ECE 276A: Sensing and Estimation in Robotics Project 2

Lin-Hsi Tsao (PID: A59015911)
Department of Electrical and Computer Engineering
University of California, San Diego
ltsao@ucsd.edu

Acknowledge: Wilson Liao (w4liao@ucsd.edu), Reference on Website

I. Introduction

In this project, we are required to implement two problems: which is Particle-filter SLAM and Texture map.

A. Particle-filter SLAM

Particle-filter SLAM (Simultaneous Localization and Mapping) is a type of algorithm used in robotics and autonomous systems to estimate the position of a robot in an unknown environment while simultaneously creating a map of the environment. It is a type of Monte Carlo localization algorithm that uses a set of particles to represent the possible locations of the robot and its surroundings.

In this algorithm, the robot's movement and sensor data are used to update the probability distribution of the robot's location, which is represented by the particles. Each particle represents a hypothesis of the robot's location and the location of the landmarks in the environment.

The particle filter uses a set of important weights to give more weight to particles that are more likely to represent the true state of the robot. As the robot moves and collects more sensor data, the particle filter updates the particles and their weights to improve the accuracy of the robot's location estimate and the map of the environment.

B. Texture map

In robotics, a texture map is a type of sensor data that is used to represent the visual texture of an object or surface. It can be obtained from a camera or other vision sensors and is often used in robot perception and recognition tasks.

Texture maps can be used to provide robots with information about the surface properties of an object, such as its color, pattern, and texture. This information can be used by the robot to identify and classify objects based on their visual appearance.

II. Problem Formulation

A. Particle-filter SLAM

1) Mapping: Assume the robot starts with identity pose and use the first LiDAR scan to create an occupancy grid map. The occupancy grid map is unknown and needs to be estimated given the trajectory $x_{0:t}$ and a sequence of observations $z_{0:t}$.

2) Prediction: We use linear velocity v_t obtained from the encoders and yaw rate ω_t obtained from the IMU to predict the robot motion via the differential-drive motion model. We implement dead-reckoning first to verify the prediction step working correctly. Then, we implement the particle-filter prediction step with N particles at the initial identity robot pose and add noise to the motion model to obtain a predicted pose μ_i for each particle i = 1, . . . , N.

3) Update: When the prediction-only filter works, we include an update step that uses scan-grid correlation to correct the robot pose. We use the update step with only 3 - 4 particles at first to see if the weight updates make sense. Below are two steps for the update:

- Convert the LiDAR scan z_{t+1} to the world frame from each particle's pose $\mu_{t+1|t+1}[k]$ to compute map correlation and update the particle weights.
- Choose the particle with the highest weight $\alpha_{t+1|t+1}[k]$, project the LiDAR scan z_{t+1} to the world frame and update t

B. Texture Map

By an occupancy grid map m, we would like to form a vector $m_c \in R^{n3}$. Each element of the vector represents an RGB floor color that corresponds to a cell in the occupancy grid map.

III. Technical Approach

A. Particle-filter SLAM

1) Mapping: Below are my steps to handle the problem.

- Step 1: Removed the measurements outside the range (0.1, 30) from the lidar scan data.
- Step 2: Since the lidar scan from $-135°$ to $135°$, we can convert the depth point coordinate to the x-y plane by

$$x = depth * cos(rad) \tag{1}$$

$$y = depth * sin(rad) \tag{2}$$

  where rad denotes the rad on the given time T.
- Step 3: We converted (x,y) coordinates in the lidar frame to the world frame by

$$_W T_L =_W T_B {}_B T_L \tag{3}$$

where ${}_WT_L$ denotes the transformation matrix from the lidar frame to the world frame, ${}_WT_B$ denotes the transformation matrix from the body frame to the world frame, and ${}_BT_L$ denotes the transformation matrix from the lidar frame to the body frame.

Based on the document, we could get the matrix and shows below:

$$ {}_WT_B = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & X \\ sin(\theta) & cos(\theta) & 0 & Y \\ 0 & 0 & 1 & 0.127 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

$$ {}_BT_L = \begin{bmatrix} 1 & 0 & 0 & 0.150915 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.51435 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5} $$

where θ denotes the lidar orientation, while X and Y denote the coordinate in the body frame.

- Step 4: Finally, we convert the x-y plane coordinates to the occupancy grid map coordinate and plot it onto the map.

B. Prediction

Below are my steps to handle the problem.

- Step 1: Sync the lidar, imu, and encoder data. I fixed the lidar time stamp to get the corresponding imu and encoder data with the certain time.
- Step 2: Perform the robot motion. From the encoder data, we can get

$$ VR = (FR + RR)/2 * 0.0022 * 40 \tag{6} $$

$$ VL = (FL + RL)/2 * 0.0022 * 40 \tag{7} $$

$$ V = (VR + VL)/2 \tag{8} $$

Then, we could update the robot pose by:

$$ X = X_{prev} + TimeInterval * V * cos(\theta_{prev}) \tag{9} $$

$$ Y = Y_{prev} + TimeInterval * V * cos(\theta_{prev}) \tag{10} $$

$$ \theta = \theta_{prev} + TimeInterval * \omega \tag{11} $$

C. Update

For every particle $\mu^k_{t|t}$, k = 1 . . . N compute: $\mu^k_{t+1|t}$ = f($\mu^k_{t|t}$, $\mu_t + \sigma_t$), where f is the motion model, μ_t = (x_t, y_t, θ_t) is the relative odometry input, and σ_t denoting N (0, σ) is a 2-D Gaussian motion noise.

