LIDAR-BASED PARTICLE FILTER SLAM:A PROJECT REPORT

Renwei Bai

Department of Mechanical and Aerospace Engineering
University of California, San Diego
rbai@.ucsd.edu

1. INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is a parameter estimation problem, which contains two parts performed simultaneously, localization and mapping. SLAM algorithm has been applied in many areas, such as indoor robots, AR, unmanned aerial vehicle and automatic vehicles. In different areas, SLAM algorithm can implement different functions. For example, the sweeping robot uses the SLAM algorithm to identify and model different scenes and calculate its own location in these scenes. In UAV field, active obstacle avoidance is a very typical weak application of SLAM. The drone only needs to know where the obstacle is to plan the trajectory and bypass the obstacle.
In this project, we will use data from the front 2-D LiDAR scanner, fiber optic gyro (FOG) and encoders for localization and mapping as well as the stereo cameras for texture mapping.
LiDAR is a Light Detection And Ranging device, providing us the observation model of the robot in this project. FOG can detect the rotation angles in **XYZ** axis. A magnetic encoder consists of a rotating gear,a permanent magnet and a sensing element. It can tell us the The distance in **x** direction traveled during time for a given encoder count **z**, wheel diameter **d** and numbers of ticks per revolution.With the data from FOG and encoder, we can calculate the trajectory of the robot. Stereo Camera are two perspective cameras rigidly connected to one another with a known transformation, which can determine the depth of a point from a single stereo observation.
The rest of paper is arranged as follows. First I give the detailed formulations of SLAM problems in Section II. And then I will describe my technical approach to SLAM and texture mapping in section III. At least, I will present my results, and discuss them.

2. PROBLEM FORMULATIONS

2.1. SLAM Problem

The SLAM algorithm can deal with two parts of the problem, localization and mapping simultaneously. In this project, localization is to infer the robot state trajectory $\mathbf{x}_{0:T}$, given given a map $\mathbf{m}$ a sequence of control inputs $\mathbf{u}_{0:T-1}$, and a sequence of measurements $\mathbf{z}_{0:T}$. The trajectory $\mathbf{x}_{0:T}$ is the motion model of the robot. It can be written in the form of a nonlinear function or equivalently a probability density function pf that describes the motion of the robot to a new state $\mathbf{x}_{t+1}$ after applying control input $\mathbf{u}_t$ at state $\mathbf{x}_t$:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \sim p_f(\cdot \mid \mathbf{x}_t, \mathbf{u}_t) \; (1)$$

where $\mathbf{w}_t$ is motion noise.

The robot motion model also depend on mt and the map m has its own motion model:

$$\mathbf{m}_{t+1} = a(\mathbf{m}_t, \mathbf{x}_t, \mathbf{noise}_t) \sim p_a(\cdot \mid \mathbf{m}_t, \mathbf{x}_t) \; (2)$$

From data set,we can obtain the input $\mathbf{u}_t$ as two parts, the linear velocity $\mathbf{v}_t$ from encoder and angular velocity ω_t from FOG:

$$\mathbf{x}_{t+1|t} = f(\mathbf{x}_{t|t}, \mathbf{u}_t + \epsilon_t) \; (3)$$

where $f(\mathbf{x}, \mathbf{u}_t)$ is the differential-drive motion model, $\mathbf{u}_t = (v_t, \omega_t)$ is the linear and angular velocity input,

$$\epsilon_t \sim \boldsymbol{N}\left(0, \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}\right) (4)$$

is a 2-D Gaussian motion noise.

The mapping part is use Occupancy Grid Map Method to build a map $\mathbf{m}$ of the environment,given a robot state trajectory $\mathbf{x}_{0:T}$ and a sequence of measurements $\mathbf{z}_{0:T}$. Occupancy grid map is a vector $\mathbf{m} \in R^n$, whose i-th entry indicates whether the i-th cell is free ($\mathbf{m_i} = -1$) or occupied ($\mathbf{m_i} = 1$).

2.2. Texture Problem

After we having the 2-D map, we need to use the stereo camera information to add RGB texture on my 2-D map. The texture problem is that given the stereo camera's position $\mathbf{P} \in R^3$ and orientation $\mathbf{R} \in SO(3)$ with with a disparity measurement $\mathbf{d}$, try to estimate depth for each pixel via the stereo camera model.

3. TECHNICAL APPROACH

In this section we discuss about the algorithms and the models used in this project.

