ECE276A Project 2: Particle Filter SLAM

Wilson Liao
Department of Electrical and Computer Engineering
University of California, San Diego
w4liao@ucsd.edu

I. Introduction

SLAM (Simultaneous Localization and Mapping) is a technique used in robotics and computer vision to create a map of an unknown environment while at the same time keeping track of the robot's position within that environment. SLAM is a challenging problem because it involves estimating both the position of the robot and the location of landmarks in the environment simultaneously, and dealing with the uncertainty that arises from sensor noise and imperfect knowledge of the environment. It's applications in robotics include autonomous navigation, mapping of unknown environments, and augmented reality.

In this project, we propose a solution for SLAM based on particle filter and occupancy grid. We utilize encoder, LiDAR and IMU data gained from a differential-drive robot with motion model and observation model to perform mapping and localization simultaneously.

We also extend this to texture mapping, which aligns the color pixels from RGBD sensor onto the occupancy grid.

II. Problem Formulation

A. Particle Filter Slam

1) Simultaneous localization and mapping:

The SLAM problem is a combination of 2 problems, which are mapping and localization. Given robot's trajectory $x_{0:T}$ and sensor measurements $z_{0:T}$ and observation model h, the mapping problem can be written in this way:

$$\min_{m} \sum_{t=0}^{T} ||z_t - h(x_t, m)||_2^2 \tag{1}$$

where m is the map.

Given the map m, sensor measurements $z_{0:T}$, observation model h, control input $u_{0:T-1}$ and motion model f, we can localize the robot, i.e. get the robot's trajectory, by defining the localization problem as follows:

$$\min_{x_{0:T}} \sum_{t=0}^{T} ||z_t - h(x_t, m)||_2^2 + \sum_{t=0}^{T-1} ||x_{t+1} - f(x_t, u_t)||_2^2 \tag{2}$$

where $x_{0:T}$ is the trajectory.

Combining (1) and (2), we can formulate an optimization problem to estimate the orientation trajectory $x_{1:T}$ and map m. The following is the objective function of the slam problem:

$$\min_{x_{1:T}, m} \sum_{t=1}^{T} ||z_t - h(x_t, m)||_2^2 + \sum_{t=0}^{T-1} ||x_{t+1} - f(x_t, u_t)||_2^2 \tag{3}$$

2) Bayes Filter:

Bayes Filter is a probabilistic technique for estimating the state x_t of a dynamical system by combining evidence from control inputs u_t and observations z_t using Markov assumptions and Bayes rule. The Bayes filter keeps track of predicted pdf and updated pdf.

- Prediction Step: given a prior pdf $p_{t|t}$ of x_t and control input u_t, we use the motion model p_f to compute the predicted pdf $p_{t+1|t}$ of x_t:

$$p_{t+1|t}(x) = \int p_f(x|s, u_t) p_{t|t}(s) ds \tag{4}$$

- Update Step: given a predicted pdf $p_{t+1|t}$ of $x_t + 1$ and measurement $z_t + 1$, we use the observation model p_h to obtain the updated pdf $p_{t+1|t+1}$ of x_t:

$$p_{t+1|t+1}(x) = \frac{p_h(z_{t+1}|x) p_{t+1|t}(x)}{\int p_h(z_{t+1}|s) p_{t+1|t}(s) ds} \tag{5}$$

3) Occupancy Grid Mapping:

Occupancy grid map m is unknown and needs to be estimated given the robot trajectory $x_{0:t}$ and a sequence of observations $z_{0:t}$. Besides, the measurements are also uncertain, we maintain a probability mass function over time:

$$p(m|z_{0:t}, x_{0:t}) \tag{6}$$

Most occupancy grid mapping algorithms assume that the cell values are independent conditioned on the robot trajectory:

$$p(m|z_{0:t}, x_{0:t}) = \prod_{i=1}^{n} p(m_i|z_{0:t}, x_{0:t}) \tag{7}$$

4) Texture Mapping:

Given a occupancy gird map m and RGBD images, the goal is to form a vector $m_c \in R^{n \times 3}$,where each entry is a RGB floor color corresponding to the cell in m.