We could update the motion of all the particles according to the odometry data:

$$ x^i_t = x^i_{t-1} + dx + N(0, \sigma_x) \tag{12} $$

$$ y^i_t = y^i_{t-1} + dy + N(0, \sigma_y) \tag{13} $$

$$ \theta^i_t = \theta^i_{t-1} + d\theta + N(0, \sigma_\theta) \tag{14} $$

IV. Results

Below are the results of my occupancy map on the first scan, trajectory & theta on the dead-reckoning, and the final results of the particle filter. From the particle filter, we could easily see that the trace is like the predicted one.

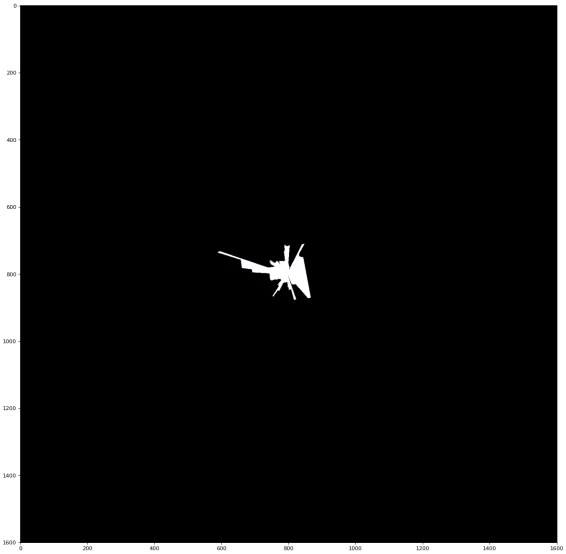

Fig. 1. Occupancy Map first scan on dataset 20

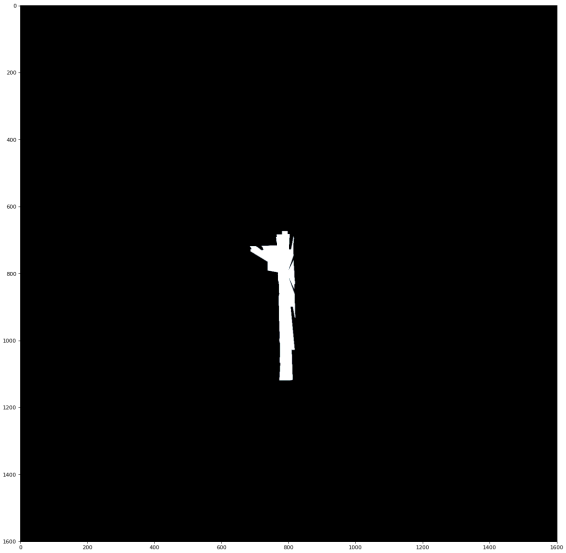

Fig. 2. Occupancy Map first scan on dataset 21

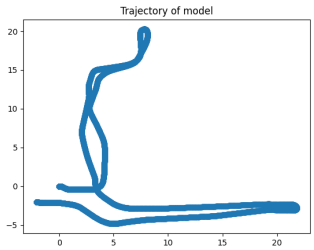

Fig. 3. Trajectory on dataset 20: dead-reckoning

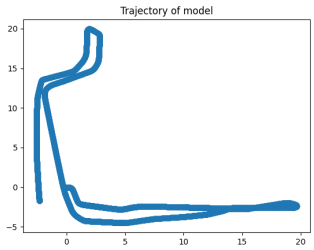

Fig. 4. Trajectory on dataset 21: dead-reckoning

Fig. 5. Theta on dataset 21: dead-reckoning

Fig. 6. Theta on dataset 21: dead-reckoning

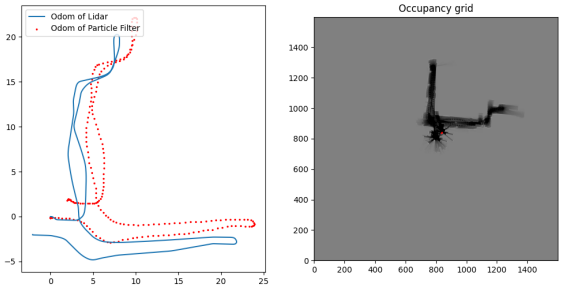

Fig. 7. Particle filter on dataset 20

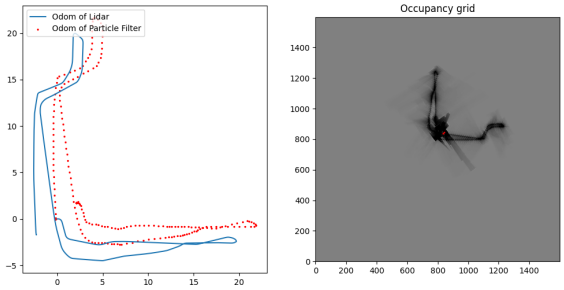

Fig. 8. Particle filter on dataset 21