3.1. SLAM Problem

The two parts in SLAM problem look like a chicken-and-egg problem—if we want to build a map, we need a robot state trajectory and we need to use a map to update our robot state trajectory.

Mapping: At first, we use the lidar data on the first timestamp to obtain an initial map.Here are two points to call our attention. Firstly, we need to remove scan points that are too close or too far. Because if the points are out of the range of lidar scanner, the distance between the point and the lidar will be recorded as 0.Secondly, the data we get from lidar scanner is the position of the points to the lidar. But the data we use to calculate the map directly is the positions of the points to the world frame$\{\mathbf{W}\}$.So we need to convert the pose ${}_L\mathrm{T}_{\mathbf{P}_n}$ to ${}_w\mathrm{T}_{\mathbf{P}_n}$ (**L** means the lidar frame, **W** means the world frame, $\mathbf{P}_n$ means scanned points):

$$_W\mathrm{T}^{(k)}_{\mathbf{P}_n} =_W \mathrm{T}^{(k)}_{\mathbf{L}} \cdot_L \mathrm{T}^{(k)}_{\mathbf{P}_n} =_W \mathrm{T}^{(k)}_{\mathbf{V}} \cdot_V \mathrm{T}^{(k)}_{\mathbf{L}} \cdot_L \mathrm{T}^{(k)}_{\mathbf{P}_n}$$

$$=_W \mathrm{T}^{(k)}_{\mathbf{F}} \cdot_F \mathrm{T}^{(k)}_{\mathbf{V}_0} \cdot_V \mathrm{T}^{(k)}_{\mathbf{L}} \cdot_L \mathrm{T}^{(k)}_{\mathbf{P}_n} =_W \mathrm{T}^{(k)}_{\mathbf{F}} \cdot_{V_0} \mathrm{T}^{-1(k)}_{\mathbf{F}} \cdot_V \mathrm{T}^{(k)}_{\mathbf{L}} \cdot_L \mathrm{T}^{(k)}_{\mathbf{P}_n}$$

$$= \begin{bmatrix} cos(\omega_t) & -sin(\omega_t) & 0 & \mathbf{v}_t \\ sin(\omega_t) & cos(\omega_t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -0.335 \\ 0 & 1 & 0 & -0.035 \\ 0 & 0 & 1 & 0.78 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \cdot$$

$$\begin{bmatrix} 0.00130 & 0.79610 & 0.60517 & 0.8349 \\ 0.99999 & -0.00042 & -0.00160 & 0.01269 \\ -0.00102 & 0.60517 & -0.79610 & 1.76416 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} cos(\theta_k) & -sin(\theta_k) & 0 & \mathbf{r}cos(\theta_k) \\ sin(\theta_k) & cos(\theta_k) & 0 & \mathbf{r}sin(\theta_k) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{P} \\ \mathbf{0} & 1 \end{bmatrix} \quad (5)$$

Because the map is a 2-D map, we only need the **(x,y)** pairs in **P**.

After getting the positions of points in world frame, we need to convert them from physical unit meters to pixel cells and implement the function **bresenham2D** to obtain the occupied cells and free cells. We input the start point and end point of lidar ray and the function will return us a list of coordinates. The last pair of (x,y) is the occupied cells and the rest are the free cells.

Due to the accuracy of the lidar sensor, we need to implement Probabilistic Occupancy Grid Mapping to get a more accurate map. The occupancy grid m is unknown and needs to be estimated given the robot trajectory $\mathbf{x}_{0:t}$ and a sequence of observations $\mathbf{z}_{0:t}$. Since the map is unknown and the measurements are uncertain, maintain a $p(\mathbf{m} \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t})$ over time. Occupancy grid mapping algorithms usually assume that the cell values are independent conditioned on the robot trajectory:

$$p(\mathbf{m} \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) = \prod_{i=1}^{n} p(m_i \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \, (6)$$

Model the map cells mi as independent Bernoulli random variables:

$$m_i = \begin{cases} +1 \text{ (Occupied)} & \text{with prob. } \gamma_{i,t} := p(m_i = 1 \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \\ -1 \text{ (Free)} & \text{with prob. } 1 - \gamma_{i,t} \end{cases} \quad (7)$$

