Simultaneous Localization and Mapping with Texture: Project Report

Pengluo Wang
Department of Electrical Computer Engineering
University of California, San Diego
pew067@eng.ucsd.edu

Abstract—This paper presented an overview for Simultaneous Localization and Mapping (SLAM) algorithm based on particle filters using odometry (encoder), inertial (IMU), 2-D laser (Hokuyo) range, and RGBD (Kinect) measurements from a differential-drive robot. Results of an implementation based on different dataset are shown to prove the effectiveness of the algorithm. The project has been uploaded to https://github.com/PenroseWang/SLAM-Based-on-Particle-Filters.

Index Terms—SLAM, Particle Filters, Markov Chain

I. Introduction

The problem of simultaneous localization and mapping (aka. SLAM) has attracted immense attention in the mobile robotics literature and becomes quite popular recently. In robotic mapping and navigation, SLAM is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it [1]–[3]. SLAM appears to be a chicken-and-egg problem in the sense that map of the environment is needed to localize the robot, but a good estimation of localization or trajectory of the robot can only be obtained given an accurate information of the environment.

SLAM algorithms are tailored to the available resources, hence not aimed at perfection, but at operational compliance. Published approaches are employed in self-driving cars, unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers, newer domestic robots and even inside the human body [4]. Here in this report we focus on SLAM algorithm using 3 sensors, encoder for odometry, IMU for inertial, and LIDAR for 2-D laser measurements. Kinect is used for create texture mapping for floors.

Structure of the report is listed as follows:

- Chapter II briefly states the problem to be solved in mathematical terms.
- Chapter III clarifies the approach to SLAM algorithm.
- Chapter IV presents results of the training/test dataset, and gives a brief analysis of the results.

II. Problem Formulation

A. Problem Overview

The architecture of a SLAM system includes two main components: the front-end and the back-end. The front-end abstracts sensor data into models that are amenable for estimation, while the back-end performs inference on the abstracted data produced by the front-end [3]. In this chapter we will mainly introduce the mathematics behind data inference by back-end components.

B. Bayes Filter

SLAM is a state estimation problem, and can be best described as a probabilistic Markov chain. Denote the robot's sate at time t as x_t, control input as u_t, observation as z_t, and environment state as m_t, and let motion model and observation model be:

- Motion:

$$x_{t+1} = f(x_t, u_t, w_t) \sim p_f(\cdot|x_t, u_t) \tag{1}$$

- Observation:

$$z_t = h(x_t, m_t, v_t) \sim p_h(\cdot|x_t, m_t), \tag{2}$$

where w_t and v_t are motion noise and observation noise.

From Markov assumption the joint pdf of the robot state $x_{0:T}$, map state $m_{0:T}$, observations $z_{0:T}$, and controls $u_{0:T-1}$ can be represented as:

$$\begin{aligned} &p(x_{0:T}, m_{0:T}, z_{0:T}, u_{0:T-1}) = \\ &p_{0|0}(x_0)\prod_{t=0}^{T} p_h(z_t|x_t, m_t)\prod_{t=1}^{T} p_f(x_t|x_{t-1}, u_{t-1}) \end{aligned} \tag{3}$$

Using Recursive Bayesian estimation (aka. Bayes filter) leads to the update and predict steps defined as follows:

- Prediction:

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

- Update:

$$p_{t+1|t+1}(x) = \frac{1}{\eta}p_h(z_{t+1}|x, m_t)p_{t+1|t}(x), \tag{5}$$

where $\frac{1}{\eta}$ is a normalization factor.

C. Particle Filter

Particle filters have been used to represent pdfs $p_{t|t}$ and $p_{t+1|t}$ by approximating them as a mixture of delta function (which is called particles). Because of the finite set of particles, a situation in which most of the updated particle weights become close to zero may happen, which is called particle depletion. Particle filter uses a procedure

called resampling to rectify this phenomenon. Stratified resampling is optimal in terms of variance, thus it has been used in the project. The algorithm of stratified resampling is described as follows:

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

D. Motion and Observation Model

A motion model describes the density function $p_f(\cdot|x, u)$ of a new robot state after motion for a given state x with control input u. Here we use differential drive model to derive the density function.

