulated onto small number of particles. In such case, the resampling step is needed to recreate most weighted particles with equally weighted, but still 100 particles in this step in my project.

The threshold for resampling is defined as 5 and comparing with Neff. $N_{eff} := \frac{1}{\sum_{k=1}^{N_{t|t}} \left(\alpha_{t|t}^{(k)}\right)^2}$

Stratified Resampling

Stratified (low variance) resampling

1: **Input**: particle set $\{\mu^{(k)}, \alpha^{(k)}\}_{k=1}^{N}$
2: **Output**: resampled particle set
3: $j \leftarrow 1,\ c \leftarrow \alpha^{(1)}$
4: **for** $k = 1, \ldots, N$ **do**
5: $\quad u \sim \mathcal{U}(0, \frac{1}{N})$
6: $\quad \beta = u + \frac{k-1}{N}$
7: $\quad$ **while** $\beta > c$ **do**
8: $\quad\quad j = j + 1,\ c = c + \alpha^{(j)}$
9: $\quad$ add $(\mu^{(j)}, \frac{1}{N})$ to the new set



D. Texture mapping

Given the RGBD data, I can know the depth and corresponding RGB value. Thus, the first step in texture mapping is to load depth image dt, and transform it into focal plane, IR optical frame, RGB optical frame, RGB focal plane, and finally into the RGB pixel frame. In this case, I can find the location of the RGB value, which corresponding to the depth value from IR frame in pixel frame.

1. IR_pixel frame to IR focal plane frame:

(Xo/Zo, Yo/Zo, 1) = inv(IR_Calib) * (u, v, 1), IR_Calib is the calibration matrix for IR camera

2. IR focal plane frame to IR optical frame:

(Xo, Yo, Zo) = (Xo/Zo, Yo/Zo, 1) *Zo, where Zo is the depth stored in depth graph.

3. IR optical frame to RGB optical frame

RGB_ (Xo, Yo, Zo) = getExtrinsics_IR_RGB * IR_ (Xo, Yo, Zo)

4. RGB optical frame to RGB focal plane

RGB_ (Xo/Zo, Yo/Zo, 1)

5. RGB focal plane to RGB pixel frame
RGB _ (u, v, 1) = RGB_Calib * RGB_ (Xo/Zo, Yo/Zo, 1), RGB_Calib is the calibration matrix for RGB camera.
Then by using selected RGB value and corresponding depth, I can recover RGB value into 3D world frame, with transformation of camera to body frame, and transformation of body to world frame which is calculated by most weighted particle, bTc, wTb, cTo. Where cTo is inverse of

$$R_{oc} := \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

Other formula reference is like following.

- **Extrinsics**:

$$\begin{pmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{pmatrix} = \begin{bmatrix} R_{oc}R_{wc}^T & -R_{oc}R_{wc}^T p_{wc} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}}_{\text{pixels}} = \underbrace{\begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix}}_{\text{calibration: } K} \underbrace{\frac{1}{Z_o} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{canonical projection: } \Pi_0} \begin{pmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{pmatrix}$$

finally, project the color with Zw < 0.1 onto occupancy grid map form the colored map.

Result and Discussion

In this project, the noise is being used to modify the angle of particles, and with odometery to modify the location of the particles. The clearest part is figure1 and figure4, since the robot does not move so much, the performance of the particles is very well, and it is clear to see the robot is scanning a hallway and a classroom. The noise added on particles can be easily adjusted by correlation part in the update step with the map from previous cycle, such that update the weight of each particle, and find the best particle for map construction.