Odds ratio of the Bernoulli random variable mi updated via Bayes rule:

$$o(m_i \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) := \frac{p(m_i = 1 \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t})}{p(m_i = -1 \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t})} = \frac{\gamma_{i,t}}{1 - \gamma_{i,t}}$$

$$= \underbrace{\frac{p_h(\mathbf{z}_t \mid \mathbf{m}_i = 1, \mathbf{x}_t)}{p_h(\mathbf{z}_t \mid \mathbf{m}_i = -1, \mathbf{x}_t)}}_{g_h(\mathbf{z}_t \mid \mathbf{m}_i, \mathbf{x}_t)} \underbrace{\frac{\gamma_{i,t-1}}{1 - \gamma_{i,t-1}}}_{o(\mathbf{m}_i \mid \mathbf{z}_{0:t-1}, \mathbf{x}_{0:t-1})} \quad (8)$$

where $g_h(\mathbf{z}_t \mid \mathbf{m}_i, \mathbf{x}_t)$ is the observation model odds ratio. Using Bayes rule again, we can simplify the observation odds ratio:

$$g_h(\mathbf{z}_t \mid \mathbf{m}_i, \mathbf{x}_t) = \underbrace{\frac{p(\mathbf{m}_i = 1 \mid \mathbf{z}_t, \mathbf{x}_t)}{p(\mathbf{m}_i = -1 \mid \mathbf{z}_t, \mathbf{x}_t)}}_{\text{inverse observation model odds ratio}} \underbrace{\frac{p(\mathbf{m}_i = -1)}{p(\mathbf{m}_i = 1)}}_{\substack{\text{odds ratio} \\ \text{map prior}}} (9)$$

The inverse observation model odds ratio specifies how much we trust the observations. If the accuracy of the lidar sensor is 80% and the ratio $=\frac{80\%}{20\%} = 4$. The second term $o(m_i) = \frac{p(\mathbf{m}_i=1)}{p(\mathbf{m}_i=1)}$ is just a prior occupancy odds ratio.Then we take a log on both sides of the observation model odds ratio to convert the products to sums. And write in the form of log-odds of the Bernoulli random variable m_i :

$$\lambda_{i,t} := \lambda(m_i \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) := log \; o(m_i \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}) \, (10)$$

Estimating the pmf of $\mathbf{m}_i$ conditioned on $\mathbf{z}_{0:t}$ and $\mathbf{x}_{0:t}$ is equivalent to accumulating the log-odds ratio$_{i,t}$ of the inverse measurement model:

$$\lambda_{i,t} = \lambda_{i,t-1} + (\Delta\lambda_{i,t} - \lambda_{i,0})(11)$$

The initial map is uniform, so $\lambda_{i,0} = log1 = 0$. For a 80% accuracy lidar sensor:

$$\Delta\lambda_{i,t} = log \, \frac{p(m_i = 1 \mid z_t, x_t)}{p(m_i = -1 \mid z_t, x_t)}$$

$$= \begin{cases} +log4 & \text{if } z_t \text{ indicates } m_i \text{ is occupied} \\ -log4 & \text{if } z_t \text{ indicates } m_i \text{ is free} \end{cases} \quad (12)$$

For each observed cell i, decrease the log-odds if it was observed free or increase the log-odds if the cell was observed occupied:

$$\lambda_{i,t+1} = \lambda_{i,t} \pm log\, 4 \quad (13)$$

Prediction: We use the encoders and the FOG data to compute the instantaneous linear and angular velocities vt and t and estimate the robot trajectory via the differential drive motion model. What is also important to be noted is that we need to convert the pose from censer frame to the world frame:

$$_W\mathrm{T}_{\mathbf{V}_0} =_W \mathrm{T}_{\mathbf{F}_0} \cdot_{F_0} \mathrm{T}_{\mathbf{V}_0} =_W \mathrm{T}_{\mathbf{F_0}} \cdot_{V_0} \mathrm{T}_{\mathbf{F_0}}^{-1}$$

$$= \begin{bmatrix} cos(\omega_0) & -sin(\omega_0) & 0 & \mathbf{v}_0 \\ sin(\omega_0) & cos(\omega_0) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -0.335 \\ 0 & 1 & 0 & -0.035 \\ 0 & 0 & 1 & 0.78 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \quad (14)$$

