Particle Filter SLAM and Texture Mapping Based on Differential-Drive Robot

Yang Jiao
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
y4jiao@ucsd.edu

I. Introduction

The Simultaneous Localization and Mapping (SLAM) problem is a fundamental problem for mobile robot autonomy. It allows the robot to gather the information of where it locates and what is the surrounding environment. It is also the premier step for a robot to interact with the world.

One implementation to solve the SLAM problem from a statistic perspective is to apply *Bayes filter*. Bayes filter is a class of filter that estimates probability of *states* using Bayes theorem. The Bayes filter approach is generally separate to *prediction* step and *update* step. In the prediction step we increase the randomness by introducing environment noise, and in update step we decrease randomness by adopting sensory observation. By applying the prediction and update steps alternatively, Bayes filter can give more accurate estimation of the states over time, since there is more historical information captured.

More specifically, we explored *particle filter* in this project, which is a kind of Bayes filter. Particle filter approximately calculate the distribution updates in Bayes filter by discretized the continuous distributions with a certain quantity of particles. In our implementation, we solved the localization problem using particle filter and created a 2D occupancy grid map based on the estimated robot trajectory. We also applied the localization and mapping results together with RGBD camera input to construct a texture map of the floor surface.

This report is organized as follows. Section II describes the main idea of how we formulate the SLAM and texture mapping problems. Section III presents the detailed modeling and implementation in our approach. Section IV discusses the results and concludes.

II. Problem Formulation

A. SLAM Formulation

Similar to project 1, in the SLAM problem we also rely on the robot sensory measurements and control inputs, which are described by *observation model* and *motion model* respectively. Again, we denote the observation model by $h(\cdot)$ and motion model by $f(\cdot)$ (Eq. 1).

$$\begin{aligned} \mathbf{z}_t &= h\left(\mathbf{x}_t, \mathbf{m}\right) \\ \mathbf{x}_{t+1} &= f\left(\mathbf{x}_t, \mathbf{u}_t\right) \end{aligned} \tag{1}$$

where the robot state, control input, sensor measurements at time t, and the map are denoted by $\mathbf{x}_t$. $\mathbf{u}_t$, $\mathbf{z}_t$, and $\mathbf{m}$ respectively.

A SLAM problem can be mathematically formulated as in Eq. 2, where we want to find the robot trajectory $\mathbf{x}_{1:T}$ and the map $\mathbf{m}$ that best matches with the motion model and observation model.

$$\min_{\mathbf{x}_{1:T}, \mathbf{m}} \sum_{t=1}^{T} \left\|\mathbf{z}_t - h\left(\mathbf{x}_t, \mathbf{m}\right)\right\|_2^2 + \sum_{t=0}^{T-1} \left\|\mathbf{x}_{t+1} - f\left(\mathbf{x}_t, \mathbf{u}_t\right)\right\|_2^2 \tag{2}$$

Alternatively, from the probability perspective, we introduce random noise to the motion and observation model, such that we can express them using probability density functions (pdf) as Eq. 3.

$$\begin{aligned} \mathbf{z}_t &= h\left(\mathbf{x}_t, \mathbf{m}_t, \mathbf{v}_t\right) \sim p_h(\cdot|\mathbf{x}_t, \mathbf{m}) \\ \mathbf{x}_{t+1} &= f\left(\mathbf{x}_t, \mathbf{u}_t, \epsilon_t\right) \sim p_f(\cdot|\mathbf{x}_t, \mathbf{u}_t) \end{aligned} \tag{3}$$

where $\mathbf{v}_t$ is the observation noise and ϵ_t is the motion noise. $\mathbf{m}_t$ represents the current map at time t. Note that Eq. 3 is also based on the *Markov assumptions*, such that

- the state $\mathbf{x}_{t+1}$ only depends on the previous input $\mathbf{u}_t$ and state $\mathbf{x}_t$ and is independent from all other history states and control inputs;
- the observation $\mathbf{z}_t$ only depends on the state $\mathbf{x}_t$.

Consequently, we can re-formulate the SLAM problem in Eq. 2 from the probability view. In this case, we want to derive point estimators for $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{m}}_t$ (e.g., using *maximum a posteriori* which is the method in *Bayes filter*). Section IIIA will discuss the implementation of such method in detail.

B. Texture Mapping Problem

Similar to the panorama construction problem in project 1, texture mapping is to map from RGBD image to produce a 2D color map of the floor surface. Here we consider the problem in Eq. 4.

$$\psi : (rgbi, rbgj), d, \boldsymbol{x}_t \mapsto (i_m, j_m) \tag{4}$$

where d is the depth obtained from the depth camera for the pixel location $(rgbi, rgbj)$. Note that we also need to use the conversion to map from a pixel (i, j) on the disparity image to $(rgbi, rgbj)$ on the RGB image for completeness. This part is presented more detailed in Section IIIB.