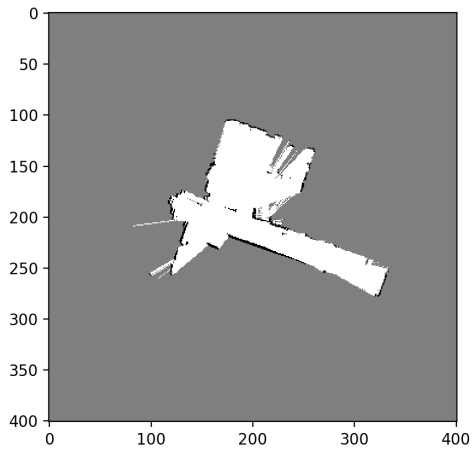


figure1: training set0

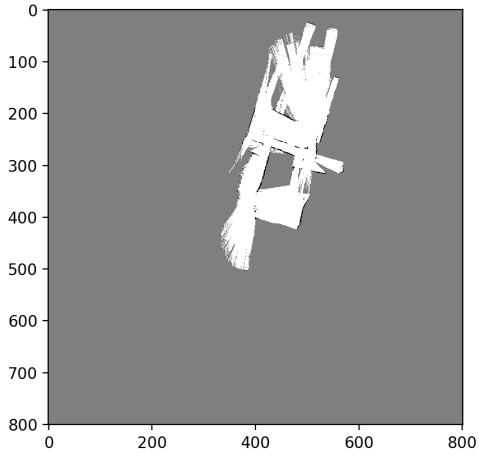


figure2: training set1

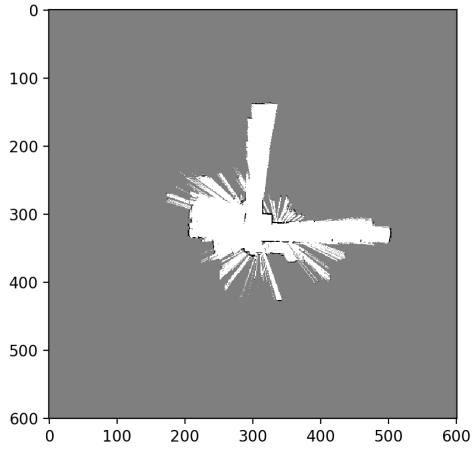


figure3: training set2

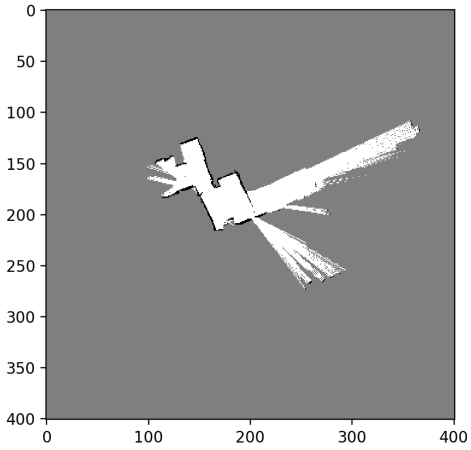


figure4: training set3

However, in figure2 and figure3, the movement of a robot is very large, and the modification part in update step does not have good performance. The reason may come from that the noise I added to modify the pose of particle is too much large, which affects the performace of the correlation part. I tried many combinations of noise, and these are best graphs I can show with one noise to run all training sets and test set. Another reason may be that the Odometry measurement is not accurate enough within an interval, which leads to predict wrong pose in the particle filter within this time interval, the map correlation with the mapping step may even aggravate the worseness. There are some ways to modify this problem; for example, increasing the number particles or in the update step not only update the location of the particles and weight, but also update the angles. i.e. for one particle with fixed angle, extending to range of angles with same location state. But this way really cost much computation, and decrease the efficiency of the code.