Where $_W\mathrm{T}_{\mathbf{V}_0}^{(k)}$ is the pose of vehicle to the world frame on initial timestamp. The next pose of the vehicle to the world frame can be calculated as follow:

$$_W\mathrm{T}_{\mathbf{V}_n} =_W \mathrm{T}_{\mathbf{V}_{n-1}} \,_{n-1}\mathrm{T}_{\mathbf{n}} = \begin{bmatrix} cos(\omega_{n-1}) & -sin(\omega_{n-1}) & 0 & \mathbf{v}_{n-1} \\ sin(\omega_{n-1}) & cos(\omega_{n-1}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot$$

$$\begin{bmatrix} 1 & 0 & 0 & -0.335 \\ 0 & 1 & 0 & -0.035 \\ 0 & 0 & 1 & 0.78 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} cos(\omega_n) & -sin(\omega_n) & 0 & \mathbf{v}_n \\ sin(\omega_n) & cos(\omega_n) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

By iteration, we can obtain $_W\mathrm{T}_{\mathbf{V}_n}$ on any timestamp.
The robot trajectory coordinates (x,y) pair can be calculated as follow:

$$(x, y) = \begin{bmatrix} \cdot & \cdot & \cdot & x \\ \cdot & \cdot & \cdot & y \\ \cdot & \cdot & \cdot & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (16)$$

Update: After we having the predicted trajectory with no noise and only a single particle, include an update step that uses scan-grid correlation to correct the robot pose.
As mentioned before, the data we obtained from sensors contains noises. In update part, we need to taken the noise ϵ_t into consideration. One of the methods to reduce the effort of the noise is the Particle Filter.The particle is a hypothesis that the value of **x** is $\mu^{(k)}$ with probability $\alpha^{(k)}$ and the particle filter is a histogram filter which allows its grid centers to move around and adaptively concentrate in areas of the state space that are more likely to contain the true state.
The particle filter uses a set of hypotheses (particles) with locations $\left\{\mu^{(k)}\right\}_k$ and weights $\left\{\alpha^{(k)}\right\}_k$ to represent the pdfs $p_{t|t}$ and $p_{t|t+1}$:

$$p_{t|t}\left(\mathbf{x}_t\right) = \sum_{k=1}^{N_{t|t}} \alpha_{t|t}^{(k)} \delta\left(\mathbf{x}_t - \mu_{t|t}^{(k)}\right) (17)$$

$$p_{t+1|t}\left(\mathbf{x}_{t+1}\right) = \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(\mathbf{x}_{t+1} - \boldsymbol{\mu}_{t+1|t}^{(k)}\right) (18)$$

In this project, we can implement a 2-D Gaussian motion noise in the linear and angular velocity input. With the Gaussian motion noise, the robot trajectory becomes the locations with weights. Let $\mu_{t+1|t}^{(k)} \sim p_f\left(\cdot \mid \mu_{t|t}^{(k)}, u_t\right)$ and $\alpha_{t+1|t}^{(k)} = \alpha_{t|t}^{(k)}$ so that :

$$p_{t+1|t}(\mathbf{x}) \approx \sum_{k=1}^{N} \alpha_{t+1|t}^{(k)} \delta\left(\mathbf{x} - \mu_{t+1|t}^{(k)}\right) (19)$$

Plug the particle representation of $p_{t+1|t}$ in the Bayes filter update step:

$$p_{t+1|t+1}(\mathbf{x}) = \frac{p_h\left(\mathbf{z}_{t+1} \mid \mathbf{x}\right) \sum_{k=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(k)} \delta\left(\mathbf{x} - \mu_{t+1|t}^{(k)}\right)}{\int p_h\left(\mathbf{z}_{t+1} \mid \mathbf{s}\right) \sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} \delta\left(\mathbf{s} - \mu_{t+1|t}^{(j)}\right) d\mathbf{s}}$$
$$= \sum_{k=1}^{N_{t+1|t}} \underbrace{\left[\frac{\alpha_{t+1|t}^{(k)} p_h\left(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(k)}\right)}{\sum_{j=1}^{N_{t+1|t}} \alpha_{t+1|t}^{(j)} p_h\left(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(j)}\right)}\right]}_{\alpha_{t+1|t+1}^{(k)}} \delta(\mathbf{x} - \underbrace{\mu_{t+1|t}^{(k)}}_{\mu_{t+1|t+1}^{(k)}})$$

(20)
So in update step, the particle poses remain unchanged but the weights are scaled by the observation model:

$$\boldsymbol{\mu}_{t+1|t+1}^{(k)} = \boldsymbol{\mu}_{t+1|t}^{(k)}$$

$$\alpha_{t+1|t+1}^{(k)} \propto p_h\left(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(k)}, \mathbf{m}\right) \alpha_{t+1|t}^{(k)} (21)$$