III. Technical Approach

A. Particle Filter SLAM

1) Motion and Observation Models: In this project, we consider the motion and observation model for a differential-drive robot with LiDAR scans to observe the environment and encoder and IMU odometry to obtain the control input. In this case, Eq. 3 become:

$$\mathbf{x}_{t+1} = f_d\left(\mathbf{x}_t, \tilde{\mathbf{u}}_t\right) := \mathbf{x}_t + \tau_t \left[\begin{array}{c} v_t \cos\left(\theta_t\right) \\ v_t \sin\left(\theta_t\right) \\ \omega_t \end{array}\right] \tag{5}$$

where the robot state is represented by $\mathbf{x}_t = [x_t, y_t, \theta_t]^T$, and the control input $\mathbf{u}_t = [v_t, \omega_t]^T$. We model the motion noise as an additive term to the control input $\tilde{\mathbf{u}}_t = \mathbf{u}_t + \boldsymbol{\epsilon}_t$, with 2D Gaussian noise $\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \left[\begin{array}{cc} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{array}\right])$.

As we are going to show in Section IIIAii, the observation model only uses in the update step and is always normalized to get the expression for posteriori. Thus, we can find the term to which $p_h(\cdot)$ is proportional instead of deriving the exact expression of $p_h(\cdot)$. Here we define the observation model in the following way:

$$p_h(\mathbf{z}|\mathbf{x}, \mathbf{m}) \propto corr(r(\mathbf{z}, \mathbf{x}), \mathbf{m}) \tag{6}$$

where $\mathbf{y} = r(\mathbf{z}, \mathbf{x})$ denotes the projection from the LiDAR frame to the world frame. For a single ray from the LiDAR scans in one time stamp $\mathbf{z}_s^{(i)} = (z_r^{(i)}, z_\alpha^{(i)})$ (2D spherical coordinates), we can transform it into Cartesian coordinates in LiDAR frame by

$$\mathbf{z}_c^{(i)} = (z_r^{(i)} \cos z_\alpha^{(i)}, z_r^{(i)} \sin z_\alpha^{(i)}) \tag{7}$$

Then transform $\mathbf{z}_c^{(i)}$ to the world frame based on the robot state $\mathbf{x}_t$.

The correlation function is implemented as:

$$corr(\mathbf{y}, \mathbf{m}) = \sum_i \mathbb{1}\{y_i = m_i\} \tag{8}$$

where

$$\mathbb{1}\left\{y_i = m_i\right\} = \begin{cases} 1, & \text{if } y_i = m_i, \\ 0, & \text{else.} \end{cases} \tag{9}$$

2) Particle Filter: We adopted the particle filter approach to estimate the robot trajectory. *Particle filter* is a kind of filter in the class of Bayes filter, where we use a discretized distribution (sampled from particles) with probability mass function (pmf) to approximate the continuous distribution of the robot state $p(\mathbf{x}_t|\mathbf{z}_{0:t}, \mathbf{u}_{0:t-1}, \mathbf{m})$. Consider a particle filter with N_p particles. The robot state at each time stamps will be represented by N_p state hypotheses. Denote the hypotheses by $\mu_{t|t}[k], \forall k = 0, \ldots, N_p - 1$ on the state $\mathbf{x}_t$ with confidence $\alpha_{t|t}[k]$ respectively, where

$$\sum_{k=0}^{N_p-1} \alpha_{t|t}[k] = 1 \tag{10}$$

In the SLAM problem, we are usually given the initial state of the robot $\mathbf{x}_0 = [0, 0, 0]^T$. Hence, we can simply initialize all of the particles with $\mu_{0|0}[k] = \mathbf{x}_0$ and $\alpha_{0|0} = 1/N_p$. Following the same formulation as Bayes filter, we have the *prediction* step and *update* step for particle filter as follows. Since particle filter uses a discretized approximation, *resampling* is performed when certain condition is met.

In the prediction step, the motion model is applied with designed noise:

$$\boldsymbol{\mu}_{t+1|t}[k] = f\left(\boldsymbol{\mu}_{t|t}[k], \mathbf{u}_t + \boldsymbol{\epsilon}_t\right) \tag{11}$$

where the motion model $f(\cdot)$ is given in Eq. 5, and we keep the weights unchanged for all hypotheses (i.e., $\alpha_{t+1|t}[k] = \alpha_{t|t}[k]$).

In the update step, the particle weights are updated based on the observation model (Eq. 6). We have

$$\alpha_{t+1|t+1}[k] \propto \operatorname{corr}\left(\mathbf{y}_{t+1}[k], \mathbf{m}\right) \alpha_{t+1|t}[k] \tag{12}$$

Then $\alpha_{t+1|t+1}[k]$ should be normalized such that the weights satisfy Eq. 10 (i.e., it represents a proper pmf).

In general, we should have $\mu_{t+1|t+1}[k] = \mu_{t+1|t}[k]$, where the particles remain unchanged after utilizing the observation information. However, we imposed grid perturbation (x, y) and yaw-angle perturbation around each particle when evaluating the map correlation, and replace the current particle by the surrounding pose with highest map correlation. This implementation will achieve the same results as using more particles but should be more efficient as we reduce N_p.