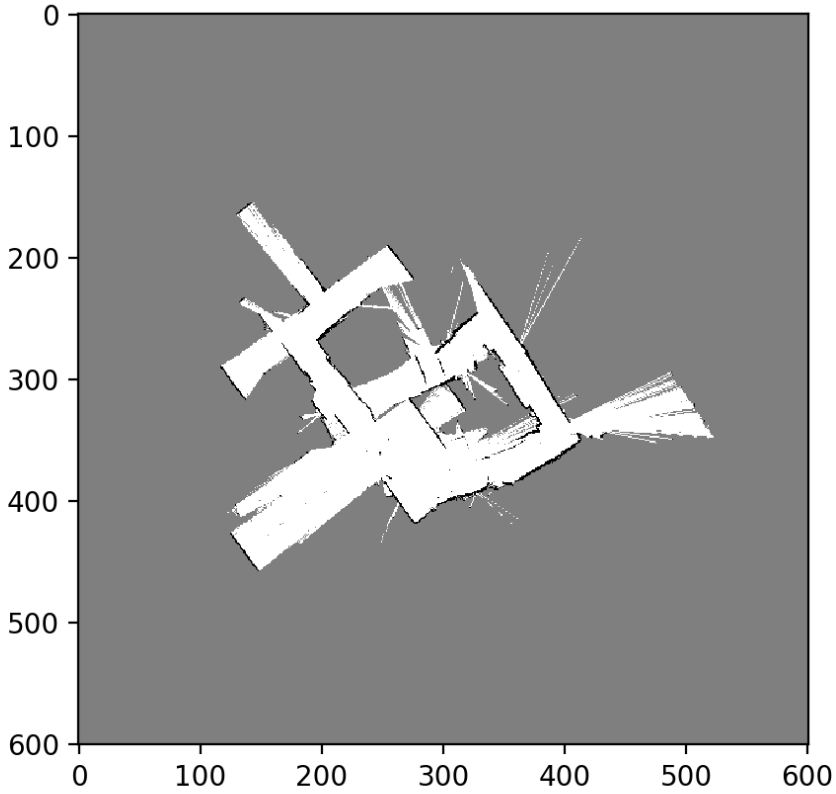


figure5: test set

The figure5 shows result of the test set, and it looks clear that the robot is moving in a hallway. The particle filter and mapping works very well under this 0.095 sigma modification noise, and seems like the noise in odometry is modified by each correlation step.

Texture Mapping

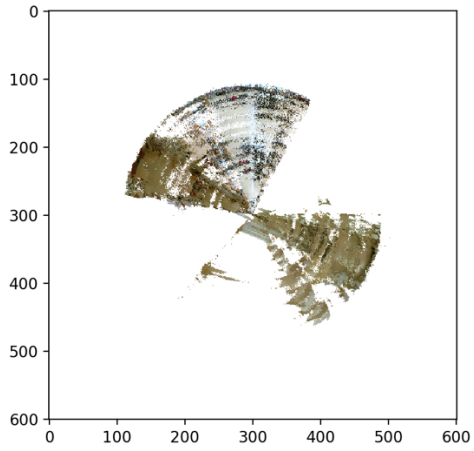


figure6: texture map of train0

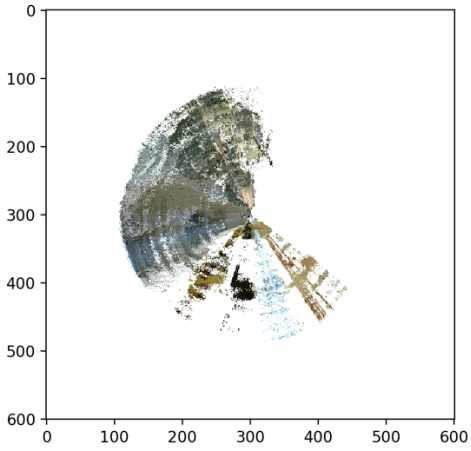


figure7: texture map of train3

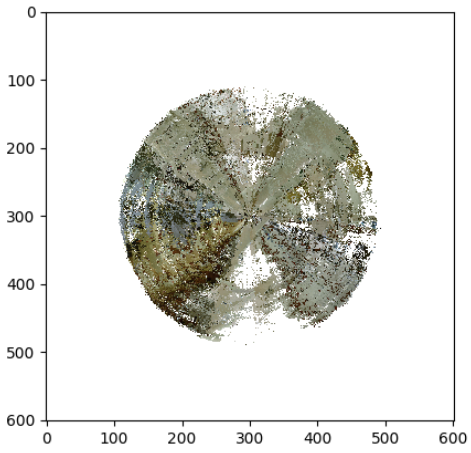


figure8: texture map of test set

There is some drifting of texture map, and it may be caused by the performance of the particle filter, and the noise on the Kinect as well as the robot movement. To improve the quality of the texture, I have to improve my particle with IMU data to let RGB data stick onto floor more accurate.