III. Technical Approach

A. Particle Filter Slam

1) Frame Transformation:

We first transform the received LiDAR scan data from range data into x-y coordinates for further transformation. Then we take the points and transform from LiDAR frame

to world frame in order to construct our occupancy map. The following shows the overall transformation:

$$_WT_L = {}_WT_B {}_BT_L \tag{8}$$

where W, B and L represent world, body, and LiDAR frame respectively. Moreover, the transformation matrix can be written in the following way:

$$_WT_B = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & x \\ sin(\theta) & cos(\theta) & 0 & y \\ 0 & 0 & 1 & 0.127 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

where x and y are the current robot position in the world frame, and θ is the yaw of the robot.

$$_BT_L = \begin{bmatrix} 1 & 0 & 0 & 0.150915 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.51435 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

Once we get the 2 transformation matrix, we can transform LiDAR scan data from LiDAR frame to world frame:

$$\begin{bmatrix} x_W \\ y_W \\ z_W \\ 1 \end{bmatrix} = {}_WT_B {}_BT_L \begin{bmatrix} x_L \\ y_L \\ 0 \\ 1 \end{bmatrix} \tag{11}$$

2) Mapping:

After we get the LiDAR scan in world frame, we first filter out points that are too close (0.1m) or too far (20m) from the robot. Then we use the Bresenham algorithm to get the grids that are passed by the line connected by the robot position adn LiDAR points. These grids are viewed as free cells while the grids where the LiDAR points set are viewed as occupied cells.

The grid map is initialized with zeros, and is updated by adding $4log\gamma$ into occupied cells and minus $log\gamma$ into free cells, where γ is set as $\frac{0.9}{0.1}$. The values of the log map are limited to ± 100.

3) Update particles:

We use particle filter to solve the SLAM problem. We first initialize N particles with random poses and uniform weights $(1/N)$. For every particle $\mu^k_{t|t}$, $k = 1...N$, we compute their next state as follows:

$$\mu^k_{t+1|t} = f(\mu^k_{t|t}, \mu_t + \sigma_t) \tag{12}$$

where f is the motion model, $\mu_t = (x_t, y_y, \theta_t)$ is the relative odometry input, and $\sigma_t \sim N(0, \sigma)$ is a 2D Gaussian motion noise.

To elaborate more, the poses of all particles are updated with the motion model in this way, which is also the motion model f of the particles:

$$\begin{bmatrix} x^i_t \\ y^i_t \\ \theta^i_t \end{bmatrix} = \begin{bmatrix} x^i_{t-1} \\ y^i_{t-1} \\ \theta^i_{t-1} \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} + \begin{bmatrix} N(0, \sigma_x) \\ N(0, \sigma_y) \\ N(0, \sigma_\theta) \end{bmatrix} \tag{13}$$

where x^i_t, y^i_t and θ^i_t are the pose for each particle i at t^{th} iteration.

The pose of each particle is updated by its previous pose and the difference in odometry and motion noise. The odometry is obtained from combining encoder and imu data as below:

$$\begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} = \tau_t \begin{bmatrix} v_t cos(\theta_t) \\ v_t sin(\theta_t) \\ \omega \end{bmatrix} \tag{14}$$

where v_t and ω are obtained from encoder and imu data respectively:

$$v_t = \frac{V_L + V_R}{2} \tag{15}$$

$$\omega = yaw(\text{angular velocity}) \tag{16}$$

$$\tag{17}$$

where

$$V_R = (FR + RR)/2 * 0.0022 * 40 \tag{18}$$

$$V_L = (FL + RL)/2 * 0.0022 * 40 \tag{19}$$

where encoder counts [FR, FL, RR, RL] corresponding to the front-right, front-left, rear-right, and rear-left wheels

4) Measurements update:

After we get the new poses of the particles from the motion update, we need to check whether the particle poses are align with the current LiDAR scan. To elaborate more, we need to transform the scan z_{t+1} to the world frame using $\mu^k_{t|t}$ (particle pose) for $k = 1...N$ and find all cells y^k_{t+1} in the grid corresponding to the scan. The weights of the particles are updated and resampled if necessary.

- Weight update: The weights of the particles are set according to how much the LiDAR scan aligned with the pose of each particle. The particle whose map corresponds highly to the previous log map is given more weight (correlation value). Thus, we can calculate the weights by summing up over the LiDAR scan hits or occupied cells. Then we multiply the previous weight vector with the correlation value of each particle, and then normalize it with softmax. The following shows the process:

$$p_h(z_{t+1}|\mu^k_{t|t,m}) \propto exp(corr(y^k_{t+1}, m)) \tag{20}$$

$$corr(y, m) := \sum_i 1(m_i = y_i) \tag{21}$$

- Resampling: We will perform resampling if the below condition is satisfied:

$$N_{eff} = \frac{1}{\sum_i^N w_i^2} < \gamma N \tag{22}$$

where N is the number of particles, and w_i is the weight of particle i, and we set ratio γ to 0.1.