As $\alpha_{t+1|t+1}[k]$ represents the confidence on a particle $\mu_{t+1|t+1}[k]$ after each update step, we can define the number of effective particles N_{eff}:

$$N_{\text{eff}} := \frac{1}{\sum_{k=1}^{N}\left(\alpha_{t+1|t+1}[k]\right)^2} \tag{13}$$

An intuition on Eq. 13 is that we always have number of effective particles in the range from 1 up to N_p. There are two extreme cases:

- when we have $\alpha_{t+1|t+1}[k_i] \to 1$ for some k_i and all other particles have very low weights, then $N_{eff} \to 1$, which means there is approximately only 1 effective particle $\mu_{t+1|t+1}[k_i]$;
- when we have all particles with equal weights $1/N_q$, we can obtain $N_{eff} = N_q$. Since all particles are equally important, they are all considered as effective particles.

Thus, we can design a certain threshold (e.g., $N_{th} = N_q/10$). When $N_{eff} \leq N_{th}$, we perform resampling on the set $\{\mu_{t+1|t+1}[k], \alpha_{t+1|t+1}[k]\}$. All of the particles will have weights $1/N_q$ after resampling so that we have $N_{eff} = N_q$.

Note that N_p remains unchanged for all time stamps. We simply choose the particle with the highest weight (after each update step) to represent $\mathbf{x}_t$.

3) Mapping: Note that the implementation with particle filter only takes care for robot states and thus only completes the localization problem. Here we discuss the mapping method as we already get the robot trajectory $\mathbf{x}_{0:T}$, where each state is represented by the particle with the highest weight.

The map is represented by a 2D occupancy grid map, where each cell m_i has a certain probability of either being occupied or free:

$$m_i = \begin{cases} +1 \text{ (Occupied)} & \text{with prob. } \gamma_{i,t} \\ -1 \text{ (Free)} & \text{with prob. } 1-\gamma_{i,t} \end{cases} \tag{14}$$

The probability $\gamma_{i,t} := p\left(m_i = 1 \mid \mathbf{z}_{0:t}, \mathbf{x}_{0:t}\right)$ is updated based on information from LiDAR scan over time. By Markov assumptions we know that each observation is independent. With the usage of log-odds, we have

$$\begin{aligned} \Delta\lambda_{i,t} &= \log \frac{p\left(m_i = 1 \mid \mathbf{z}_t, \mathbf{x}_t\right)}{p\left(m_i = -1 \mid \mathbf{z}_t, \mathbf{x}_t\right)} \\ &= \begin{cases} +\log 4 & \text{if } \mathbf{z}_t \text{ indicates } m_i \text{ is occupied} \\ -\log 4 & \text{if } \mathbf{z}_t \text{ indicates } m_i \text{ is free} \end{cases} \end{aligned} \tag{15}$$

where we get $\log 4$ because we model the LiDAR as an 80%-correct sensor (odds ratio equals to 4). Hence, we can update the map after each LiDAR scan by $\lambda_{i,t+1} = \lambda_{i,t} \pm \log 4$. Besides, $\lambda_{i,t}$ is also truncated in a certain range $\lambda_{MIN} \leq \lambda_{i,t} \leq \lambda_{MAX}$ to avoid overconfident estimation.

B. Texture Mapping

The texture mapping is performed using RGBD camera data (RGB images and disparity images). We describe the pipeline for texture mapping from one RGBD image as follows.

1) The depth camera and RGB camera do not share the exactly same location. Given a pixel (i, j) in the disparity image, we can associate it with a pixel in the RGB image by:

$$\begin{aligned} rgbi =& (526.37i + (-4.5 \times 1750.46)dd \\ & + 19276.0)/585.051 \\ rgbj =& (526.37j + 16662)/585.051 \end{aligned} \tag{16}$$

where $dd = (-0.00304d + 3.31)$ and d is the disparity value at pixel (i, j).

2) For all pairs of depth-RGB pixels, we can obtain the coordinates in physical unit of the pixel location in the camera optical frame $p^o = [X_o, Y_o, Z_o]^T$, where $Z_o = depth = \frac{1.03}{dd}$, and X_o, Y_o can be obtained from

$$\begin{aligned} \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} &= K^{-1} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} * \text{depth} \\ K &= \begin{bmatrix} fs_u & fs_\theta & c_v \\ 0 & fs_v & c_u \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{17}$$

3) As the RGBD camera observation is noisy, we only trust the image with depth in certain range $Z_{MIN} \leq Z_o \leq Z_{MAX}$, and abandon all other pixels;

4) The valid points p^o are transformed from camera optical frame to world frame, given the relative pose of the depth camera to the body frame and the robot state $\mathbf{x}_t$;

$$p^w = {}_wT_o(\mathbf{x}_t)p^o \tag{18}$$

where $p^w = [X_w, Y_w, Z_w]^T$ is the same point represented in the world frame.