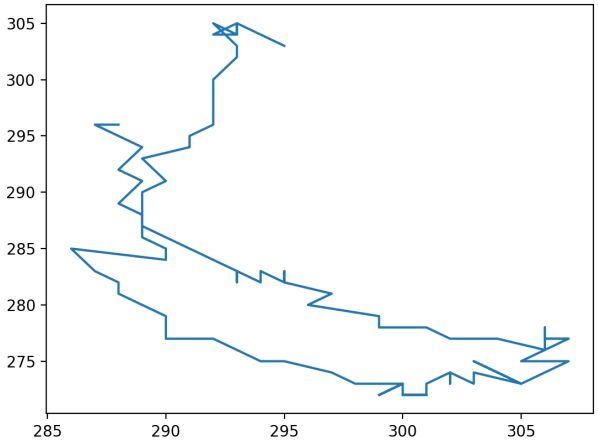


figure9 trajectory0

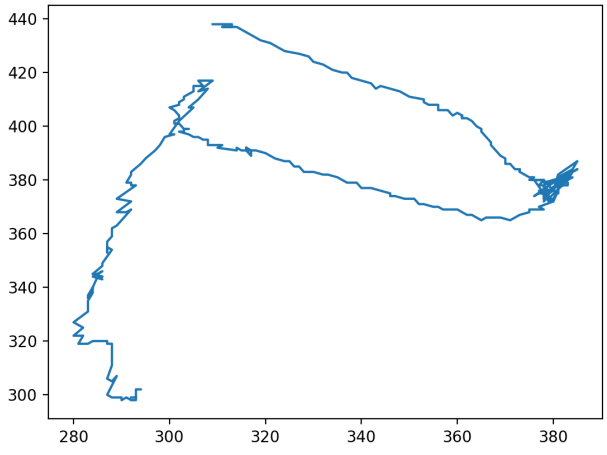


figure10: trajectory1

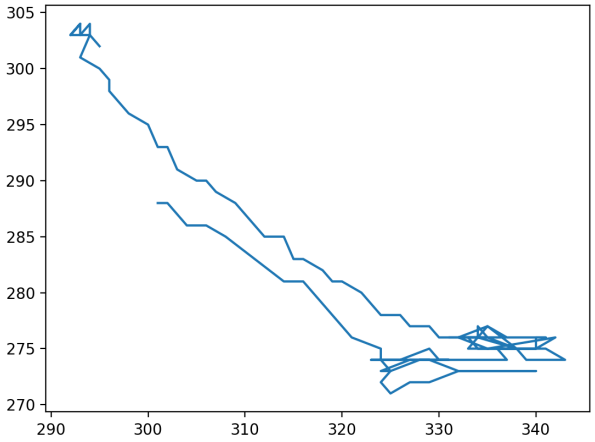


figure11: trajectory2

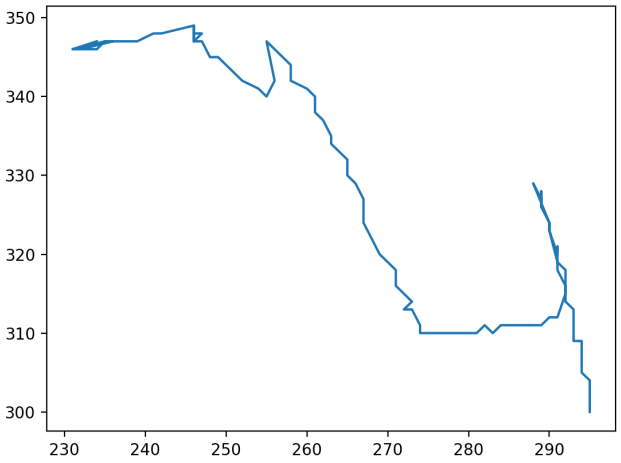


figure11: trajectory 3

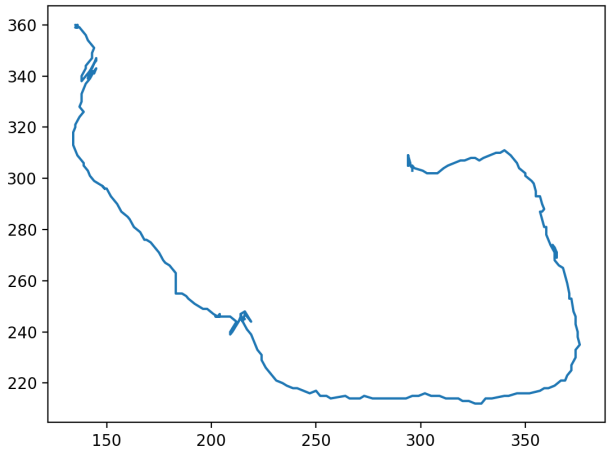
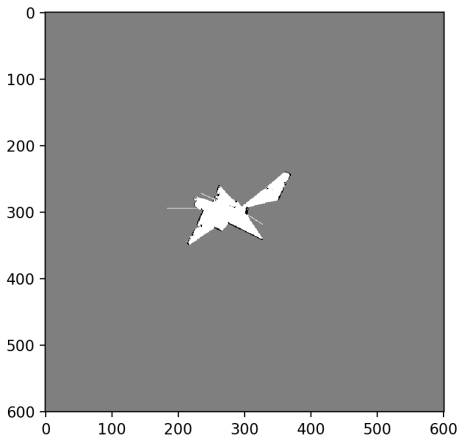
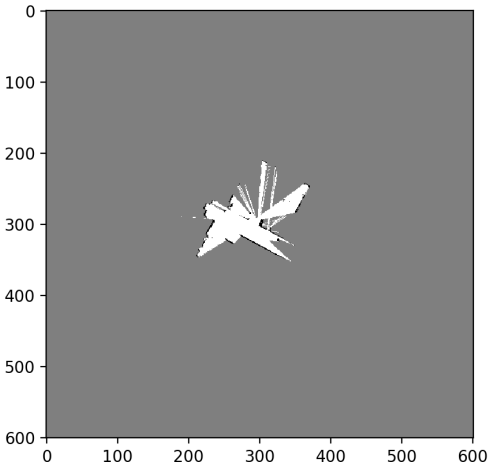