The $p_h\left(\mathbf{z}_{t+1} \mid \boldsymbol{\mu}_{t+1|t}^{(k)}, \mathbf{m}\right)$ is the Laser Correlation Model: a model for a laser scan **z** obtained from sensor pose **x** in occupancy grid m based on correlation between **z** and **m**.The laser correlation model sets the likelihood of a laser scan **z** proportional to the correlation between the scan's world-frame projection $\mathbf{y} = r(\mathbf{z}, \mathbf{x})$ via the robot pose **x** and the occupancy grid **m**:

$$p_h(\mathbf{z} \mid \mathbf{x}, \mathbf{m}) \propto \operatorname{corr}(r(\mathbf{z}, \mathbf{x}), \mathbf{m}) (22)$$

In this project, we used Lidar Occupancy Grid Mapping to obtain the map m. So we can use scan-grid correlation to generate the weights of the particles:
Transform the scan **z** from the laser frame to the world frame using the robot pose **x** (transformation from the body frame

to the world frame).
Find all grid coordinates y that correspond to the scan, i.e., y is a vector of grid cell indices i which are visited by the laser scan rays.
Let **y** = r(**z**, **x**) be the transformation from a lidar scan z to grid cell indices y. Definite a similarity function corr(r(**z**, **x**), **m**) between the transformed and discredited scan **y** and the occupancy grid **m**:

$$\operatorname{corr}(\mathbf{y}, \mathbf{m}) = \sum_i 1\{y_i = m_i\}$$

We choose the particle with largest weight $\alpha_{t|t}^{(k)}$ as the new motion model of the next timestamp and project the laser scan zt to the world frame and update the map log-odds. With the map log-odds we can compute new correlation and update the trajectory. The iteration begins.

3.2. Texture Problem

In this part,at first we need to use the function **compute_stereo** to compute a appropriate disparity measurement. Together with the Stereo Camera Model we can obtain the the depth **z** of each RGB pixel:

$$\begin{bmatrix} u_L \\ v_L \\ u_R \\ v_R \end{bmatrix} = \underbrace{\begin{bmatrix} fs_u & 0 & c_u & 0 \\ 0 & fs_v & c_v & 0 \\ fs_u & 0 & c_u & -f_{s_u}b \\ 0 & f_{s_v} & c_v & 0 \end{bmatrix}}_{M} \frac{1}{z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Because of the stereo setup, two rows of M are identical. The vertical coordinates of the two pixel observations are always the same because the epipolar lines in the stereo configuration are horizontal. Drop the v_R equation replace the u_R equation with a disparity measurement $d = u_L - u_R = \frac{1}{z} fs_u b$, we get:

$$\begin{bmatrix} u_L \\ v_L \\ d \end{bmatrix} = \begin{bmatrix} f_{s_u} & 0 & c_u & 0 \\ 0 & fS_v & c_v & 0 \\ 0 & 0 & 0 & f_{s_u}b \end{bmatrix} \frac{1}{z} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}_oR_r R^\top(\mathbf{m} - \mathbf{p})$$

m is the point $\in R^3$ in the world frame observed by a stereo camera.To texture the occupancy grid map, we need to transform the LiDAR points from the LiDAR frame to the stereo camera frame.

$${}_S\mathrm{m}_P = {}_V\mathrm{T}_S^{-1}\, {}_V\mathrm{T}_L\, {}_L\mathrm{m}_P$$

After transformation of points, we can obtain the RGB information of the occupied cells and free cells on stereo camera timestamp. Then we can transform the RGB values to the world frame:

$${}_W\mathrm{m}_{P_n} = {}_W\mathrm{T}_{V_n}\, {}_{V_n}\mathrm{T}_{S_n}\, {}_{S_n}\mathrm{m}_{P_n}$$

Where ${}_W\mathrm{T}_{V_n}$ can be obtained in formula (15).

4. RESULTS

The pictures shown below are the images of the trajectory **Fig.1** and occupancy grid map **Fig.2** over time for the SLAM problem. To be honest, I did not get a good result for this project. Because my code is very long and complex. Some functions I used in my code really kill time, like the **np.append** function in a for-loop structure over a hundred thousand times iteration. As a result, I had to cut down the particle number and update my map for every 10 timestamps. I do not know how to use the **mapCorrelation** function provided in **pr2_utils.py**. So I wrote a correlation computing function by myself. But unfortunately, it did not work. The correlation declines with the iteration increasing. I think my pose matrix which convert points in lidar frame to the world frame is correct. Something must be wrong in my programming process.
And for texture problem, I really did not have the time to write the program. Because I spent too much time on modifying the SLAM program to kill bugs.
In conclusion, the results of project 2 is not good.