5) The points that associated to the floor are selected (given the floor has coordinates with $z_w = 0m$). We set a certain threshold (such as $z_{th} = 0.01m$) and select the points with $|Z_w| \leq z_{th}$;

6) The selected floor points are mapped to occupancy map pixel location, and the associated RGB values are assigned to the texture map.

IV. Results

There are 4 kinds of sensors mounted on the differential-drive robot, encoder, IMU, LiDAR, and RGBD camera. For both SLAM and texture mapping tasks, we are provided with two datasets with all 4 kinds of sensor data. The robot travelled in the same indoor environment in the two dataset. In the first dataset, the robot navigated around the hallways on the flat ground. However, the robot steered up and down on some slopes in the second dataset, where more environment noise was introduced.

Encoder and IMU measured the robot linear and angular velocities respectively. Denote the encoder counts $\mathbf{C}_t = [FR, FL, RR, RL]$, corresponding to the front-right, front-left, rear-right, and rear-left wheels' counts at time t. Then the linear velocity at the center of the robot chassis is given by

$$v_t = \frac{\sum \mathbf{C}_t}{4} \cdot l \tag{19}$$

where $l = 0.0022m$ as the wheel travels $l = \frac{\pi \times \text{wheel diameter}}{360}$ per tic.

We can direct obtain the angular velocity ω_t by the yaw rate reading from IMU. The LiDAR scans started from $z_\alpha = -135\,\text{deg}$, and the conversion of LiDAR input has been given in Eq. 7. The intrinsic parameters of the depth camera are given as $fs_u = fs_v = 585.05108211$, $fs_\theta = 0$, $c_v = 242.94140713$, and $c_u = 315.83800193$. The sensor data is aligned by the nearest time stamps.

We define the body frame of the robot at the center of the chassis, which gives the LiDAR location in the body frame as $p_l^b = [0.13323, 0, 0.51435]^T(m)$, and the camera location $p_c^b = [0.16766, 0, 0.38001]^T(m)$. Note that the camera is $18\,\text{deg}$ tilt (pitch angle).

Section IVA and IVB discuss the results from particle filter SLAM and texture mapping respectively, while the complete graphical results are presented in Appendix.

A. Particle Filter SLAM

We downsampled the LiDAR data by a rate of 5 (i.e., 4 LiDAR frames are skipped before running one update step). This is because the LiDAR data were recorded in a frequency of 40Hz, but the update step in particle filter runs much slower than the prediction step. It is not necessary to use all LiDAR data to frequently run update steps, which is also very time expensive.

During the testing of particle filter SLAM, the following parameters have the most significant effect on the filter performance:

- The number of particles N_p: in general, the larger N_p the better performance. Here we use N_p in the scale of 100-200;
- The scale of the noise variance: we found the noise and especially the angular velocity noise ϵ_ω has a huge impact on the performance. We also adopted different ϵ_ω for two datasets. The first one with variance $\sigma_\omega = 0.02$ and the second one with variance $\sigma_\omega = 0.05$;
- The threshold for resampling N_{th}: here we used the default $N_{th} = N_p/10$. As some other classmates reported raising N_{th} (i.e., resampling more frequently) improves the performance, I did not observe a similar trend, and it might even hurt the results. My hypothesis is that the choice of N_{th} is related to the noise variance. If we use a larger σ_ω, then we might want to resample more frequently to compensate the randomness.

As we can see from the figures in the Appendix, the particle filter SLAM effectively improved the localization and thus mapping results comparing to the dead reckoning ones. Since the update step includes the historical observations (the current map), we can see the occupancy map has fewer rotated or skewed hallways.

B. Texture Mapping

On the other hand, the texture mapping results are not ideal. One possible reason is that the robot camera cannot achieve floor texture where is too close to the robot. Thus, we may see some circular shape without texture information plotted. Some possible improvement methods include to chop the data outside the occupancy map and the increase the update rate in particle filter SLAM so that there will be more available robot states for texture mapping. Besides, the current $Z_{MAX} = 10m$ which could result in losing some useful pixels.