Suppose the object is moving in the 2D map. Then the state can be represented as $s = (p, \theta) \in SE(2)$, where $p = (x, y) \in \mathbb{R}^2$ is the position and $\theta \in (-\pi, \pi]$ is the orientation (yaw angle). And control $u = (v, \omega)$, where $v \in \mathbb{R}$ is the linear velocity and $\omega \in \mathbb{R}$ is the rotational velocity (yaw rate). Then it's easy to obtain the motion model as

$$\begin{aligned} s_{t+1} =& f(s_t, u_t, w_t) \\ =& s_t + \tau \begin{pmatrix} v_t \text{sinc}(\frac{\omega_t \tau}{2}) \cos(\theta_t + \frac{\omega_t \tau}{2}) + w_{x_t} \\ v_t \text{sinc}(\frac{\omega_t \tau}{2}) \sin(\theta_t + \frac{\omega_t \tau}{2}) + w_{y_t} \\ \omega_t + w_{\omega_t} \end{pmatrix}, \end{aligned} \tag{6}$$

where τ is the time interval, and w_t is the noise. Encoder and IMU can be used to measure value of control $u = (v, \omega)$.

We use laser (LIDAR) correlation model for the observation model in the sense that we use occupancy grid for map representation. Laser correlation model calculates the correlation between a laser scan z obtained from sensor pose x and an occupancy map m with free ($m_i = 0$) and occupied ($m_i = 1$) cells. The procedure can be stated as follows:

1) Transform the LIDAR scan z to the world frame using x and find all points y in the grid that correspond to the scan.
2) Let observation model be proportional to the similarity $\text{corr}(y, m)$ between the transformed scan y and the grid map m.

The weights can be converted to probabilities via the softmax function:

$$p_h(z|x, m) = \frac{e^{\text{corr}(y,m)}}{\sum_v \text{corr}(v, m)} \propto e^{\text{corr}(y,m)} \tag{7}$$

E. Rotation and Projection

Since measurements of sensors are taken under different coordinate systems (body frame, world frame, etc.), they need to be transformed into the same world frame for processing. Considering using rigid body motion, a transformation can be easily represented as matrix multiplication under homogeneous coordinates. And the pose of a rigid body can be represented by its position and rotation under its world frame.

Suppose a rigid body frame B has position $p_{wb} \in \mathbb{R}^3$ and rotation $R_{wb} \in SO(3)$ under world frame W. Then the pose of the rigid body can be described by a matrix:

$$T_{wb} = \begin{bmatrix} R_{wb} & p_{wb} \\ 0 & 1 \end{bmatrix} \subset \mathbb{R}^{4\times 4}. \tag{8}$$

Given a point s_b in the rigid body frame, its world frame coordinates can be calculated as:

$$s_w = R_{wb} s_B + p_{wb}, \tag{9}$$

which is equivalent as

$$\begin{bmatrix} s_w \\ 1 \end{bmatrix} = T_{wb} \begin{bmatrix} s_b \\ 1 \end{bmatrix} \tag{10}$$

when using homogeneous coordinates.

Now consider representation under canonical pinhole camera projection. The extrinsic matrix is defined as

$$T_{oc} = \begin{bmatrix} R_{oc} R_{wc}^T & -R_{oc} R_{wc}^T p_{wc} \\ 0 & 1 \end{bmatrix}, \tag{11}$$

thus we have

$$\begin{bmatrix} X_o & Y_o & Z_o & 1 \end{bmatrix}^T = T_{oc}^T = \begin{bmatrix} X_w & Y_w & Z_w & 1 \end{bmatrix} \tag{12}$$

The projection and intrinsic matrix transform the coordinates in optical frame to image frame:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fs_u & fs_\theta & c_u \\ 0 & fs_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \frac{1}{Z_o} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_o \\ Y_o \\ Z_o \\ 1 \end{bmatrix}. \tag{13}$$

III. Technical Approach

A. Data Pre-processing

Notice that the robot have several sensors measured under different time stamp, so the first step is to perform data synchronization in order to obtain accurate measurements under same time. Notice that in the given data, measurements of encoder and LIDAR are taken in a similar frequency around 40 Hz. IMU measurements have a higher frequency and Kinect have a lower frequency compared with encoder and LIDAR. In the sense that IMU data is noisy and LIDAR is more important for constructing the map environment, LIDAR time stamp has been used as the standard time for all other measurements. Since in one interval of LIDAR time stamp, several IMU data will be measured, the mean of the measurements will be calculated and chosed as the input the current time stamp. After data synchronization, time interval for each time stamp, and update flag of each sensors will be created.

In order to reduce the calculation afterwards, data transformation is also performed at the same time.

Given encoder counts $[FR, FL, RR, RL]$ corresponding to the four wheels, the right wheels travel a distance of $d_r = (FR + RR)/20.0022$, while the left wheels travel a distance of $d_l = (FL + RL)/20.0022$. Thus the speed of robot v at current time stamp t can be represented as $v_t = (d_r+d_l)/2\tau$, where τ is the time interval. The variance of speed is also calculated during this process to estimate the noise adaptively. At each time stamp, the variance of the data within ± 3 time stamps will be measured as the noise of the current stamp.