We use Stratified Resampling algorithm for resampling, the steps are simplified as below: (1) Add the particle weights along the circumference of a circle. (2) Divide the circle into N squal pieces and sample a uniform distribution in each piece. (3) Select the particles corresponding to the uniform distribution samples.

B. Texture map

We first obtain the depth and the pixel location (rgbi, rgbj) of the associated RGB color by the following conversion:

$$dd = (-0.00304d + 3.31) \tag{23}$$

$$depth = \frac{1.03}{dd} \tag{24}$$

$$rgbi = \frac{(526.37i + (-4.5 * 1750.46)dd + 19276)}{585.051} \tag{25}$$

$$rgbi = \frac{(526.37j + 16662)}{585.051} \tag{26}$$

Once we get the rgbi, rgbj pixel coordinates, we transfrom them to Optical frame coordinates with the following:

$$\begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} = K^{-1} \begin{bmatrix} rgbi \\ rgbj \\ 1 \end{bmatrix} * depth \tag{27}$$

$$K = \begin{bmatrix} 585.05 & 0 & 242.94 \\ 0 & 585.05 & 315.84 \\ 0 & 0 & 1 \end{bmatrix} \tag{28}$$

where K is the calibration matrix of the depth camera.

Then we transform coordinates from Optical frame to world frame with the following:

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = {}_WT_B\,{}_BT_{co}T_c^{-1} \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} \tag{29}$$

where ${}_oT_c$ transform camera frame to optical frame, ${}_BT_c$ transform camera frame to body frame, ${}_WT_B$ transform body frame to world frame.

Now, we can get the indices of these points on the occupancy grid map by converting the coordinates in the frame to pixel frame.

Finally, we can insert the RGB values to the map over time.

IV. Results

A. Particle Filter Slam

The following are the trajectories and occupancy grid maps over time constructed by SLAM using data from dataset 20 and 21, where the blue line is the trajectory gained by dead-reckoning while the red dots are the best particle pose in each iteration. Besides, the black pixels are free and gray pixels are occupied.

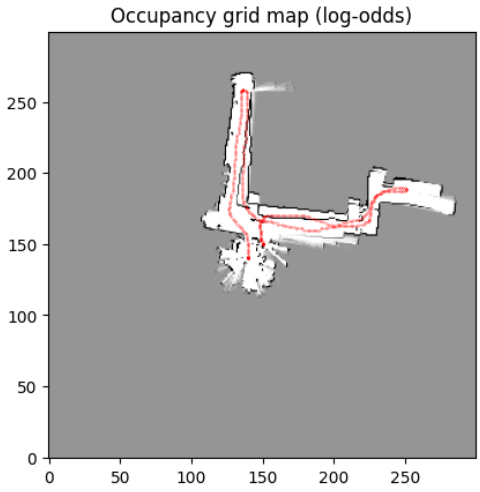

Fig. 1. dataset 20

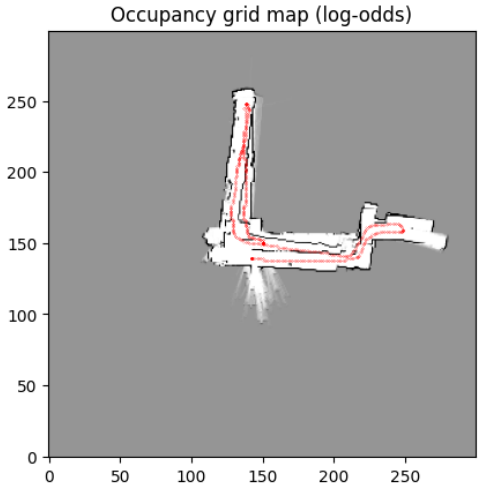

Fig. 2. dataset 21

From the result above, we can see that the trajectory gained from particle filter have a little shift from the ones gained from dead-reckoning, which we viewed it as an improve from using only dead-reckoning. Besides, the occupancy grids we gained from using particle filter slam look good and reasonable, where the free grids are all presented around our trajectory.

B. Texture map

I do not have enough time to work on texture map, but I know I can follow the steps mentioned above to work on it.