APPENDIX

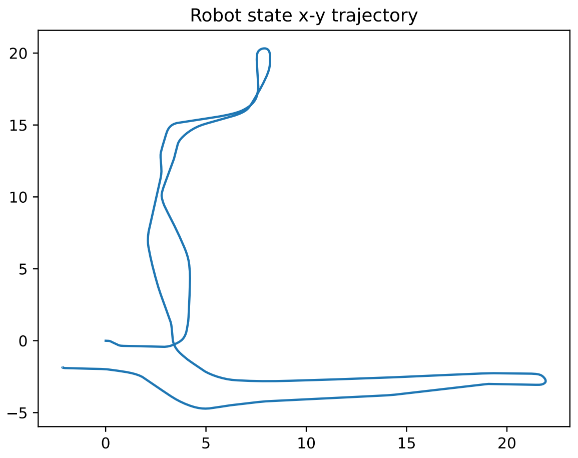


Fig. 1: Dead reckoning trajectory on dataset 20

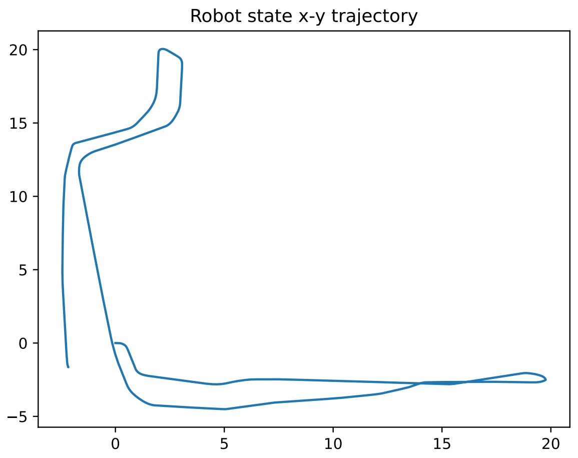


Fig. 2: Dead reckoning trajectory on dataset 21

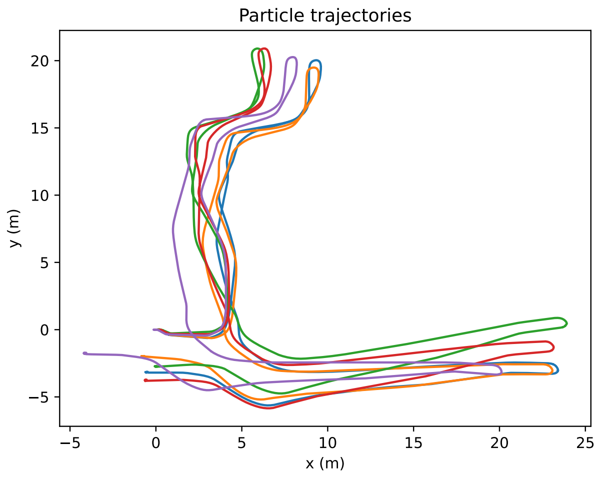


Fig. 3: Prediction-only trajectory on dataset 20 with 5 particles

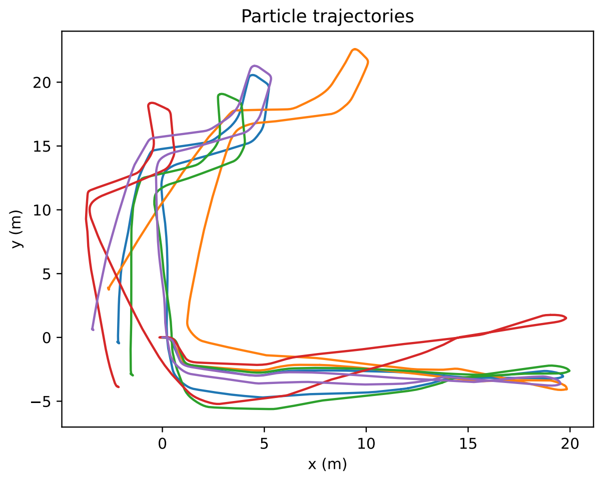


Fig. 4: Prediction-only trajectory on dataset 21 with 5 particles. Note that the particles spreaded more away from each other than the prediction-only results for dataset 20, because a larger noise variance was imposed.