For LIDAR measurements, scan points that are too close or too far will be removed. After obtaining the valid range r and angle θ , we can calculate the coordinates (x, y) of each LIDAR beam as

$$x = r\cos(\theta), \quad y = r\sin(\theta).$$

The IMU data is noisy, since it is collected from a moving robot with a lot of high frequency vibrations. Thus a low pass filter with order 10 and bandwith at 10 Hz is performed on the data to reject high frequency noise. Similar with encoder, the variance is measured for each time stamp to estimate noise adaptively.

For Kinect RGBD data, it's important to save the file path with current time stamp. No further processing is need.

B. Grid Map

A python class Map() is created to handle all functions and properties of a grid map. The basic properties of a grid map are listed as follows:

- Map size and resolution properties: specify x axis ranges from $[x_{min}, x_{max}]$, y axis ranges from $[y_{min}, y_{max}]$, and map resolution res.
- Grid map m: grid map is saved as a matrix, whose dimension is defined by x_{min}, x_{max}, y_{min}, y_{max}, and res. Each entry of the matrix have value $m_i \in \{-1, 0, 1\}$, where -1 represents unknown cell, 0 represents free cells, and 1 represents occupied cells.
- Log-odds m_t: the map cells m_i can be modeled as independent Bernoulli random variables. Given occupancy measurements $z_{0:t}$, the distribution of m_i is

$$p(m_i|z_{0:t}) = \begin{cases} \gamma := p(m_i = 1|z_{0:t}, x_{0:t}) & , m_i = 1 \\ 1 - \gamma & , m_i = 0 \end{cases} \tag{14}$$

Using Bayes rule we have

$$\begin{aligned} \gamma_{i,t} &= p(m_i = 1|z_{0:t}, x_{0:t}) \\ &= \frac{1}{\eta_t} p_h(z_t|m_i = 1, x_t) p(m_i = 1|z_{0:t-1}, x_{0:t-1}) \\ &= \frac{1}{\eta_t} p_h(z_t|m_i = 1, x_t)\gamma_{i,t-1}. \end{aligned} \tag{15}$$

Define odds ratio of a binary random variable m_i updated over time as

$$\begin{aligned} o(m_i|z_{0:t}, x_{0:t}) &:= \frac{\gamma_{i,t}}{1 - \gamma_{i,t}} \\ &= \frac{p_h(z_t|m_i = 1, x_t)}{p_h(z_t|m_i = 0, x_t)} \frac{\gamma_{i,t-1}}{1 - \gamma_{i,t-1}} \\ &= g_h(z_t|m_i, x_t) o(m_i|z_{0:t-1}, x_{0:t-1}), \end{aligned} \tag{16}$$

where for simplicity, we specify the value of $p_h(z_t = 1|m_i = 1, x_t)$ to be 80% as the trust for LIDAR measurement. Under this condition we have $g_h(z_t|m_i, x_t) = 4$.Take the log for both side the equation:

$$\begin{aligned} \lambda(m_i|z_{0:t}, x_{0:t}) &:= \log o(m_i|z_{0:t}, x_{0:t}) \\ &= \log o(m_i|z_{0:t}, x_{0:t}) + \log g_h(z_t|m_i, x_t) \\ &= \lambda(m_i|z_{0:t}, x_{0:t}) + \log g_h(z_t|m_i, x_t), \end{aligned} \tag{17}$$

which is the update rule for log-odds. The grid map can be derived from log-odds simply by thresholding on 0.

- Texture: texture is created if RGBD images provided. Disparity is given to calculate depth information by:

$$\text{depth} = \frac{1.03}{-0.00304 \times \text{disparity} + 3.31}. \tag{18}$$

The transform from 2D RGB image to world frame can be derived from equation (12) and (13). The floor pixel location can be obtained by thresholding on z axis.

C. Robot and Particles

A python class Robot() is created to handle all functions and properties of a robot body. The basic properties of a robot are listed as follows:

- Robot state: robot state is represented as $s = (p, \theta) \in SE(2)$, where $p = (x, y) \in \mathbb{R}^2$ is the position and $\theta \in (-\pi, \pi]$ is the orientation (yaw angle). Its state is decided by the best particle which has the largest weight.
- Particles list: robot saves a list with all particles.
- Robot and sensor poses: each sensor has its location with regard to the body frame. In order to transform the coordinates to world frame, robot and sensor poses need to be calculated real-time or beforehand.