figure12: trajectory test

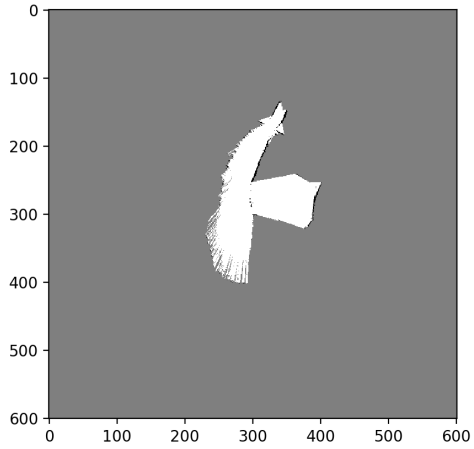
middle step

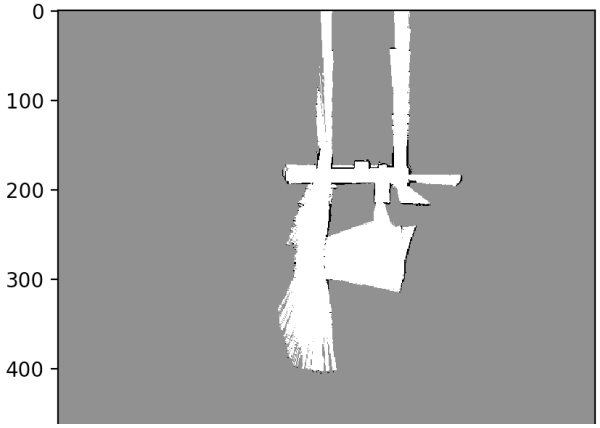
training set0



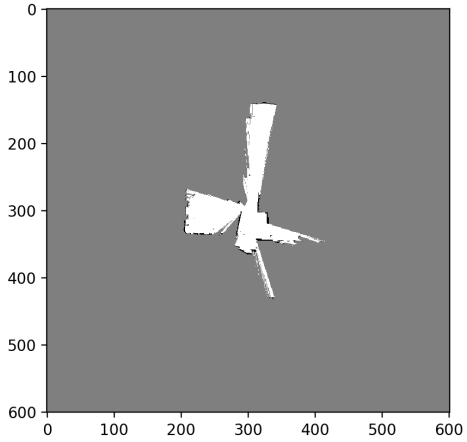


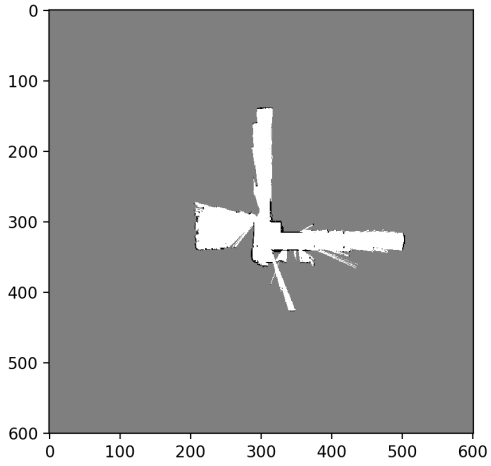
training set1



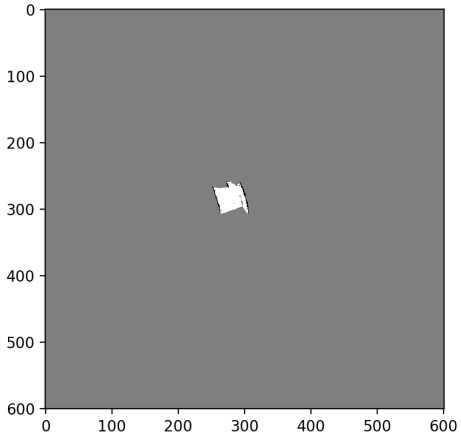