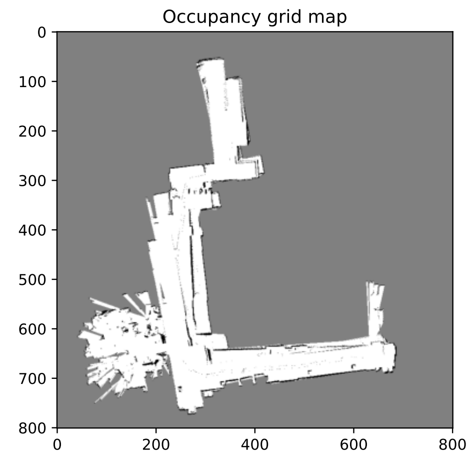


Fig. 5: Map from dead reckoning robot trajectory on dataset 20

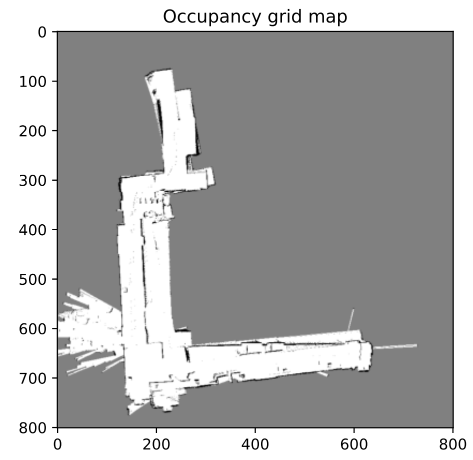
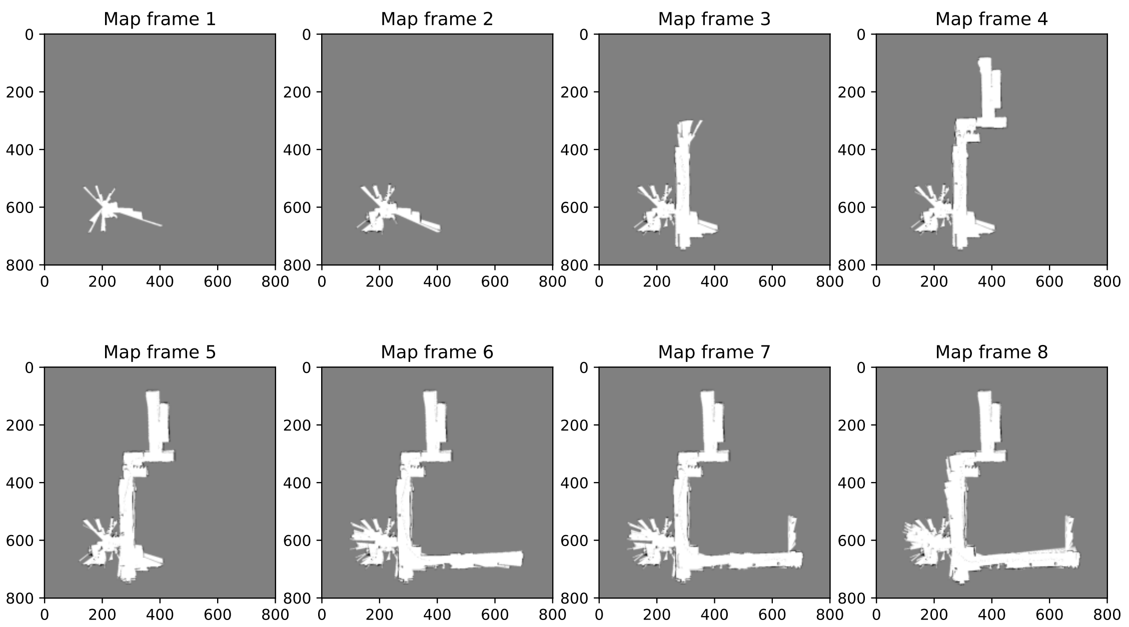
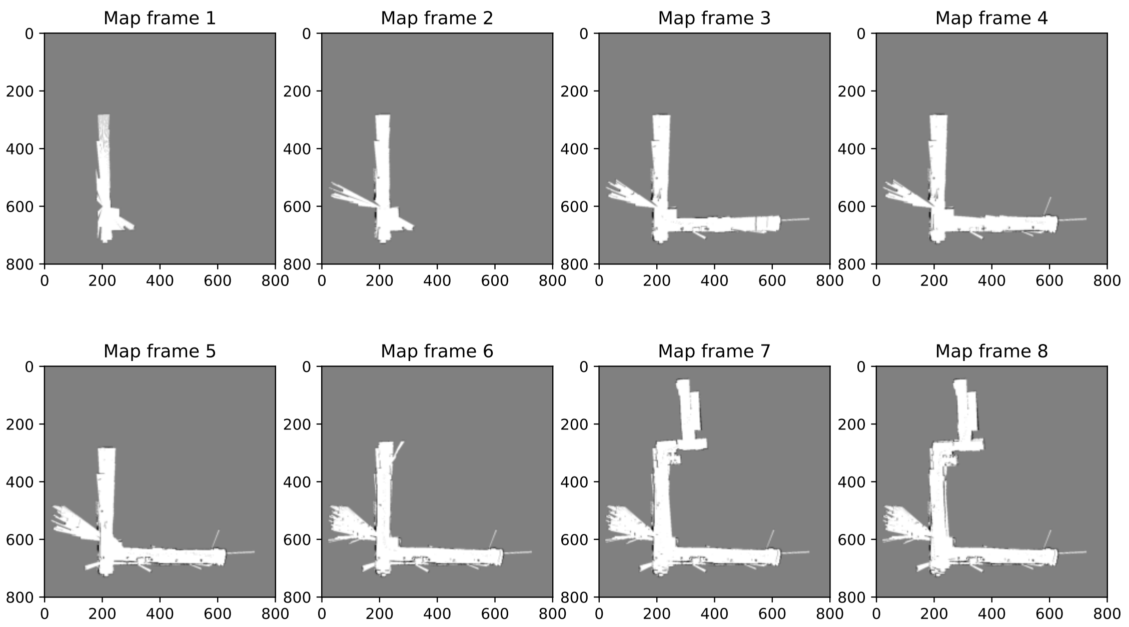