A python class Particle() is created handle all functions and properties of a robot body. The basic properties of a robot are listed as follows:

- Particle state: same as robot state.
- Particle weight: each particle in the particles list has its own weight, which is updated based on the map correlation.
- Particle map: each particle will have its own map to increase the accuracy for visited places. Loop closure can be achieved based on this information. In order

to reduce computation and space, only the occupied cell in the map are saved in a list, instead of saving all the map information in each particle.

Given control input, the particles will update their localization based on the motion model. Gaussian noise is introduced during this step.

And for map correlation, instead of just shifting the robot in x and y axis, I also perform a shift in ω space to find the best particle state. This will slightly improve the performance sometimes.

IV. Results and Analysis

This chapter will display some results showing the effectiveness of algorithm in practical uses. Best result on each dataset is shown first on next sub-section, and then some intermediate results and associated analysis follows.

A. Results on Trainin/Test Set

Results on each dataset are shown in Fig.1, 2 and 3. The purple pixels are unknown cells (undetected by lisar), yellow pixels are occupied cells (wall), cyan pixels are free cells (corridors and rooms). Red triangle represents the robot and its orientation is pointed by one of the angle. Blue dots represent all the particles. Easy to see that the particles have reasonable locations (at the body of the robot). And loop closure is achieved. Images of the SLAM over time has been made to video and included in Github website and the Gradescope as .gif files.

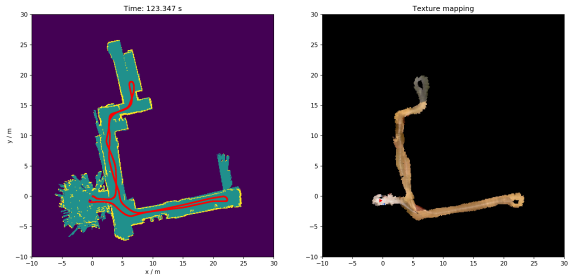


Fig. 1: Result on dataset 20 with texture.

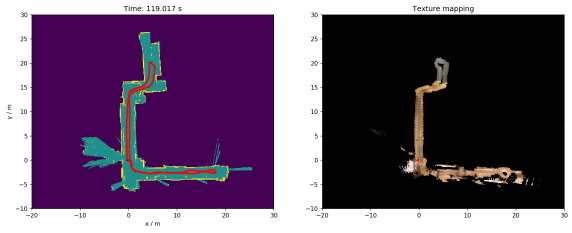


Fig. 2: Result on dataset 21 with texture.

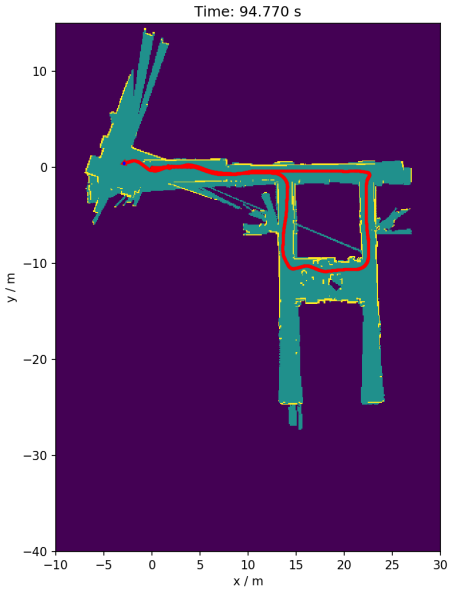


Fig. 3: Result on dataset 23 without texture.

B. Texture Mapping

The difficulty of texture mapping is to find the area of floor based on the depth information and data association. Correctness of theses steps can be verified in Fig. 4.

C. Importance of Introducing Noises

As mentioned before, Gaussian noises are added during motion prediction step. Fig. 5, 6 and 7 show the results of adding no noises compared with adding Gaussian noises during prediction. As we can see the performance is much better when noises are introduced.

V. Future Work

As we can see on Fig. 3, at the end of the trajectory, orientation of the robot is wrongly estimated. This might be solved by introducing a bigger noise or let map correlation performed at a smaller resolution. The trade-off might be more computation complexity, though.

References

[1] Durrant-Whyte, H.; Bailey, T. (2006). "Simultaneous localization and mapping: part I". IEEE Robotics & Automation Magazine. 13 (2): 99–110. doi:10.1109/mra.2006.1638022. ISSN 1070-9932.