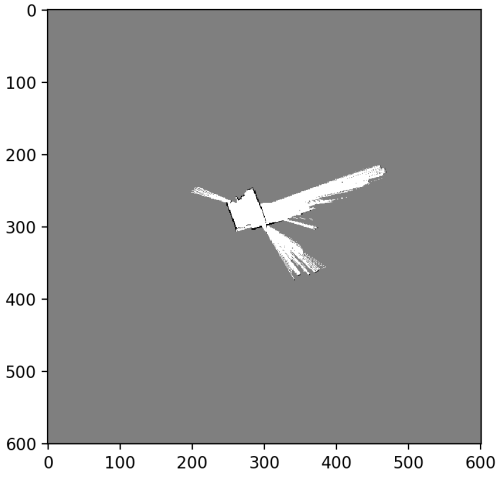
training set2



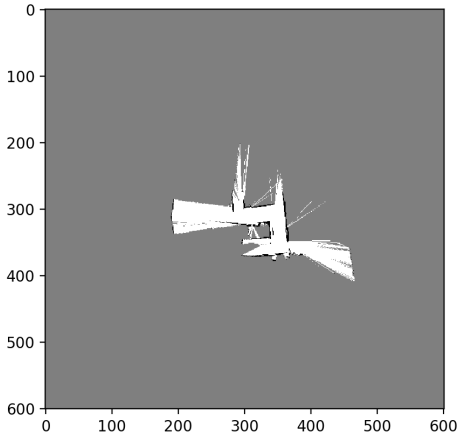


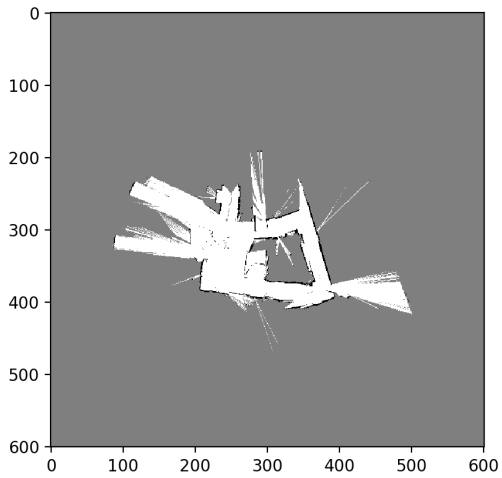
training set3





test set


0
100
200
300
400
500
600
0
100
200
300
400
500
600


0
100
200
300
400
500
600
0
100
200
300
400
500
600