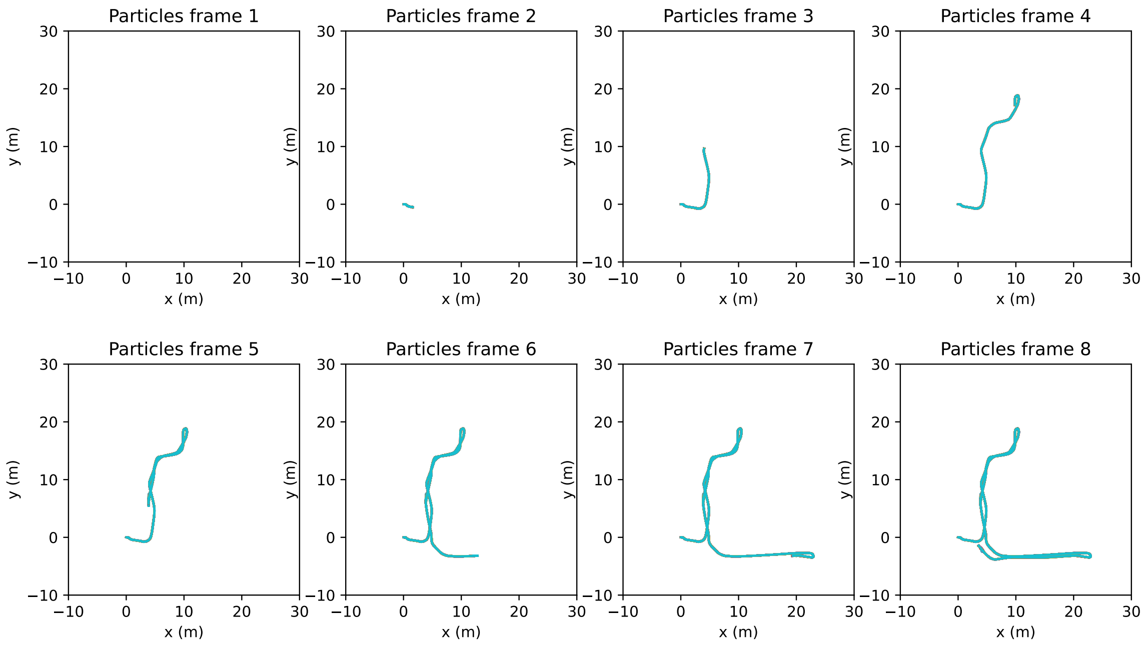
Fig. 6: Map from dead reckoning robot trajectory on dataset 21



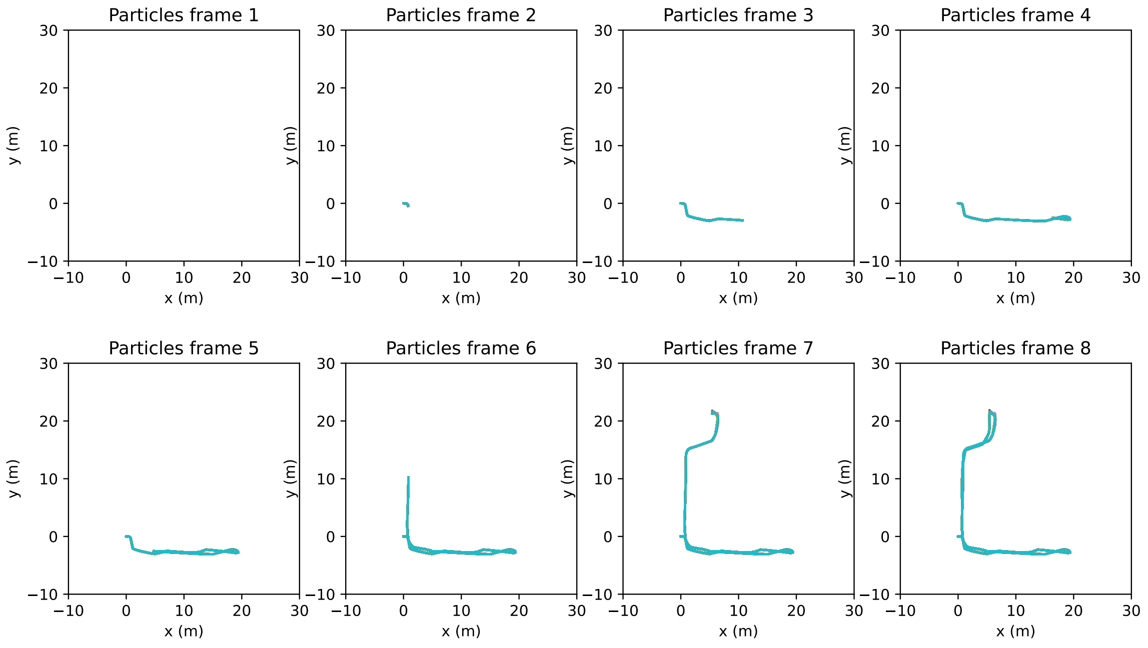
(a) Particle filter SLAM mapping result on dataset 20 ($N_p = 200$)