[2] Bailey, T.; Durrant-Whyte, H. (2006). "Simultaneous localization and mapping (SLAM): part II". IEEE Robotics & Automation Magazine. 13 (3): 108–117. doi:10.1109/mra.2006.1678144. ISSN 1070-9932

[3] Cadena, Cesar; Carlone, Luca; Carrillo, Henry; Latif, Yasir; Scaramuzza, Davide; Neira, Jose; Reid, Ian; Leonard, John J. (2016). "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". IEEE Transactions on Robotics. 32 (6): 1309–1332. doi:10.1109/tro.2016.2624754. ISSN 1552-3098

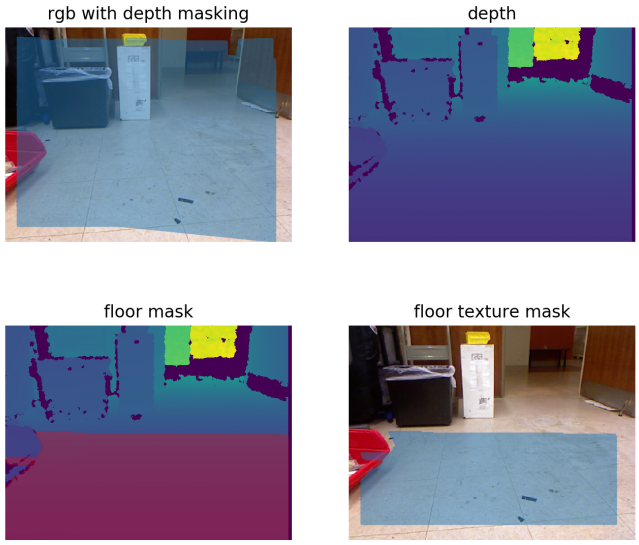
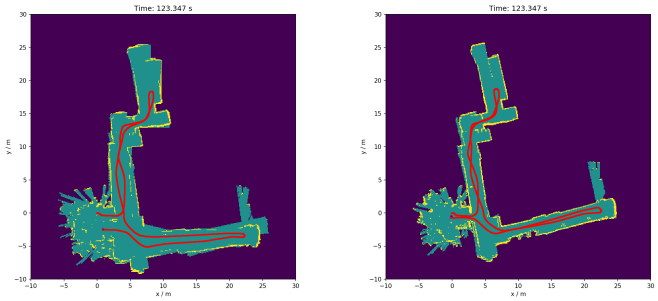


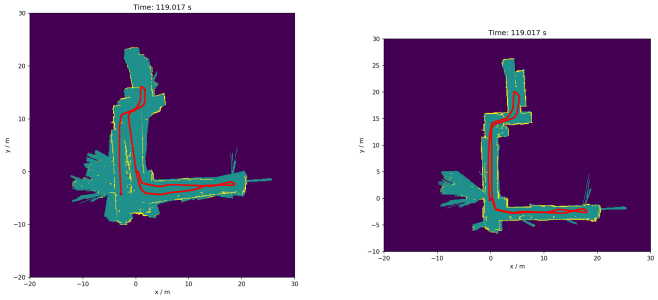
Fig. 4: Verify the correctness of texture mapping. Top left shows the original RGB image with the blue mask represents the area with depth information. Top right is the depth value calculate by (18). Bottom left is the floor area by thresholding the z axis in world frame below 0.15 m. Bottom right is the floor texture pixels by mapping the depth mask back to RGB image.



(a) No Gaussian noises (b) With Gaussian noises

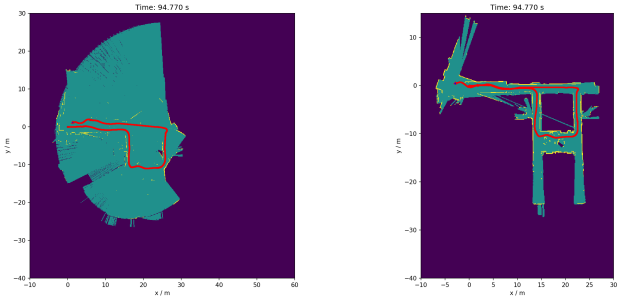
Fig. 5: Results on dataset 20

[4] Mountney, P.; et al. (Stoyanov, D.; Davison, A.; Yang, G-Z.) (2006). "Simultaneous Stereoscope Localization and Soft-Tissue Mapping for Minimal Invasive Surgery". MICCAI. Lecture Notes in Computer Science. 1: 347–354. doi:10.1007/11866565_43



(a) No Gaussian noises (b) With Gaussian noises

Fig. 6: Results on dataset 21



(a) No Gaussian noises (b) With Gaussian noises

Fig. 7: Results on dataset 23