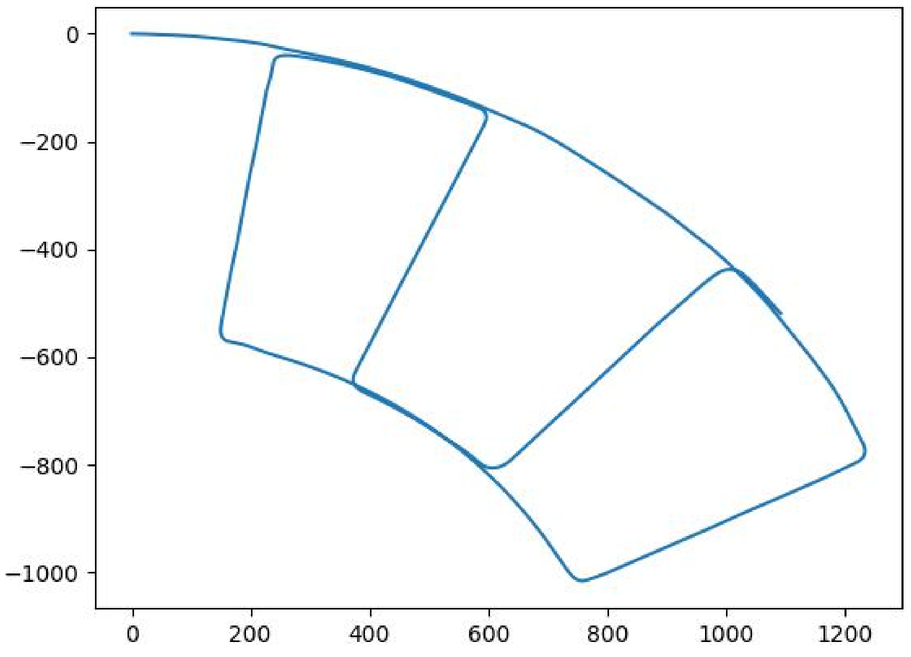

Fig.1 The trajectory of the robot

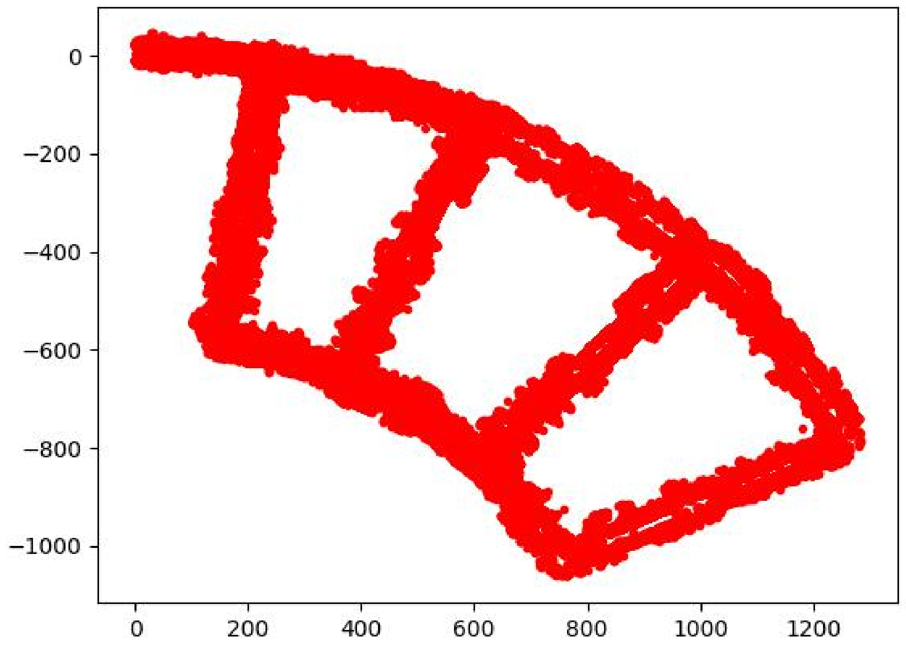

Fig.2 The occupancy grid map