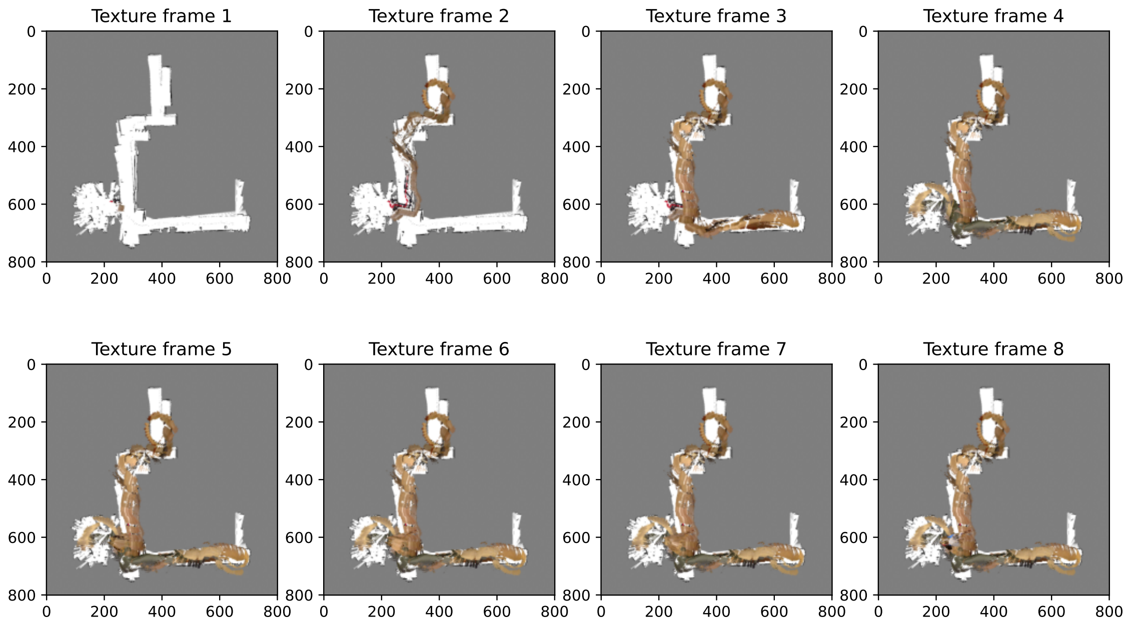
(b) Particle filter SLAM mapping result on dataset 21 ($N_p = 200$)



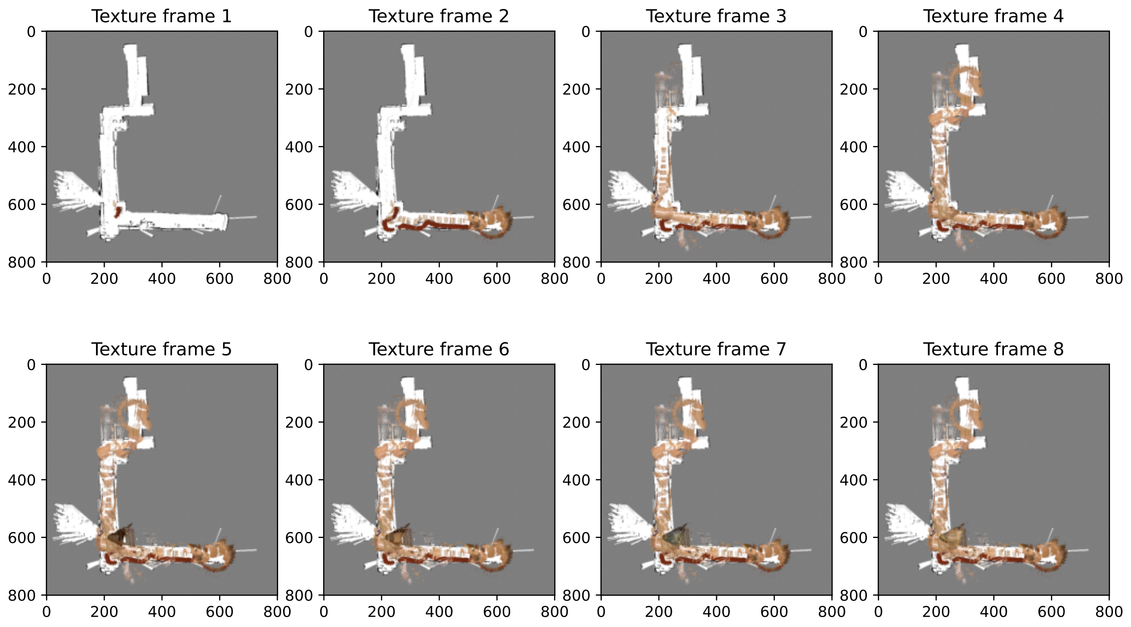
(a) Particle filter SLAM trajectory result on dataset 20 ($N_p = 200$)



(b) Particle filter SLAM trajectory result on dataset 21 ($N_p = 200$)



(a) Texture mapping result on dataset 20



(b) Texture mapping result on dataset 21